

Keysight InfiniiVision 2000 X-Series Oscilloscopes



Programmer's
Guide

Notices

© Keysight Technologies, Inc. 2005-2015

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 Keysight Technologies, Inc. as governed by United States and international copyright laws.

Revision

Version 02.39.0000

Edition

July 15, 2015

Available in electronic format only

Published by:

Keysight 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, Keysight 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. Keysight 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 Keysight 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 License

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.

U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at www.keysight.com/find/sweula. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

Safety Notices

CAUTION

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

WARNING

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

In This Book

This book is your guide to programming the 2000 X-Series oscilloscopes:

Table 1 InfiniiVision 2000 X-Series Oscilloscope Models

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

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

- **Chapter 1**, “What’s New,” starting on page 25, describes programming command changes in the latest version of oscilloscope software.
- **Chapter 2**, “Setting Up,” starting on page 41, describes the steps you must take before you can program the oscilloscope.
- **Chapter 3**, “Getting Started,” starting on page 49, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- **Chapter 4**, “Commands Quick Reference,” starting on page 63, is a brief listing of the 2000 X-Series oscilloscope commands and syntax.

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

- **Chapter 5**, “Common (*) Commands,” starting on page 119, describes commands defined by the IEEE 488.2 standard that are common to all instruments.
- **Chapter 6**, “Root (:) Commands,” starting on page 145, describes commands that reside at the root level of the command tree and control many of the basic functions of the oscilloscope.
- **Chapter 7**, “:ACQuire Commands,” starting on page 181, describes commands for setting the parameters used when acquiring and storing data.
- **Chapter 8**, “:BUS<n> Commands,” starting on page 195, describes commands that control all oscilloscope functions associated with the digital channels bus display.
- **Chapter 9**, “:CALibrate Commands,” starting on page 205, describes utility commands for determining the state of the calibration factor protection button.

- **Chapter 10**, “:`:CHANnel<n>` Commands,” starting on page 215, describes commands that control all oscilloscope functions associated with individual analog channels or groups of channels.
- **Chapter 11**, “:`:DEMO` Commands,” starting on page 237, describes commands that control the education kit (Option EDU) demonstration signals that can be output on the oscilloscope’s Demo 1 and Demo 2 terminals.
- **Chapter 12**, “:`:DIGItal<d>` Commands,” starting on page 243, describes commands that control all oscilloscope functions associated with individual digital channels.
- **Chapter 13**, “:`:DISPlay` Commands,” starting on page 251, describes commands that control how waveforms, graticule, and text are displayed and written on the screen.
- **Chapter 14**, “:`:DVM` Commands,” starting on page 265, describes commands that control the optional DSOXDVM digital voltmeter analysis feature.
- **Chapter 15**, “:`:EXTernal` Trigger Commands,” starting on page 273, describes commands that control the input characteristics of the external trigger input.
- **Chapter 16**, “:`:FUNCtion` Commands,” starting on page 279, describes commands that control math waveforms.
- **Chapter 17**, “:`:HARDcopy` Commands,” starting on page 297, describes commands that set and query the selection of hardcopy device and formatting options.
- **Chapter 18**, “:`:LISTer` Commands,” starting on page 315, describes commands that turn on/off the Lister display for decoded serial data and get the Lister data.
- **Chapter 19**, “:`:MARKer` Commands,” starting on page 319, describes commands that set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
- **Chapter 20**, “:`:MEASure` Commands,” starting on page 335, describes commands that select automatic measurements (and control markers).
- **Chapter 21**, “:`:MTEST` Commands,” starting on page 379, describes commands that control the mask test features provided with Option LMT.
- **Chapter 22**, “:`:POD` Commands,” starting on page 413, describes commands that control all oscilloscope functions associated with groups of digital channels.
- **Chapter 23**, “:`:RECall` Commands,” starting on page 419, describes commands that recall previously saved oscilloscope setups, reference waveforms, or masks.
- **Chapter 24**, “:`:SAVE` Commands,” starting on page 427, describes commands that save oscilloscope setups, screen images, and data.
- **Chapter 25**, “:`:SBUS<n>` Commands,” starting on page 449, describes commands that control oscilloscope functions associated with the serial decode bus and serial triggering.

- **Chapter 26**, “:SEARch Commands,” starting on page 533, describes commands that control oscilloscope functions associated with searching for waveform events.
- **Chapter 27**, “:SYSTem Commands,” starting on page 565, describes commands that control basic system functions of the oscilloscope.
- **Chapter 28**, “:TIMEbase Commands,” starting on page 579, describes commands that control all horizontal sweep functions.
- **Chapter 29**, “:TRIGger Commands,” starting on page 591, describes commands that control the trigger modes and parameters for each trigger type.
- **Chapter 30**, “:WAVeform Commands,” starting on page 629, describes commands that provide access to waveform data.
- **Chapter 31**, “:WGEN Commands,” starting on page 665, describes commands that control waveform generator (Option WGN) functions and parameters.
- **Chapter 32**, “:WMEMory<r> Commands,” starting on page 695, describes commands that control reference waveforms.
- **Chapter 33**, “Obsolete and Discontinued Commands,” starting on page 705, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- **Chapter 34**, “Error Messages,” starting on page 753, lists the instrument error messages that can occur while programming the oscilloscope.

The command descriptions in this reference show upper and lowercase characters. For example, :AUToscale indicates that the entire command name is :AUTOSCALE. The short form, :AUT, is also accepted by the oscilloscope.

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

- **Chapter 35**, “Status Reporting,” starting on page 761, describes the oscilloscope’s status registers and how to check the status of the instrument.
- **Chapter 36**, “Synchronizing Acquisitions,” starting on page 783, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- **Chapter 37**, “More About Oscilloscope Commands,” starting on page 793, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- **Chapter 38**, “Programming Examples,” starting on page 803.

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

- See Also**
- For more information on using the SICL, VISA, and VISA COM libraries in general, see the documentation that comes with the Keysight IO Libraries Suite.
 - For information on controller PC interface configuration, see the documentation for the interface card used (for example, the Keysight 82350A GPIB interface).
 - For information on oscilloscope front-panel operation, see the *User's Guide*.
 - For detailed connectivity information, refer to the *Keysight Technologies USB/LAN/GPIB Connectivity Guide*. For a printable electronic copy of the *Connectivity Guide*, direct your Web browser to www.keysight.com and search for "Connectivity Guide".
 - For the latest versions of this and other manuals, see:
<http://www.keysight.com/find/2000X-Series-manual>

Contents

In This Book / 3

1 What's New

What's New in Version 2.39 / 26
What's New in Version 2.38 / 27
What's New in Version 2.30 / 28
What's New in Version 2.20 / 30
What's New in Version 2.10 / 32
What's New in Version 2.00 / 33
What's New in Version 1.20 / 34
What's New in Version 1.10 / 35
Version 1.00 at Introduction / 36
Command Differences From 7000B Series Oscilloscopes / 37

2 Setting Up

Step 1. Install Keysight IO Libraries Suite software / 42
Step 2. Connect and set up the oscilloscope / 43
Using the USB (Device) Interface / 43
Using the LAN Interface / 43
Using the GPIB Interface / 44
Step 3. Verify the oscilloscope connection / 45

3 Getting Started

Basic Oscilloscope Program Structure / 50
Initializing / 50
Capturing Data / 50
Analyzing Captured Data / 51

Programming the Oscilloscope / 52
Referencing the IO Library / 52
Opening the Oscilloscope Connection via the IO Library / 53
Initializing the Interface and the Oscilloscope / 53
Using :AUToscale to Automate Oscilloscope Setup / 54
Using Other Oscilloscope Setup Commands / 54
Capturing Data with the :DIGitize Command / 55
Reading Query Responses from the Oscilloscope / 57
Reading Query Results into String Variables / 58
Reading Query Results into Numeric Variables / 58
Reading Definite-Length Block Query Response Data / 58
Sending Multiple Queries and Reading Results / 59
Checking Instrument Status / 60
Other Ways of Sending Commands / 61
Telnet Sockets / 61
Sending SCPI Commands Using Browser Web Control / 61

4 Commands Quick Reference

Command Summary / 64
Syntax Elements / 116
Number Format / 116
<NL> (Line Terminator) / 116
[] (Optional Syntax Terms) / 116
{ } (Braces) / 116
::= (Defined As) / 116
< > (Angle Brackets) / 117
... (Ellipsis) / 117
n,...,p (Value Ranges) / 117
d (Digits) / 117
Quoted ASCII String / 117
Definite-Length Block Response Data / 117

5 Common (*) Commands

*CLS (Clear Status) / 123
*ESE (Standard Event Status Enable) / 124
*ESR (Standard Event Status Register) / 126
*IDN (Identification Number) / 128
*LRN (Learn Device Setup) / 129
*OPC (Operation Complete) / 130
*OPT (Option Identification) / 131

*RCL (Recall) / 133
*RST (Reset) / 134
*SAV (Save) / 137
*SRE (Service Request Enable) / 138
*STB (Read Status Byte) / 140
*TRG (Trigger) / 142
*TST (Self Test) / 143
*WAI (Wait To Continue) / 144

6 Root (:) Commands

:ACTivity / 149
:AER (Arm Event Register) / 150
:AUToscale / 151
:AUToscale:AMODE / 153
:AUToscale:CHANnels / 154
:AUToscale:FDEBug / 155
:BLANK / 156
:DIGItize / 157
:MTEnable (Mask Test Event Enable Register) / 159
:MTERegister[:EVENT] (Mask Test Event Event Register) / 161
:OPEE (Operation Status Enable Register) / 163
:OPERegister:CONDITION (Operation Status Condition Register) / 165
:OPERegister[:EVENT] (Operation Status Event Register) / 167
:OVLenable (Overload Event Enable Register) / 169
:OVLRegister (Overload Event Register) / 171
:PRINt / 173
:RUN / 174
:SERial / 175
:SINGle / 176
:STATus / 177
:STOP / 178
:TER (Trigger Event Register) / 179
:VIEW / 180

7 :ACQuire Commands

:ACQuire:COMplete / 183
:ACQuire:COUNt / 184
:ACQuire:MODE / 185
:ACQuire:POINts / 186
:ACQuire:SEGmented:ANALyze / 187
:ACQuire:SEGmented:COUNt / 188

:ACQuire:SEGMENTed:INDEX / 189
:ACQuire:SRATe / 192
:ACQuire:TYPE / 193

8 :BUS<n> Commands

:BUS<n>:BIT<m> / 197
:BUS<n>:BITS / 198
:BUS<n>:CLEAR / 200
:BUS<n>:DISPLAY / 201
:BUS<n>:LABEL / 202
:BUS<n>:MASK / 203

9 :CALibrate Commands

:CALibrate:DATE / 207
:CALibrate:LABEL / 208
:CALibrate:OUTPut / 209
:CALibrate:PROTected / 210
:CALibrate:STARt / 211
:CALibrate:STATus / 212
:CALibrate:TEMPerature / 213
:CALibrate:TIME / 214

10 :CHANnel<n> Commands

:CHANnel<n>:BANDwidth / 218
:CHANnel<n>:BWLimit / 219
:CHANnel<n>:COUpling / 220
:CHANnel<n>:DISPLAY / 221
:CHANnel<n>:IMPedance / 222
:CHANnel<n>:INVert / 223
:CHANnel<n>:LABEL / 224
:CHANnel<n>:OFFSet / 225
:CHANnel<n>:PROBe / 226
:CHANnel<n>:PROBe:HEAD[:TYPE] / 227
:CHANnel<n>:PROBe:ID / 228
:CHANnel<n>:PROBe:SKEW / 229
:CHANnel<n>:PROBe:STYPe / 230
:CHANnel<n>:PROTection / 231
:CHANnel<n>:RANGE / 232
:CHANnel<n>:SCALe / 233
:CHANnel<n>:UNITs / 234
:CHANnel<n>:VERNier / 235

11 :DEMO Commands

:DEMO:FUNCTION / 238
:DEMO:FUNCTION:PHASE:PHASE / 240
:DEMO:OUTPut / 241

12 :DIGItal<d> Commands

:DIGItal<d>:DISPlay / 245
:DIGItal<d>:LABel / 246
:DIGItal<d>:POSITION / 247
:DIGItal<d>:SIZE / 248
:DIGItal<d>:THReShold / 249

13 :DISPlay Commands

:DISPlay:ANNotation / 253
:DISPlay:ANNotation:BACKground / 254
:DISPlay:ANNotation:COLor / 255
:DISPlay:ANNotation:TEXT / 256
:DISPlay:CLEar / 257
:DISPlay:DATA / 258
:DISPlay:INTensity:WAVEform / 259
:DISPlay:LABel / 260
:DISPlay:LABList / 261
:DISPlay:PERSistence / 262
:DISPlay:VECTors / 263

14 :DVM Commands

:DVM:ARAnge / 266
:DVM:CURREnt / 267
:DVM:ENABLE / 268
:DVM:FREQuency / 269
:DVM:MODE / 270
:DVM:SOURce / 271

15 :EXTernal Trigger Commands

:EXTernal:BWLImit / 274
:EXTernal:PROBe / 275
:EXTernal:RANGE / 276
:EXTernal:UNITs / 277

16 :FUNCTION Commands

:FUNCTION:DISPlay / 282

:FUNCTION[:FFT]:CENTer / 283
:FUNCTION[:FFT]:SPAN / 284
:FUNCTION[:FFT]:VTYPe / 285
:FUNCTION[:FFT]:WINDOW / 286
:FUNCTION:GOFT:OPERation / 287
:FUNCTION:GOFT:SOURce1 / 288
:FUNCTION:GOFT:SOURce2 / 289
:FUNCTION:OFFSet / 290
:FUNCTION:OPERation / 291
:FUNCTION:RANGe / 292
:FUNCTION:REFerence / 293
:FUNCTION:SCALe / 294
:FUNCTION:SOURce1 / 295
:FUNCTION:SOURce2 / 296

17 :HARDcopy Commands

:HARDcopy:AREA / 299
:HARDcopy:APRinter / 300
:HARDcopy:FACTors / 301
:HARDcopy:FFEed / 302
:HARDcopy:INKSaver / 303
:HARDcopy:LAYout / 304
:HARDcopy:NETWork:ADDRess / 305
:HARDcopy:NETWork:APPLy / 306
:HARDcopy:NETWork:DOMain / 307
:HARDcopy:NETWork:PASSWORD / 308
:HARDcopy:NETWork:SLOT / 309
:HARDcopy:NETWork:USERname / 310
:HARDcopy:PALette / 311
:HARDcopy:PRINter:LIST / 312
:HARDcopy:STARt / 313

18 :LISTER Commands

:LISTER:DATA / 316
:LISTER:DISPLAY / 317
:LISTER:REFerence / 318

19 :MARKer Commands

:MARKer:MODE / 321
:MARKer:X1Position / 322
:MARKer:X1Y1source / 323

```
:MARKer:X2Position / 324
:MARKer:X2Y2source / 325
:MARKer:XDELta / 326
:MARKer:XUNits / 327
:MARKer:XUNits:USE / 328
:MARKer:Y1Position / 329
:MARKer:Y2Position / 330
:MARKer:YDELta / 331
:MARKer:YUNits / 332
:MARKer:YUNits:USE / 333
```

20 :MEASure Commands

```
:MEASure:ALL / 344
:MEASure:CLEar / 345
:MEASure:DEFine / 346
:MEASure:DELay / 349
:MEASure:DUTYcycle / 351
:MEASure:FALLtime / 352
:MEASure:FREQuency / 353
:MEASure:NWIDth / 354
:MEASure:OVERshoot / 355
:MEASure:PERiod / 357
:MEASure:PHASe / 358
:MEASure:PRESHoot / 359
:MEASure:PWIDth / 360
:MEASure:RISetime / 361
:MEASure:SHOW / 362
:MEASure:SOURce / 363
:MEASure:TEDGE / 365
:MEASure:TVALue / 367
:MEASure:VAMPLitude / 369
:MEASure:VAverage / 370
:MEASure:VBASe / 371
:MEASure:VMAX / 372
:MEASure:VMIN / 373
:MEASure:VPP / 374
:MEASure:VRMS / 375
:MEASure:VTIMe / 376
:MEASure:VTOP / 377
:MEASure:WINDOW / 378
```

21 :MTEST Commands

:MTEST:ALL / 384
:MTEST:AMASK:CREate / 385
:MTEST:AMASK:SOURce / 386
:MTEST:AMASK:UNITs / 387
:MTEST:AMASK:XDELta / 388
:MTEST:AMASK:YDELta / 389
:MTEST:COUNT:FWAVeforms / 390
:MTEST:COUNT:RESet / 391
:MTEST:COUNT:TIME / 392
:MTEST:COUNT:WAVeforms / 393
:MTEST:DATA / 394
:MTEST:DELetE / 395
:MTEST:ENABLE / 396
:MTEST:LOCK / 397
:MTEST:RMODE / 398
:MTEST:RMODE:FACTion:MEASure / 399
:MTEST:RMODE:FACTion:PRINT / 400
:MTEST:RMODE:FACTion:SAVE / 401
:MTEST:RMODE:FACTion:STOP / 402
:MTEST:RMODE:SIGMa / 403
:MTEST:RMODE:TIME / 404
:MTEST:RMODE:WAVeforms / 405
:MTEST:SCALe:BIND / 406
:MTEST:SCALe:X1 / 407
:MTEST:SCALe:XDELta / 408
:MTEST:SCALe:Y1 / 409
:MTEST:SCALe:Y2 / 410
:MTEST:SOURce / 411
:MTEST:TITLe / 412

22 :POD Commands

:POD<n>:DISPlay / 415
:POD<n>:SIZE / 416
:POD<n>:THReShold / 417

23 :RECall Commands

:RECall:FILEname / 421
:RECall:MASK[:STARt] / 422
:RECall:PWD / 423
:RECall:SETup[:STARt] / 424

:RECall:WMEMory<r>[:START] / 425

24 :SAVE Commands

:SAVE:FILEname / 430
:SAVE:IMAGe[:START] / 431
:SAVE:IMAGe:FACTors / 432
:SAVE:IMAGe:FORMAT / 433
:SAVE:IMAGe:INKSaver / 434
:SAVE:IMAGe:PAlette / 435
:SAVE:LISTER[:START] / 436
:SAVE:MASK[:START] / 437
:SAVE:MULTi[:START] / 438
:SAVE:PWD / 439
:SAVE:SETup[:START] / 440
:SAVE:WAVEform[:START] / 441
:SAVE:WAVEform:FORMAT / 442
:SAVE:WAVEform:LENGTH / 443
:SAVE:WAVEform:LENGTH:MAX / 444
:SAVE:WAVEform:SEGmented / 445
:SAVE:WMEMory:SOURce / 446
:SAVE:WMEMory[:START] / 447

25 :SBUS<n> Commands

General :SBUS<n> Commands / 451
:SBUS<n>:DISPLAY / 452
:SBUS<n>:MODE / 453
:SBUS<n>:CAN Commands / 454
:SBUS<n>:CAN:COUNT:ERRor / 456
:SBUS<n>:CAN:COUNT:OVERload / 457
:SBUS<n>:CAN:COUNT:RESET / 458
:SBUS<n>:CAN:COUNT:TOTal / 459
:SBUS<n>:CAN:COUNT:UTILization / 460
:SBUS<n>:CAN:SAMPLEpoint / 461
:SBUS<n>:CAN:SIGNAl:BAUDrate / 462
:SBUS<n>:CAN:SIGNAl:DEFinition / 463
:SBUS<n>:CAN:SOURce / 464
:SBUS<n>:CAN:TRIGger / 465
:SBUS<n>:CAN:TRIGger:PATTERn:DATA / 467
:SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGTH / 468
:SBUS<n>:CAN:TRIGger:PATTERn:ID / 469
:SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE / 470

```
:SBUS<n>:IIC Commands / 471
:SBUS<n>:IIC:ASIZe / 472
:SBUS<n>:IIC[:SOURce]:CLOCK / 473
:SBUS<n>:IIC[:SOURce]:DATA / 474
:SBUS<n>:IIC:TRIGger:PATTern:ADDReSS / 475
:SBUS<n>:IIC:TRIGger:PATTern:DATA / 476
:SBUS<n>:IIC:TRIGger:PATTern:DATA2 / 477
:SBUS<n>:IIC:TRIGger:QUALifier / 478
:SBUS<n>:IIC:TRIGger[:TYPE] / 479

:SBUS<n>:LIN Commands / 481
:SBUS<n>:LIN:PARity / 483
:SBUS<n>:LIN:SAMPlepoint / 484
:SBUS<n>:LIN:SIGNal:BAUDrate / 485
:SBUS<n>:LIN:SOURce / 486
:SBUS<n>:LIN:STANDARD / 487
:SBUS<n>:LIN:SYNChronous / 488
:SBUS<n>:LIN:TRIGger / 489
:SBUS<n>:LIN:TRIGger:ID / 490
:SBUS<n>:LIN:TRIGger:PATTern:DATA / 491
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth / 493
:SBUS<n>:LIN:TRIGger:PATTern:FORMAT / 494

:SBUS<n>:SPI Commands / 495
:SBUS<n>:SPI:BITorder / 497
:SBUS<n>:SPI:CLOCK:SLOPe / 498
:SBUS<n>:SPI:CLOCK:TIMEout / 499
:SBUS<n>:SPI:FRAMing / 500
:SBUS<n>:SPI:SOURce:CLOCK / 501
:SBUS<n>:SPI:SOURce:FRAMe / 502
:SBUS<n>:SPI:SOURce:MISO / 503
:SBUS<n>:SPI:SOURce:MOStI / 504
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA / 505
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh / 506
:SBUS<n>:SPI:TRIGger:PATTern:MOStI:DATA / 507
:SBUS<n>:SPI:TRIGger:PATTern:MOStI:WIDTh / 508
:SBUS<n>:SPI:TRIGger:TYPE / 509
:SBUS<n>:SPI:WIDTh / 510

:SBUS<n>:UART Commands / 511
:SBUS<n>:UART:BASE / 513
:SBUS<n>:UART:BAUDrate / 514
:SBUS<n>:UART:BITorder / 515
```

```
:SBUS<n>:UART:COUNT:ERRor / 516
:SBUS<n>:UART:COUNT:RESet / 517
:SBUS<n>:UART:COUNT:RXFRAMES / 518
:SBUS<n>:UART:COUNT:TXFRAMES / 519
:SBUS<n>:UART:FRAMing / 520
:SBUS<n>:UART:PARity / 521
:SBUS<n>:UART:POLarity / 522
:SBUS<n>:UART:SOURce:RX / 523
:SBUS<n>:UART:SOURce:TX / 524
:SBUS<n>:UART:TRIGger:BASE / 525
:SBUS<n>:UART:TRIGger:BURSt / 526
:SBUS<n>:UART:TRIGger:DATA / 527
:SBUS<n>:UART:TRIGger:IDLE / 528
:SBUS<n>:UART:TRIGger:QUALifier / 529
:SBUS<n>:UART:TRIGger:TYPE / 530
:SBUS<n>:UART:WIDTh / 531
```

26 :SEARch Commands

```
General :SEARch Commands / 534
:SEARch:COUNT / 535
:SEARch:MODE / 536
:SEARch:STATe / 537

:SEARch:SERial:CAN Commands / 538
:SEARch:SERial:CAN:MODE / 539
:SEARch:SERial:CAN:PATTERn:DATA / 540
:SEARch:SERial:CAN:PATTERn:DATA:LENGTH / 541
:SEARch:SERial:CAN:PATTERn:ID / 542
:SEARch:SERial:CAN:PATTERn:ID:MODE / 543

:SEARch:SERial:IIC Commands / 544
:SEARch:SERial:IIC:MODE / 545
:SEARch:SERial:IIC:PATTERn:ADDRess / 547
:SEARch:SERial:IIC:PATTERn:DATA / 548
:SEARch:SERial:IIC:PATTERn:DATA2 / 549
:SEARch:SERial:IIC:QUALifier / 550

:SEARch:SERial:LIN Commands / 551
:SEARch:SERial:LIN:ID / 552
:SEARch:SERial:LIN:MODE / 553
:SEARch:SERial:LIN:PATTERn:DATA / 554
:SEARch:SERial:LIN:PATTERn:DATA:LENGTH / 555
:SEARch:SERial:LIN:PATTERn:FORMAT / 556
```

:SEARch:SERial:SPI Commands / 557
:SEARch:SERial:SPI:MODE / 558
:SEARch:SERial:SPI:PATTERn:DATA / 559
:SEARch:SERial:SPI:PATTERn:WIDTh / 560

:SEARch:SERial:UART Commands / 561
:SEARch:SERial:UART:DATA / 562
:SEARch:SERial:UART:MODE / 563
:SEARch:SERial:UART:QUALifier / 564

27 :SYSTem Commands

:SYSTem:DATE / 567
:SYSTem:DSP / 568
:SYSTem:ERRor / 569
:SYSTem:LOCK / 570
:SYSTem:MENU / 571
:SYSTem:PRESet / 572
:SYSTem:PROTection:LOCK / 575
:SYSTem:SETup / 576
:SYSTem:TIME / 578

28 :TIMEbase Commands

:TIMEbase:MODE / 581
:TIMEbase:POSITION / 582
:TIMEbase:RANGE / 583
:TIMEbase:REFERENCE / 584
:TIMEbase:SCALe / 585
:TIMEbase:VERNier / 586
:TIMEbase:WINDOW:POSITION / 587
:TIMEbase:WINDOW:RANGE / 588
:TIMEbase:WINDOW:SCALe / 589

29 :TRIGger Commands

General :TRIGger Commands / 593
:TRIGger:FORCe / 594
:TRIGger:HFReject / 595
:TRIGger:HOLDoff / 596
:TRIGger:LEVel:ASETup / 597
:TRIGger:LEVel:HIGH / 598
:TRIGger:LEVel:LOW / 599
:TRIGger:MODE / 600
:TRIGger:NREject / 601

:TRIGger:SWEep / 602
:TRIGger[:EDGE] Commands / 603
:TRIGger[:EDGE]:COUpling / 604
:TRIGger[:EDGE]:LEVel / 605
:TRIGger[:EDGE]:REJect / 606
:TRIGger[:EDGE]:SLOPe / 607
:TRIGger[:EDGE]:SOURce / 608
:TRIGger:GLITch Commands / 609
:TRIGger:GLITch:GREaterthan / 611
:TRIGger:GLITch:LESSthan / 612
:TRIGger:GLITch:LEVel / 613
:TRIGger:GLITch:POLarity / 614
:TRIGger:GLITch:QUALifier / 615
:TRIGger:GLITch:RANGE / 616
:TRIGger:GLITch:SOURce / 617
:TRIGger:PATTern Commands / 618
:TRIGger:PATTern / 619
:TRIGger:PATTern:FORMat / 621
:TRIGger:PATTern:QUALifier / 622
:TRIGger:TV Commands / 623
:TRIGger:TV:LINE / 624
:TRIGger:TV:MODE / 625
:TRIGger:TV:POLarity / 626
:TRIGger:TV:SOURce / 627
:TRIGger:TV:STANDARD / 628

30 :WAVeform Commands

:WAVeform:BYTeorder / 637
:WAVeform:COUNt / 638
:WAVeform:DATA / 639
:WAVeform:FORMat / 641
:WAVeform:POINTs / 642
:WAVeform:POINTs:MODE / 644
:WAVeform:PREamble / 646
:WAVeform:SEGmented:COUNt / 649
:WAVeform:SEGmented:TTAG / 650
:WAVeform:SOURce / 651
:WAVeform:SOURce:SUBSource / 655
:WAVeform:TYPE / 656
:WAVeform:UNSIGNED / 657

```
:WAVEform:VIEW / 658  
:WAVEform:XINCrement / 659  
:WAVEform:XORigin / 660  
:WAVEform:XREFerence / 661  
:WAVEform:YINCrement / 662  
:WAVEform:YORigin / 663  
:WAVEform:YREFerence / 664
```

31 :WGEN Commands

```
:WGEN:FREQuency / 668  
:WGEN:FUNCTION / 669  
:WGEN:FUNCTION:PULSe:WIDTH / 671  
:WGEN:FUNCTION:RAMP:SYMMetry / 672  
:WGEN:FUNCTION:SQUare:DCYCle / 673  
:WGEN:MODulation:AM:DEPTH / 674  
:WGEN:MODulation:AM:FREQuency / 675  
:WGEN:MODulation:FM:DEViation / 676  
:WGEN:MODulation:FM:FREQuency / 677  
:WGEN:MODulation:FSKey:FREQuency / 678  
:WGEN:MODulation:FSKey:RATE / 679  
:WGEN:MODulation:FUNCTION / 680  
:WGEN:MODulation:FUNCTION:RAMP:SYMMetry / 681  
:WGEN:MODulation:NOISe / 682  
:WGEN:MODulation:STATe / 683  
:WGEN:MODulation:TYPE / 684  
:WGEN:OUTPut / 686  
:WGEN:OUTPut:LOAD / 687  
:WGEN:PERiod / 688  
:WGEN:RST / 689  
:WGEN:VOLTage / 690  
:WGEN:VOLTage:HIGH / 691  
:WGEN:VOLTage:LOW / 692  
:WGEN:VOLTage:OFFSet / 693
```

32 :WMEMory<r> Commands

```
:WMEMory<r>:CLEar / 697  
:WMEMory<r>:DISPlay / 698  
:WMEMory<r>:LABel / 699  
:WMEMory<r>:SAVE / 700  
:WMEMory<r>:SKEW / 701  
:WMEMory<r>:YOFFset / 702
```

:WMEMory<r>:YRANge / 703
:WMEMory<r>:YScale / 704

33 Obsolete and Discontinued Commands

:CHANnel:ACTivity / 710
:CHANnel:LABel / 711
:CHANnel:THRehold / 712
:CHANnel2:SKEW / 713
:CHANnel<n>:INPut / 714
:CHANnel<n>:PMODe / 715
:DISPlay:CONNect / 716
:DISPlay:ORDer / 717
:ERASe / 718
:EXTernal:PMODe / 719
:FUNCtion:SOURce / 720
:FUNCtion:VIEW / 721
:HARDcopy:DESTination / 722
:HARDcopy:FILEname / 723
:HARDcopy:GRAYscale / 724
:HARDcopy:IGColors / 725
:HARDcopy:PDRiver / 726
:MEASure:LOWER / 727
:MEASure:SCRatch / 728
:MEASure:TDELta / 729
:MEASure:THReholds / 730
:MEASure:TSTArt / 731
:MEASure:TSTOp / 732
:MEASure:TVOLT / 733
:MEASure:UPPer / 734
:MEASure:VDELta / 735
:MEASure:VSTArt / 736
:MEASure:VSTOp / 737
:MTEST:AMASK:{SAVE | STORe} / 738
:MTEST:AVERage / 739
:MTEST:AVERage:COUNT / 740
:MTEST:LOAD / 741
:MTEST:RUMode / 742
:MTEST:RUMode:SOFailure / 743
:MTEST:{STARt | STOP} / 744
:MTEST:TRIGger:SOURce / 745
:PRINT? / 746

:SAVE:IMAGe:AREA / 748
:TIMEbase:DElay / 749
:TRIGger:THRehold / 750
:TRIGger:TV:TVMode / 751

34 Error Messages

35 Status Reporting

Status Reporting Data Structures / 763
Status Byte Register (STB) / 766
Service Request Enable Register (SRE) / 768
Trigger Event Register (TER) / 769
Output Queue / 770
Message Queue / 771
(Standard) Event Status Register (ESR) / 772
(Standard) Event Status Enable Register (ESE) / 773
Error Queue / 774
Operation Status Event Register (:OPERegister[:EVENT]) / 775
Operation Status Condition Register (:OPERegister:CONDition) / 776
Arm Event Register (AER) / 777
Overload Event Register (:OVLRegister) / 778
Mask Test Event Event Register (:MTERegister[:EVENT]) / 779
Clearing Registers and Queues / 780
Status Reporting Decision Chart / 781

36 Synchronizing Acquisitions

Synchronization in the Programming Flow / 784
Set Up the Oscilloscope / 784
Acquire a Waveform / 784
Retrieve Results / 784
Blocking Synchronization / 785
Polling Synchronization With Timeout / 786
Synchronizing with a Single-Shot Device Under Test (DUT) / 788
Synchronization with an Averaging Acquisition / 790

37 More About Oscilloscope Commands

- Command Classifications / 794
 - Core Commands / 794
 - Non-Core Commands / 794
 - Obsolete Commands / 794
- Valid Command/Query Strings / 795
 - Program Message Syntax / 795
 - Duplicate Mnemonics / 799
 - Tree Traversal Rules and Multiple Commands / 799
- Query Return Values / 801
- All Oscilloscope Commands Are Sequential / 802

38 Programming Examples

- VISA COM Examples / 804
 - VISA COM Example in Visual Basic / 804
 - VISA COM Example in C# / 813
 - VISA COM Example in Visual Basic .NET / 822
 - VISA COM Example in Python / 830
- VISA Examples / 837
 - VISA Example in C / 837
 - VISA Example in Visual Basic / 846
 - VISA Example in C# / 856
 - VISA Example in Visual Basic .NET / 867
 - VISA Example in Python / 877
- SICL Examples / 884
 - SICL Example in C / 884
 - SICL Example in Visual Basic / 893
- SCPI.NET Examples / 904
 - SCPI.NET Example in C# / 904
 - SCPI.NET Example in Visual Basic .NET / 910
 - SCPI.NET Example in IronPython / 916

Index

1 What's New

- What's New in Version 2.39 / 26
- What's New in Version 2.38 / 27
- What's New in Version 2.30 / 28
- What's New in Version 2.20 / 30
- What's New in Version 2.10 / 32
- What's New in Version 2.00 / 33
- What's New in Version 1.20 / 34
- What's New in Version 1.10 / 35
- Version 1.00 at Introduction / 36
- Command Differences From 7000B Series Oscilloscopes / 37

What's New in Version 2.39

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

New Commands

Command	Differences
:DISPLAY:INTensity:WAVeform (see page 259)	Sets the waveform intensity.

What's New in Version 2.38

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

New Commands

Command	Differences
:CHANnel<n>:BANDwidth (see page 218)	Sets bandwidth limiting for an analog input channel.

What's New in Version 2.30

New features in version 2.30 of the InfiniiVision 2000 X-Series oscilloscope software are:

- Support for CAN/LIN, I2C/SPI, and UART/RS232 serial triggering and decode.
- Saving Multi Channel Waveform data (*.h5) format files that can be opened by the N8900A InfiniiView oscilloscope analysis software.

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

New Commands

Command	Differences
:LISTer Commands (see page 315)	Commands for turning on/off the serial decode Lister display and returning data from the Lister display.
:SAVE:LISTer[:STARt] (see page 436)	Saves the Lister display data to a file.
:SAVE:MULTi[:STARt] (see page 438)	Lets you save Multi Channel Waveform data (*.h5) format files that can be opened by the N8900A InfiniiView oscilloscope analysis software.
:SBUS<n>:CAN Commands (see page 454)	Commands for CAN triggering and serial decode.
:SBUS<n>:DISPlay (see page 452)	Turns serial decode ON or OFF.
:SBUS<n>:IIC Commands (see page 471)	Commands for IIC triggering and serial decode.
:SBUS<n>:LIN Commands (see page 481)	Commands for LIN triggering and serial decode.
:SBUS<n>:MODE (see page 453)	Selects the type of serial decode.
:SBUS<n>:SPI Commands (see page 454)	Commands for SPI triggering and serial decode.
:SBUS<n>:UART Commands (see page 511)	Commands for UART/RS232 triggering and serial decode.
:SEARch:COUNT (see page 535)	Returns the number of search events found.
:SEARch:MODE (see page 536)	Can select SERial1.
:SEARch:SERial:CAN Commands (see page 538)	Commands for finding CAN events in the captured data.
:SEARch:SERial:IIC Commands (see page 544)	Commands for finding IIC events in the captured data.

Command	Differences
:SEARch:SERial:LIN Commands (see page 551)	Commands for finding LIN events in the captured data.
:SEARch:SERial:SPI Commands (see page 557)	Commands for finding SPI events in the captured data.
:SEARch:SERial:UART Commands (see page 561)	Commands for finding UART/RS232 events in the captured data.
:SEARch:STATe (see page 537)	Enables or disables the search feature.
:TRIGger:LEVel:ASETup (see page 597)	Sets the trigger levels of all displayed analog channels to their waveforms' 50% values.

Changed Commands

Command	Differences
:BLANK (see page 156)	You can now use the SBUS1 source parameter to turn off the serial decode display.
:DIGItize (see page 157)	You can now use the SBUS1 source parameter to acquire the serial waveform.
:SAVE:WAVeform:FORMat (see page 442)	The ALB format is no longer supported.
:STATus (see page 177)	You can now use the SBUS1 source parameter to view the serial decode display status.
:SYSTem:MENU (see page 571)	The LISTer parameter is now available.
:VIEW (see page 180)	You can now use the SBUS1 source parameter to turn on the serial decode display.
:WAVeform:SOURce (see page 651)	Can select SBUS1 as the waveform source.
:WAVeform:SOURce:SUBSourc e (see page 655)	With the SPI and UART/RS232 serial decode options, subsources are now valid in the 2000 X-Series oscilloscopes.

What's New in Version 2.20

New features in version 2.20 of the InfiniiVision 2000 X-Series oscilloscope software are:

- Support for modulation of the waveform generator output.
- Support for controlling the optional DSOXDVM digital voltmeter analysis feature
- Ability to turn reference waveform locations on or off and view their status using the :VIEW, :BLANK, and :STATus commands.

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

New Commands

Command	Description
:DVM Commands (see page 265)	Commands for controlling the optional DSOXDVM digital voltmeter analysis feature.
:WGEN:MODulation:AM:DEPTH (see page 674)	Specifies the amount of amplitude modulation.
:WGEN:MODulation:AM:FREQuency (see page 675)	Specifies the frequency of the modulating signal.
:WGEN:MODulation:FM:DEViation (see page 676)	Specifies the frequency deviation from the original carrier signal frequency.
:WGEN:MODulation:FM:FREQuency (see page 677)	Specifies the frequency of the modulating signal.
:WGEN:MODulation:FSKey:FREQuency (see page 678)	Specifies the "hop frequency".
:WGEN:MODulation:FSKey:RATE (see page 679)	Specifies the rate at which the output frequency "shifts".
:WGEN:MODulation:FUNCTION (see page 680)	Specifies the shape of the modulating signal.
:WGEN:MODulation:FUNCTION:RAMP:SYMMetry (see page 681)	Specifies the amount of time per cycle that the ramp waveform is rising.
:WGEN:MODulation:STATE (see page 683)	Enables or disables modulated waveform generator output.
:WGEN:MODulation:TYPE (see page 684)	Selects the modulation type: Amplitude Modulation (AM), Frequency Modulation (FM), or Frequency-Shift Keying Modulation (FSK).

Changed Commands

Command	Differences
:BLANK (see page 156)	You can now use the WMEMory<r> source parameter to turn off the display of a reference waveform location.
:STATus (see page 177)	You can now use the WMEMory<r> source parameter to view the display status of a reference waveform location.
:VIEW (see page 180)	You can now use the WMEMory<r> source parameter to turn on the display of a reference waveform location.

What's New in Version 2.10

New features in version 2.10 of the InfiniiVision 2000 X-Series oscilloscope software are:

- Support for adding an annotation to the display.

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

New Commands

Command	Description
:DISPlay:ANNotation (see page 253)	Turns screen annotation on or off.
:DISPlay:ANNotation:BACKground (see page 254)	Specifies the background of the annotation to be either opaque, inverted, or transparent.
:DISPlay:ANNotation:COLor (see page 255)	Specifies the color of the annotation.
:DISPlay:ANNotation:TEXT (see page 256)	Specifies the annotation string, up to 254 characters.

What's New in Version 2.00

New features in version 2.00 of the InfiniiVision 2000 X-Series oscilloscope software are:

- Ability to add noise to the waveform generator's output signal.

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

New Commands

Command	Description
:WGEN:MODulation:NOISe (see page 682)	Adds noise to the waveform generator's output signal.

What's New in Version 1.20

New features in version 1.20 of the InfiniiVision 2000 X-Series oscilloscope software are:

- X cursor units that let you measure time (seconds), frequency (Hertz), phase (degrees), and ratio (percent), and Y cursor units that let you measure the channel units (base) or ratio (percent).
- Option for specifying FFT vertical units as V RMS as well as decibels.
- Option for saving the maximum number of waveform data points.

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

New Commands

Command	Description
:FUNCtion[FFT]:VTYPe (see page 285)	Specifies FFT vertical units as DECibel or VRMS.
:MARKer:XUNIts (see page 327)	Specifies the units for X cursors.
:MARKer:XUNIts:USE (see page 328)	Sets the current X1 and X2 cursor locations as 0 and 360 degrees if XUNIts is DEGRees or as 0 and 100 percent if XUNIts is PERCent.
:MARKer:YUNIts (see page 332)	Specifies the units for Y cursors.
:MARKer:YUNIts:USE (see page 333)	Sets the current Y1 and Y2 cursor locations as 0 and 100 percent if YUNIts is PERCent.
:SAVE:WAVeform:LENGth:MAX (see page 444)	Enable or disables saving the maximum number of waveform data points.
:TRIGger:FORCe (see page 594)	Now documented, this command is equivalent to the front panel [Force Trigger] key which causes an acquisition to be captured even though the trigger condition has not been met.

What's New in Version 1.10

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

- Support for the new extended Video triggering license.

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

New Commands

Command	Description
:SYSTem:PRESet (see page 572)	Now documented, this command is equivalent to the front panel [Default Setup] key which leaves some user settings, like preferences, unchanged. The *RST command is equivalent to a factory default setup where no user settings are left unchanged.

Version 1.00 at Introduction

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

The command set is most closely related to the InfiniiVision 7000B Series oscilloscopes (and the 7000A Series, 6000 Series, and 54620/54640 Series oscilloscopes before them). For more information, see "["Command Differences From 7000B Series Oscilloscopes"](#) on page 37.

Command Differences From 7000B Series Oscilloscopes

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

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

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

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

New Commands

Command	Description
:DEMO Commands (see page 237)	Commands for using built-in demo signals (with the Option EDU license that comes with the N6455A Education Kit).
:HARDcopy:NETWork Commands (see page 297)	For accessing network printers.
:MEASure:WINDOW (see page 378)	When the zoomed time base is on, specifies whether the measurement window is the zoomed time base or the main time base.
:MTEST:ALL (see page 384)	Specifies whether all channels are included in the mask test.
:RECall:WMEMory<r>[:STARt] (see page 425)	Recalls reference waveforms.
:SAVE:WMEMory:SOURce (see page 446)	Selects the source for saving a reference waveform.
:SAVE:WMEMory[:STARt] (see page 447)	Saves reference waveforms.
:TRIGger:LEVel:HIGH (see page 598)	Sets runt and transition (rise/fall time) trigger high level.
:TRIGger:LEVel:LOW (see page 599)	Sets runt and transition (rise/fall time) trigger low level.

Command	Description
:TRIGger:PATTERn Commands (see page 618)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:DURation subsystem.
:WGEN Commands (see page 665)	Commands for controlling the built-in waveform generator (with Option WGN license).
:WMEMory<r> Commands (see page 695)	Commands for reference waveforms.

Changed Commands

Command	Differences From InfiniiVision 7000B Series Oscilloscopes
:ACQuire:MODE (see page 185)	There is no ETIMe parameter with the 2000 X-Series oscilloscopes.
:CALibrate:OUTPut (see page 209)	The TRIG OUT signal can be a trigger output, mask test failure, or waveform generator sync pulse.
:DISPlay:DATA (see page 258)	Monochrome TIFF images of the graticule cannot be saved or restored.
:DISPlay:LABList (see page 261)	The label list contains up to 77, 10-character labels (instead of 75).
:DISPlay:VECTors (see page 263)	Always ON with 2000 X-Series oscilloscopes.
:MARKer Commands (see page 319)	Can select reference waveforms as marker source.
:MEASure Commands (see page 335)	Can select reference waveforms as the source for many measurements.
:SAVE:IMAGe[:STARt] (see page 431)	Cannot save images to internal locations.
:TRIGger:PATTERn (see page 619)	Takes <string> parameter instead of <value>,<mask> parameters.
:WAVeform:SOURce (see page 651)	Can select reference waveforms as the source.
:VIEW (see page 180)	PMEMory (pixel memory) locations are not present.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences

Discontinued Commands

Command	Description
:ACQuire:RSIGnal	The 2000 X-Series oscilloscope does not have a 10 MHz REF BNC connector.
:CALibrate:SWITch?	Replaced by :CALibrate:PROTected? (see page 210). The oscilloscope has a protection button instead of a switch.
:DISPlay:SOURce	PMEMory (pixel memory) locations are not present.
:EXTernal:IMPedance	External TRIG IN connector is now fixed at 1 MΩ.
:EXTernal:PROBe:ID	Not supported on external TRIG IN connector.
:EXTernal:PROBe:STYPe	Not supported on external TRIG IN connector.
:EXTernal:PROTection	Not supported on external TRIG IN connector.
:HARDcopy:DEvice, :HARDcopy:FORMAT	Use the :SAVE:IMAGe:FORMAT, :SAVE:WAVeform:FORMAT, and :HARDcopy:APRinter commands instead.
:MERGe	Waveform traces have been replaced by reference waveforms.
:RECall:IMAGe[:STARt]	Waveform traces have been replaced by reference waveforms.
:SYSTem:PRECision	The 2000 X-Series oscilloscopes' measurement record, and maximum record size, is 62,500 points, and there is no need for a special precision mode.
:TIMEbase:REFClock	The 2000 X-Series oscilloscope does not have a 10 MHz REF BNC connector.

2 Setting Up

Step 1. Install Keysight IO Libraries Suite software / 42

Step 2. Connect and set up the oscilloscope / 43

Step 3. Verify the oscilloscope connection / 45

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

Step 1. Install Keysight IO Libraries Suite software

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

Step 2. Connect and set up the oscilloscope

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

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

When installed, these interfaces are always active.

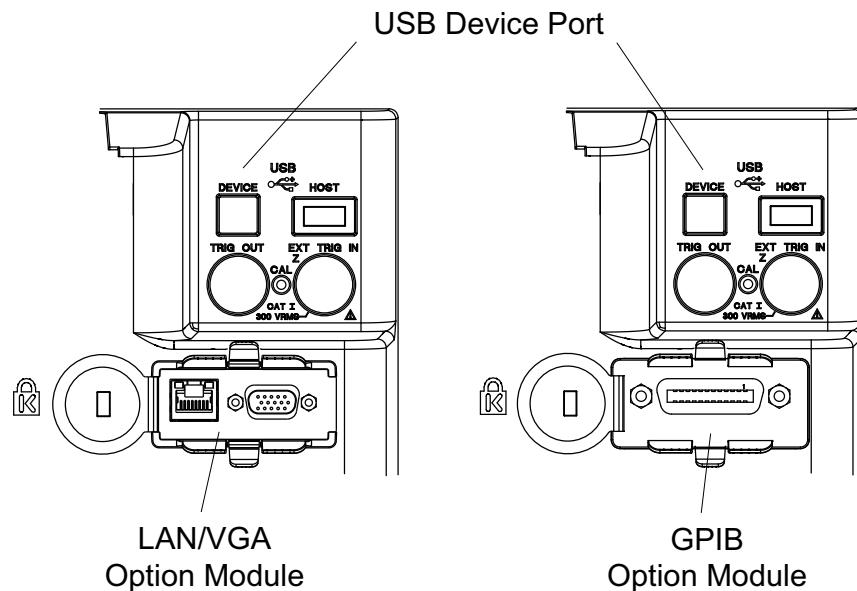


Figure 1 Control Connectors on Rear Panel

Using the USB (Device) Interface

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

This is a USB 2.0 high-speed port.

Using the LAN Interface

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

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

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

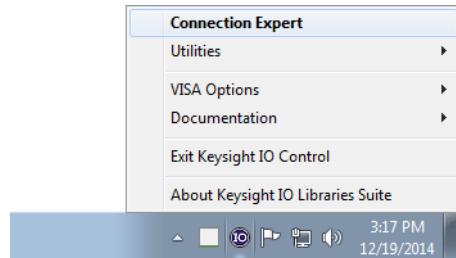
- 3 Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port on the LAN/VGA option module.
- 4 Configure the oscilloscope's LAN interface:
 - a Press the **Configure** softkey until "LAN" is selected.
 - b Press the **LAN Settings** softkey.
 - c Press the **Config** softkey, and enable all the configuration options supported by your network.
 - d If automatic configuration is not supported, press the **Addresses** softkey.
Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the IP Address, Subnet Mask, Gateway IP, and DNS IP values.
When you are done, press the **[Back up]** key.
 - e Press the **Host name** softkey. Use the softkeys and the Entry knob to enter the Host name.
When you are done, press the **[Back up]** key.

Using the GPIB Interface

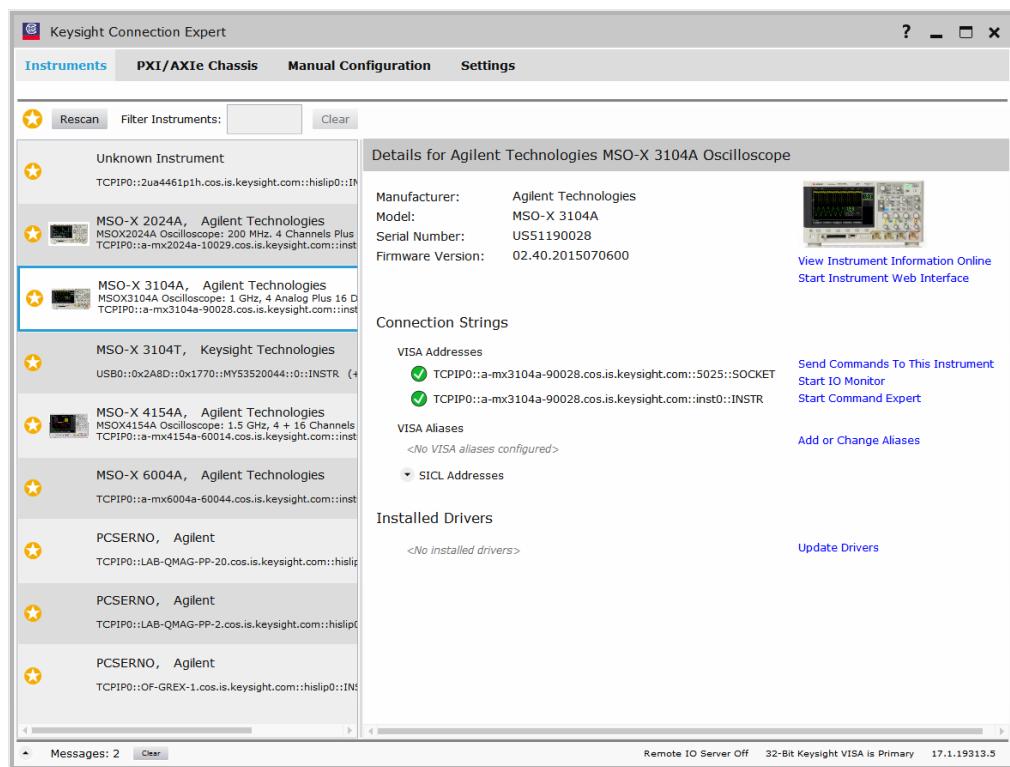
- 1 Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the GPIB option module.
- 2 Configure the oscilloscope's GPIB interface:
 - a Press the **Configure** softkey until "GPIB" is selected.
 - b Use the Entry knob to select the **Address** value.

Step 3. Verify the oscilloscope connection

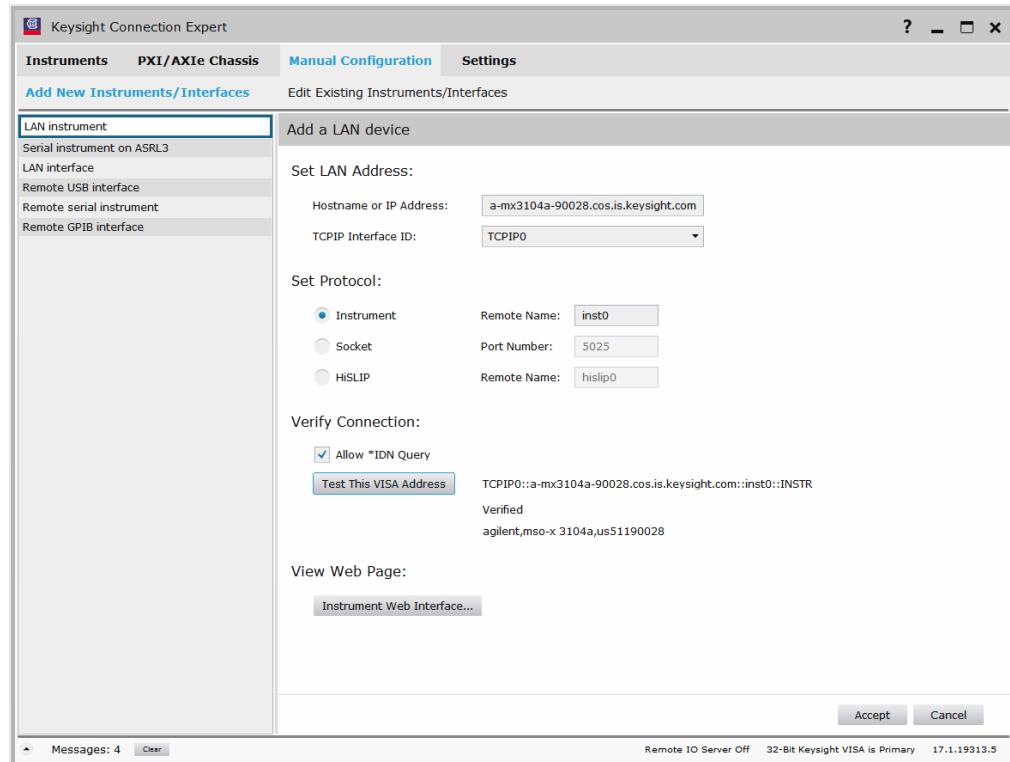
- 1 On the controller PC, click on the Keysight IO Control icon in the taskbar and choose **Connection Expert** from the popup menu.



- 2 In the Keysight Connection Expert application, instruments connected to the controller's USB and GPIB interfaces as well as instruments on the same LAN subnet should automatically appear in the Instruments tab.



- 3 If your instrument does not appear, you can add it using the Manual Configuration tab.



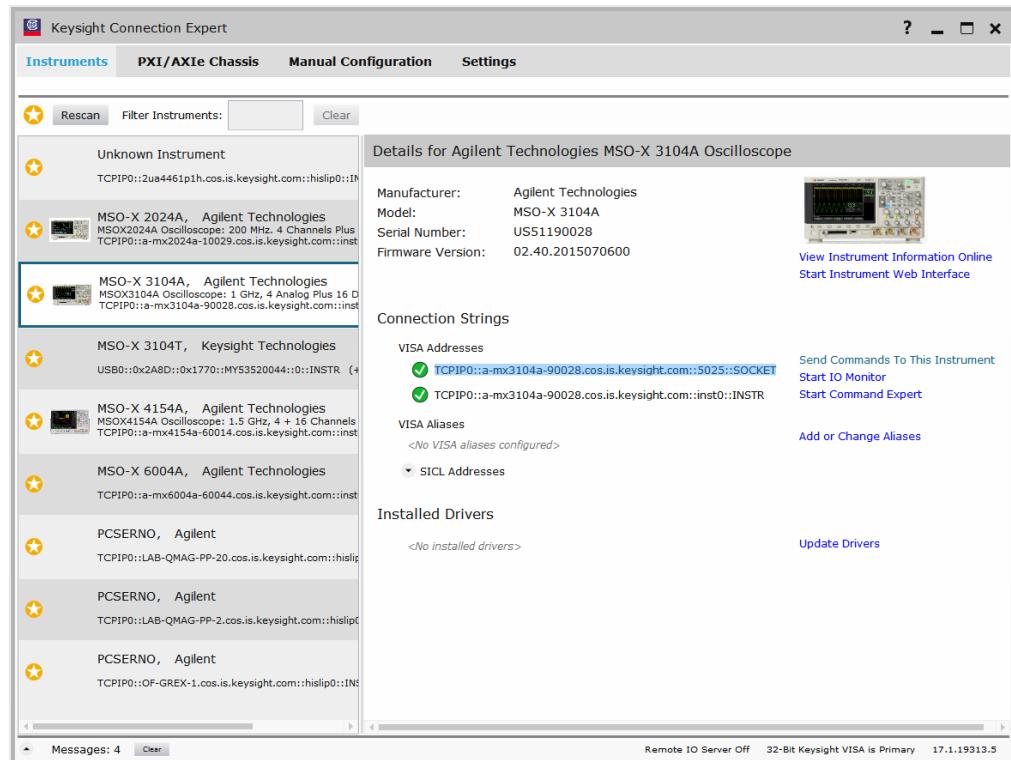
For example, to add a device:

- a Select **LAN instrument** in the list on the left.
- b Enter the oscilloscope's **Hostname or IP address**.
- c Select the protocol.
- d Select **Instrument** under Set Protocol.
- e Click **Test This VISA Address** to verify the connection.
- f If the connection test is successful, click **Accept** to add the instrument.

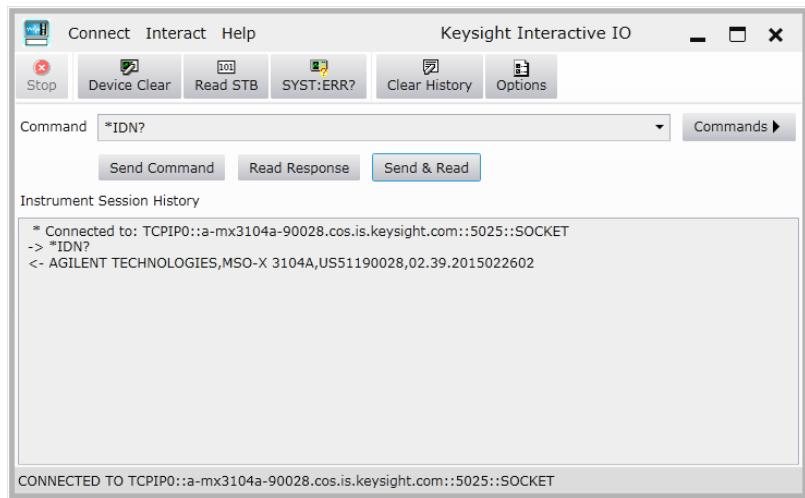
If the connection test is not successful, go back and verify the LAN connections and the oscilloscope setup.

4 Test some commands on the instrument:

- a** In the Details for the selected instrument, click **Send Commands To This Instrument**.



- b** In the Keysight 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 Keysight Interactive IO application.
- 5 In the Keysight Connection Expert application, choose **File > Exit** from the menu to exit the application.

3 Getting Started

Basic Oscilloscope Program Structure / 50

Programming the Oscilloscope / 52

Other Ways of Sending Commands / 61

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

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

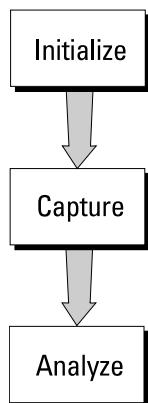
NOTE

Language for Program Examples

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

Basic Oscilloscope Program Structure

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



Initializing

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

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

Capturing Data

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

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

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

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

Analyzing Captured Data

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

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

Programming the Oscilloscope

- "Referencing the IO Library" on page 52
- "Opening the Oscilloscope Connection via the IO Library" on page 53
- "Using :AUToscale to Automate Oscilloscope Setup" on page 54
- "Using Other Oscilloscope Setup Commands" on page 54
- "Capturing Data with the :DIGitize Command" on page 55
- "Reading Query Responses from the Oscilloscope" on page 57
- "Reading Query Results into String Variables" on page 58
- "Reading Query Results into Numeric Variables" on page 58
- "Reading Definite-Length Block Query Response Data" on page 58
- "Sending Multiple Queries and Reading Results" on page 59
- "Checking Instrument Status" on page 60

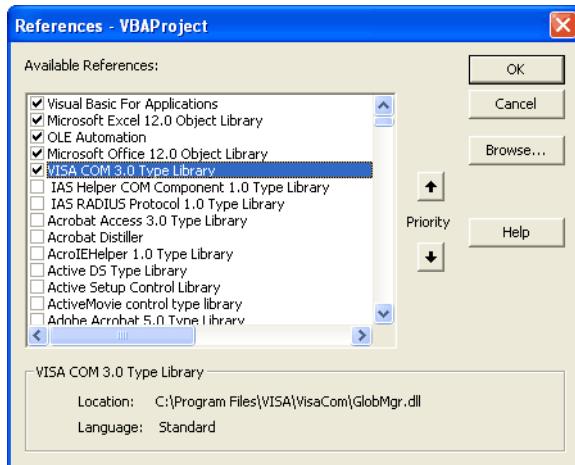
Referencing the IO Library

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 Keysight IO Libraries Suite documentation for more information).

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

- 1** Choose **Project>References...** from the main menu.
- 2** In the References dialog, check the "VISA COM 3.0 Type Library".
- 3** Click **OK**.

Opening the Oscilloscope Connection via the IO Library

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

For example, when using the Keysight 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
' Keysight Connection Expert (installed with Keysight IO Libraries Suite
').
Set myScope.IO = myMgr.Open("<VISA Address>")

' Send a command.
myScope.WriteString ":DISPLAY:LABEL ON"
```

The ":DISPLAY:LABEL ON" in the above example is called a *program message*. Program messages are explained in more detail in "[Program Message Syntax](#)" on page 795.

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 Keysight VISA COM library, you can use the resource session object's Clear method to clear the interface buffer:

```

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

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

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

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

```

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

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

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

NOTE

Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in [Chapter 5](#), “Common (*) Commands,” starting on page 119.

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

Using :AUToscale to Automate Oscilloscope Setup

The :AUToscale command performs a very useful function for unknown waveforms by setting up the vertical channel, time base, and trigger level of the instrument.

The syntax for the autoscale command is:

```
myScope.WriteString ":AUToscale"
```

Using Other Oscilloscope Setup Commands

A typical oscilloscope setup would set the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope. An example of the commands that might be sent to the oscilloscope are:

```

myScope.WriteString ":CHANnel1:PROBe 10"
myScope.WriteString ":CHANnel1:RANGE 16"
myScope.WriteString ":CHANnel1:OFFSet 1.00"
myScope.WriteString ":TIMEbase:MODE MAIN"
myScope.WriteString ":TIMEbase:RANGE 1E-3"
myScope.WriteString ":TIMEbase:DELay 100E-6"

```

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

Example Oscilloscope Setup Code

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

```
' Initialize the instrument interface to a known state.
myScope.IO.Clear
myScope.IO.Timeout = 10000      ' Set interface timeout to 10 seconds.

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

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

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

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

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

Capturing Data with the :DIGITIZE Command

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

NOTE**Ensure New Data is Collected**

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

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

NOTE**Set :TIMEbase:MODE to MAIN when using :DIGITIZE**

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

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

The following program example shows a typical setup:

```
myScope.WriteString ":ACQUIRE:TYPE AVERAGE"
myScope.WriteString ":ACQUIRE:COMPLETE 100"
myScope.WriteString ":ACQUIRE:COUNT 8"
myScope.WriteString ":DIGITIZE CHANnel1"
myScope.WriteString ":WAVEFORM:SOURCE CHANnel1"
myScope.WriteString ":WAVEFORM:FORMAT BYTE"
myScope.WriteString ":WAVEFORM:POINTS 500"
myScope.WriteString ":WAVEFORM:DATA?"
```

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

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

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

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

For more information, see the waveform subsystem commands and corresponding program code examples in [Chapter 30](#), “:WAVeform Commands,” starting on page 629.

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

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

Reading Query Responses from the Oscilloscope

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

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

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

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

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

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

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

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

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

Reading Query Results into String Variables

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

NOTE

Express String Variables Using Exact Syntax

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

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

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

After running this program, the controller displays:

Range (string): +40.0E+00

Reading Query Results into Numeric Variables

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

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

After running this program, the controller displays:

Range (variant): 40

Reading Definite-Length Block Query Response Data

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

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

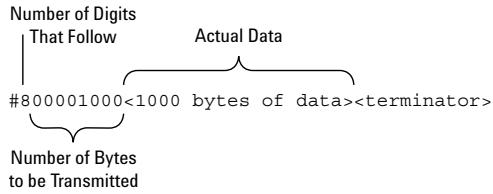


Figure 2 Definite-length block response data

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

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

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

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

Sending Multiple Queries and Reading Results

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

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

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

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

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

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

```
myScope.WriteString ":TIMEbase:RANGE?;DElay?"
Dim strResults() As String
```

```
strResults() = myScope.ReadList(ASCIIIType_BSTR)
MsgBox "Timebase range: " + strResults(0) + ", delay: " + strResults(1)
```

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

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

Checking Instrument Status

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

For more information, see [Chapter 35](#), “Status Reporting,” starting on page 761 which explains how to check the status of the instrument.

Other Ways of Sending Commands

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

- "[Telnet Sockets](#)" on page 61
- "[Sending SCPI Commands Using Browser Web Control](#)" on page 61

Telnet Sockets

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

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

```
telnet <hostname> 5024
```

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

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

```
telnet <hostname> 5025
```

Sending SCPI Commands Using Browser Web Control

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

4 Commands Quick Reference

Command Summary / 64

Syntax Elements / 116

Command Summary

- Common (*) Commands Summary (see [page 65](#))
- Root (:) Commands Summary (see [page 67](#))
- :ACQuire Commands Summary (see [page 70](#))
- :BUS<n> Commands Summary (see [page 71](#))
- :CALibrate Commands Summary (see [page 72](#))
- :CHANnel<n> Commands Summary (see [page 73](#))
- :DEMO Commands Summary (see [page 75](#))
- :DIGItal<n> Commands Summary (see [page 75](#))
- :DISPlay Commands Summary (see [page 76](#))
- :DVM Commands Summary (see [page 77](#))
- :EXTernal Trigger Commands Summary (see [page 77](#))
- :FUNCTION Commands Summary (see [page 78](#))
- :HARDcopy Commands Summary (see [page 79](#))
- :LISTer Commands Summary (see [page 81](#))
- :MARKer Commands Summary (see [page 81](#))
- :MEASure Commands Summary (see [page 82](#))
- :MTEST Commands Summary (see [page 89](#))
- :POD<n> Commands Summary (see [page 91](#))
- :RECall Commands Summary (see [page 92](#))
- :SAVE Commands Summary (see [page 93](#))
- General :SBUS<n> Commands Summary (see [page 94](#))
- :SBUS<n>:CAN Commands Summary (see [page 95](#))
- :SBUS<n>:IIC Commands Summary (see [page 96](#))
- :SBUS<n>:LIN Commands Summary (see [page 97](#))
- :SBUS<n>:SPI Commands Summary (see [page 98](#))
- :SBUS<n>:UART Commands Summary (see [page 99](#))
- General :SEARch Commands Summary (see [page 101](#))
- :SEARch:SERial:CAN Commands Summary (see [page 101](#))
- :SEARch:SERial:IIC Commands Summary (see [page 102](#))
- :SEARch:SERial:LIN Commands Summary (see [page 102](#))
- :SEARch:SERial:SPI Commands Summary (see [page 103](#))
- :SEARch:SERial:UART Commands Summary (see [page 104](#))
- :SYSTem Commands Summary (see [page 104](#))

- :TIMEbase Commands Summary (see [page 105](#))
- General :TRIGger Commands Summary (see [page 106](#))
- :TRIGger[:EDGE] Commands Summary (see [page 107](#))
- :TRIGger:GLITch Commands Summary (see [page 108](#))
- :TRIGger:PATTern Commands Summary (see [page 109](#))
- :TRIGger:TV Commands Summary (see [page 110](#))
- :WAVEform Commands Summary (see [page 110](#))
- :WGEN Commands Summary (see [page 113](#))
- :WMEMory<r> Commands Summary (see [page 114](#))

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 123)	n/a	n/a
*ESE <mask> (see page 124)	*ESE? (see page 125)	<mask> ::= 0 to 255; an integer in NR1 format: Bit Weight Name Enables ---- ----- ---- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete
n/a	*ESR? (see page 126)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 126)	AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 129)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 130)	*OPC? (see page 130)	ASCII "1" is placed in the output queue when all pending device operations have completed.

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns																																				
n/a	*OPT? (see page 131)	<pre> <return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <MSO>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Power Measurements>, <reserved>, <Segmented Memory>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Waveform Generator>, <reserved>, <reserved> <All field> ::= {0 All} <reserved> ::= 0 <MSO> ::= {0 MSO} <Power Measurements> ::= {0 PWR} <Segmented Memory> ::= {0 SGM} <Mask Test> ::= {0 MASK} <Bandwidth> ::= {0 BW10 BW20} <Waveform Generator> ::= {0 WAVEGEN} </pre>																																				
*RCL <value> (see page 133)	n/a	<pre> <value> ::= {0 1 4 5 6 7 8 9} </pre>																																				
*RST (see page 134)	n/a	See *RST (Reset) (see page 134)																																				
*SAV <value> (see page 137)	n/a	<pre> <value> ::= {0 1 4 5 6 7 8 9} </pre>																																				
*SRE <mask> (see page 138)	*SRE? (see page 139)	<p><mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>Enables</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>Operation Status Reg</td> </tr> <tr> <td>6</td> <td>64</td> <td>----</td> <td>(Not used.)</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td>Event Status Bit</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td>Message Available</td> </tr> <tr> <td>3</td> <td>8</td> <td>----</td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td>Message</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td>User</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td>Trigger</td> </tr> </tbody> </table>	Bit	Weight	Name	Enables	7	128	OPER	Operation Status Reg	6	64	----	(Not used.)	5	32	ESB	Event Status Bit	4	16	MAV	Message Available	3	8	----	(Not used.)	2	4	MSG	Message	1	2	USR	User	0	1	TRG	Trigger
Bit	Weight	Name	Enables																																			
7	128	OPER	Operation Status Reg																																			
6	64	----	(Not used.)																																			
5	32	ESB	Event Status Bit																																			
4	16	MAV	Message Available																																			
3	8	----	(Not used.)																																			
2	4	MSG	Message																																			
1	2	USR	User																																			
0	1	TRG	Trigger																																			

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns																																													
n/a	*STB? (see page 140)	<p><value> ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>"1"</th> <th>Indicates</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td></td> <td>Operation status condition occurred.</td> </tr> <tr> <td>6</td> <td>64</td> <td>RQS/</td> <td></td> <td>Instrument is MSS requesting service.</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td></td> <td>Enabled event status condition occurred.</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td></td> <td>Message available.</td> </tr> <tr> <td>3</td> <td>8</td> <td>----</td> <td>(Not used.)</td> <td></td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td></td> <td>Message displayed.</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td></td> <td>User event condition occurred.</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td></td> <td>A trigger occurred.</td> </tr> </tbody> </table>	Bit	Weight	Name	"1"	Indicates	7	128	OPER		Operation status condition occurred.	6	64	RQS/		Instrument is MSS requesting service.	5	32	ESB		Enabled event status condition occurred.	4	16	MAV		Message available.	3	8	----	(Not used.)		2	4	MSG		Message displayed.	1	2	USR		User event condition occurred.	0	1	TRG		A trigger occurred.
Bit	Weight	Name	"1"	Indicates																																											
7	128	OPER		Operation status condition occurred.																																											
6	64	RQS/		Instrument is MSS requesting service.																																											
5	32	ESB		Enabled event status condition occurred.																																											
4	16	MAV		Message available.																																											
3	8	----	(Not used.)																																												
2	4	MSG		Message displayed.																																											
1	2	USR		User event condition occurred.																																											
0	1	TRG		A trigger occurred.																																											
*TRG (see page 142)	n/a	n/a																																													
n/a	*TST? (see page 143)	<result> ::= 0 or non-zero value; an integer in NR1 format																																													
*WAI (see page 144)	n/a	n/a																																													

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 149)	:ACTivity? (see page 149)	<p><return value> ::= <edges>,<levels></p> <p><edges> ::= presence of edges (32-bit integer in NR1 format)</p> <p><levels> ::= logical highs or lows (32-bit integer in NR1 format)</p>
n/a	:AER? (see page 150)	{0 1}; an integer in NR1 format

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:AUToscale [<source/> [,...<source>]] (see page 151)	n/a	<p><source> ::= CHANnel<n> for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> POD1 POD2} for MSO models</p> <p><source> can be repeated up to 5 times</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p>
:AUToscale:AMODE <value> (see page 153)	:AUToscale:AMODE?	<value> ::= {NORMAl CURREnt}}
:AUToscale:CHANnels <value> (see page 154)	:AUToscale:CHANnels?	<value> ::= {ALL DISPlayed}}
:AUToscale:FDEBug {{0 OFF} {1 ON}} (see page 155)	:AUToscale:FDEBug?	{0 1}
:BLANK [<source>] (see page 156)	n/a	<p><source> ::= {CHANnel<n>} FUNCtion MATH SBUS1 WMEMOry<r>} for DSO models</p> <p><source> ::= {CHANnel<n>} DIGItal<d> POD{1 2} BUS{1 2} FUNCtion MATH SBUS1 WMEMOry<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p>
:DIGItize [<source/> [,...<source>]] (see page 157)	n/a	<p><source> ::= {CHANnel<n>} FUNCtion MATH} for DSO models</p> <p><source> ::= {CHANnel<n>} DIGItal<d> POD{1 2} BUS{1 2} FUNCtion MATH} for MSO models</p> <p><source> can be repeated up to 5 times</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p>

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns																																	
:MTEenable <n> (see page 159)	:MTEenable? (see page 159)	<n> ::= 16-bit integer in NR1 format																																	
n/a	:MTERegister[:EVENT] ? (see page 161)	<n> ::= 16-bit integer in NR1 format																																	
:OPEE <n> (see page 163)	:OPEE? (see page 164)	<n> ::= 15-bit integer in NR1 format																																	
n/a	:OPERregister:CONDitiOn? (see page 165)	<n> ::= 15-bit integer in NR1 format																																	
n/a	:OPERregister[:EVENT] ? (see page 167)	<n> ::= 15-bit integer in NR1 format																																	
:OVLenable <mask> (see page 169)	:OVLenable? (see page 170)	<p><mask> ::= 16-bit integer in NR1 format as shown:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Input</th> </tr> </thead> <tbody> <tr><td>10</td><td>1024</td><td>Ext Trigger Fault</td></tr> <tr><td>9</td><td>512</td><td>Channel 4 Fault</td></tr> <tr><td>8</td><td>256</td><td>Channel 3 Fault</td></tr> <tr><td>7</td><td>128</td><td>Channel 2 Fault</td></tr> <tr><td>6</td><td>64</td><td>Channel 1 Fault</td></tr> <tr><td>4</td><td>16</td><td>Ext Trigger OVL</td></tr> <tr><td>3</td><td>8</td><td>Channel 4 OVL</td></tr> <tr><td>2</td><td>4</td><td>Channel 3 OVL</td></tr> <tr><td>1</td><td>2</td><td>Channel 2 OVL</td></tr> <tr><td>0</td><td>1</td><td>Channel 1 OVL</td></tr> </tbody> </table>	Bit	Weight	Input	10	1024	Ext Trigger Fault	9	512	Channel 4 Fault	8	256	Channel 3 Fault	7	128	Channel 2 Fault	6	64	Channel 1 Fault	4	16	Ext Trigger OVL	3	8	Channel 4 OVL	2	4	Channel 3 OVL	1	2	Channel 2 OVL	0	1	Channel 1 OVL
Bit	Weight	Input																																	
10	1024	Ext Trigger Fault																																	
9	512	Channel 4 Fault																																	
8	256	Channel 3 Fault																																	
7	128	Channel 2 Fault																																	
6	64	Channel 1 Fault																																	
4	16	Ext Trigger OVL																																	
3	8	Channel 4 OVL																																	
2	4	Channel 3 OVL																																	
1	2	Channel 2 OVL																																	
0	1	Channel 1 OVL																																	
n/a	:OVLRegister? (see page 171)	<value> ::= integer in NR1 format. See OVLenable for <value>																																	
:PRINT [<options>] (see page 173)	n/a	<options> ::= [<print option>] [, ..., <print option>] <print option> ::= {COLor GRAYscale PRINTER0 PRINTER1 BMP8bit BMP PNG NOFactors FACTors} <print option> can be repeated up to 5 times.																																	
:RUN (see page 174)	n/a	n/a																																	
n/a	:SERial (see page 175)	<return value> ::= unquoted string containing serial number																																	
:SINGle (see page 176)	n/a	n/a																																	

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:STATUs? <display> (see page 177)	{0 1} <display> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCtion MATH SBUS1 WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 178)	n/a	n/a
n/a	:TER? (see page 179)	{0 1}
:VIEW <source> (see page 180)	n/a	<source> ::= {CHANnel<n> FUNCtion MATH SBUS1 WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCtion MATH SBUS1 WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:COMplete <complete> (see page 183)	:ACQuire:COMplete? (see page 183)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNT <count> (see page 184)	:ACQuire:COUNT? (see page 184)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:MODE <mode> (see page 185)	:ACQuire:MODE? (see page 185)	<mode> ::= {RTIMe SEGmented}
n/a	:ACQuire:POINTS? (see page 186)	<# points> ::= an integer in NR1 format
:ACQuire:SEGmented:AN ALyze (see page 187)	n/a	n/a (with Option SGM)

Table 4 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
:ACQuire:SEGmented:CO UNT <count> (see page 188)	:ACQuire:SEGmented:CO UNT? (see page 188)	<count> ::= an integer from 2 to 25 in NR1 format (with Option SGM)
:ACQuire:SEGmented:IN Dex <index> (see page 189)	:ACQuire:SEGmented:IN Dex? (see page 189)	<index> ::= an integer from 1 to 25 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATE? (see page 192)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 193)	:ACQuire:TYPE? (see page 193)	<type> ::= {NORMal AVERage HRESolution PEAK}

Table 5 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 OFF} {1 ON}} (see page 197)	:BUS<n>:BIT<m>? (see page 197)	{0 1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 OFF} {1 ON}} (see page 198)	:BUS<n>:BITS? (see page 198)	<channel_list>, {0 1} <channel_list> ::= (@<m>,<m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:CLEar (see page 200)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 201)	:BUS<n>:DISPlay? (see page 201)	{0 1} <n> ::= 1 or 2; an integer in NR1 format

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

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see page 202)	:BUS<n>:LABel? (see page 202)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 203)	:BUS<n>:MASK? (see page 203)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

Table 6 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 207)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 208)	:CALibrate:LABel? (see page 208)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 209)	:CALibrate:OUTPut? (see page 209)	<signal> ::= {TRIGgers MASK WAVEgen}
n/a	:CALibrate:PROTected? (see page 210)	{"PROTected" "UNPROtected"}
:CALibrate:START (see page 211)	n/a	n/a
n/a	:CALibrate:STATUS? (see page 212)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string

Table 6 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPERATURE? (see page 213)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 214)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Table 7 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BANDwidth <limit> (see page 218)	:CHANnel<n>:BANDwidth? [MAXimum] (see page 218)	<limit> ::= 25E6 in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:BWLimit {{0 OFF} {1 ON}} (see page 219)	:CHANnel<n>:BWLimit? (see page 219)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUpling <coupling> (see page 220)	:CHANnel<n>:COUpling? (see page 220)	<coupling> ::= {AC DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 OFF} {1 ON}} (see page 221)	:CHANnel<n>:DISPlay? (see page 221)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 222)	:CHANnel<n>:IMPedance? (see page 222)	<impedance> ::= ONEmeg <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 OFF} {1 ON}} (see page 223)	:CHANnel<n>:INVert? (see page 223)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABEL <string> (see page 224)	:CHANnel<n>:LABEL? (see page 224)	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 225)	:CHANnel<n>:OFFSet? (see page 225)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 226)	:CHANnel<n>:PROBe? (see page 226)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format

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

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:HEA D[:TYPE] <head_param> (see page 227)	:CHANnel<n>:PROBe:HEA D[:TYPE]? (see page 227)	<head_param> ::= {SEND0 SEND6 SEND12 SEND20 DIFF0 DIFF6 DIFF12 DIFF20 NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 228)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKE W <skew_value> (see page 229)	:CHANnel<n>:PROBe:SKE W? (see page 229)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STY Pe <signal type> (see page 230)	:CHANnel<n>:PROBe:STY Pe? (see page 230)	<signal type> ::= {DIFFerential SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTectio n (see page 231)	:CHANnel<n>:PROTectio n? (see page 231)	NORM <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGE <range>[suffix] (see page 232)	:CHANnel<n>:RANGE? (see page 232)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 233)	:CHANnel<n>:SCALe? (see page 233)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITS <units> (see page 234)	:CHANnel<n>:UNITS? (see page 234)	<units> ::= {VOLT AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier { {0 OFF} {1 ON} } (see page 235)	:CHANnel<n>:VERNier? (see page 235)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format

Table 8 :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCTION <signal> (see page 238)	:DEMO:FUNCTION? (see page 239)	<signal> ::= {SINusoid NOISy PHASe RINGing SINGle AM CLK GLITch BURSt MSO RFBurst LFSine FMBurst}
:DEMO:FUNCTION:PHASE: PHASE <angle> (see page 240)	:DEMO:FUNCTION:PHASE: PHASE? (see page 240)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0 OFF} {1 ON}} (see page 241)	:DEMO:OUTPut? (see page 241)	{0 1}

Table 9 :DIGItal<d> Commands Summary

Command	Query	Options and Query Returns
:DIGItal<d>:DISPlay { {0 OFF} {1 ON}} (see page 245)	:DIGItal<d>:DISPlay? (see page 245)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 1}
:DIGItal<d>:LABel <string> (see page 246)	:DIGItal<d>:LABel? (see page 246)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGItal<d>:POSIon <position> (see page 247)	:DIGItal<d>:POSIon? (see page 247)	<d> ::= 0 to (# digital channels - 1) in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small Returns -1 when there is no space to display the digital waveform.
:DIGItal<d>:SIZE <value> (see page 248)	:DIGItal<d>:SIZE? (see page 248)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALL MEDIUM LARGE}
:DIGItal<d>:THRehold <value>[suffix] (see page 249)	:DIGItal<d>:THRehold? (see page 249)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {CMOS ECL TTL <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V mV uV}

Table 10 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation {{0 OFF} {1 ON}} (see page 253)	:DISPlay:ANNotation? (see page 253)	{0 1}
:DISPlay:ANNotation:B ACKground <mode> (see page 254)	:DISPlay:ANNotation:B ACKground? (see page 254)	<mode> ::= {OPAQue INVerted TRANsparent}
:DISPlay:ANNotation:C OLoR <color> (see page 255)	:DISPlay:ANNotation:C OLoR? (see page 255)	<color> ::= {CH1 CH2 CH3 CH4 DIG MATH REF MARKer WHITe RED}
:DISPlay:ANNotation:T EXT <string> (see page 256)	:DISPlay:ANNotation:T EXT? (see page 256)	<string> ::= quoted ASCII string (up to 254 characters)
:DISPlay:CLEar (see page 257)	n/a	n/a
n/a	:DISPlay:DATA? [<format>] [,] [<palett e>] (see page 258)	<format> ::= {BMP BMP8bit PNG} <palette> ::= {COLOR GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:INTensity:WA Veform <value> (see page 259)	:DISPlay:INTensity:WA Veform? (see page 259)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:LABel {{0 OFF} {1 ON}} (see page 260)	:DISPlay:LABel? (see page 260)	{0 1}
:DISPlay:LABList <binary block> (see page 261)	:DISPlay:LABList? (see page 261)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:PERSistence <value> (see page 262)	:DISPlay:PERSistence? (see page 262)	<value> ::= {MINimum INFinite <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {1 ON} (see page 263)	:DISPlay:VECTors? (see page 263)	1

Table 11 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARAnge {{0 OFF} {1 ON}} (see page 266)	:DVM:ARAnge? (see page 266)	{0 1}
n/a	:DVM:CURREnt? (see page 267)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABLE {{0 OFF} {1 ON}} (see page 268)	:DVM:ENABLE? (see page 268)	{0 1}
n/a	:DVM:FREQuency? (see page 267)	<freq_value> ::= floating-point number in NR3 format
:DVM:MODE <mode> (see page 270)	:DVM:MODE? (see page 270)	<dvm_mode> ::= {ACRMs DC DCRMs FREQuency}
:DVM:SOURce <source> (see page 271)	:DVM:SOURce? (see page 271)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format

Table 12 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLImit <bwlimit> (see page 274)	:EXTernal:BWLImit? (see page 274)	<bwlimit> ::= {0 OFF}
:EXTernal:PROBe <attenuation> (see page 275)	:EXTernal:PROBe? (see page 275)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGE <range>[<suffix>] (see page 276)	:EXTernal:RANGE? (see page 276)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V mV}
:EXTernal:UNITS <units> (see page 277)	:EXTernal:UNITS? (see page 277)	<units> ::= {VOLT AMPere}

Table 13 :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:DISPlay {{0 OFF} {1 ON}} (see page 282)	:FUNCTION:DISPLAY? (see page 282)	{0 1}
:FUNCTION[:FFT]:CENTEr <frequency> (see page 283)	:FUNCTION[:FFT]:CENTEr? (see page 283)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCTION[:FFT]:SPAN (see page 284)	:FUNCTION[:FFT]:SPAN? (see page 284)	 ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCTION[:FFT]:VTYPE <units> (see page 285)	:FUNCTION[:FFT]:VTYPE? (see page 285)	<units> ::= {DECibel VRMS}
:FUNCTION[:FFT]:WINDOW <>window> (see page 286)	:FUNCTION[:FFT]:WINDOW? (see page 286)	<window> ::= {RECTangular HANNing FLATtop BHARRis}
:FUNCTION:GOFT:OPERation <operation> (see page 287)	:FUNCTION:GOFT:OPERation? (see page 287)	<operation> ::= {ADD SUBTract MULTIply}
:FUNCTION:GOFT:SOURce 1 <source> (see page 288)	:FUNCTION:GOFT:SOURce 1? (see page 288)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see page 289)	:FUNCTION:GOFT:SOURce 2? (see page 289)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:FUNCTION:OFFSet <offset> (see page 290)	:FUNCTION:OFFSet? (see page 290)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:OPERation <operation> (see page 291)	:FUNCTION:OPERation? (see page 291)	<operation> ::= {ADD SUBTract MULTIply FFT}

Table 13 :FUNCTION Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION:RANGE <range> (see page 292)	:FUNCTION:RANGE? (see page 292)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the FFT function is 8 to 800 dBV.
:FUNCTION:REFERENCE <level> (see page 293)	:FUNCTION:REFERENCE? (see page 293)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:SCALE <scale value>[<suffix>] (see page 294)	:FUNCTION:SCALE? (see page 294)	<scale value> ::= integer in NR1 format <suffix> ::= {V dB}
:FUNCTION:SOURcel <source> (see page 295)	:FUNCTION:SOURcel? (see page 295)	<source> ::= {CHANnel<n> GOFT} <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models GOFT is only for FFT operation.
:FUNCTION:SOURce2 <source> (see page 296)	:FUNCTION:SOURce2? (see page 296)	<source> ::= {CHANnel<n> NONE} <n> ::= {{1 2} {3 4}} for 4ch models, depending on SOURcel selection <n> ::= {1 2} for 2ch models

Table 14 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 299)	:HARDcopy:AREA? (see page 299)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 300)	:HARDcopy:APRinter? (see page 300)	<active_printer> ::= {<index> <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 OFF} {1 ON}} (see page 301)	:HARDcopy:FACTors? (see page 301)	{0 1}

Table 14 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:FFeed {{0 OFF} {1 ON}} (see page 302)	:HARDcopy:FFeed? (see page 302)	{0 1}
:HARDcopy:INKSaver {{0 OFF} {1 ON}} (see page 303)	:HARDcopy:INKSaver? (see page 303)	{0 1}
:HARDcopy:LAYOUT <layout> (see page 304)	:HARDcopy:LAYOUT? (see page 304)	<layout> ::= {LANDscape PORTRait}
:HARDcopy:NETWork:ADD Ress <address> (see page 305)	:HARDcopy:NETWork:ADD Ress? (see page 305)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APP Ly (see page 306)	n/a	n/a
:HARDcopy:NETWork:DOM ain <domain> (see page 307)	:HARDcopy:NETWork:DOM ain? (see page 307)	<domain> ::= quoted ASCII string
:HARDcopy:NETWork:PAS Sword <password> (see page 308)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLOT <slot> (see page 309)	:HARDcopy:NETWork:SLOT? (see page 309)	<slot> ::= {NET0 NET1}
:HARDcopy:NETWork:USE Rname <username> (see page 310)	:HARDcopy:NETWork:USE Rname? (see page 310)	<username> ::= quoted ASCII string
:HARDcopy:PAlette <palette> (see page 311)	:HARDcopy:PAlette? (see page 311)	<palette> ::= {COLor GRAYscale NONE}
n/a	:HARDcopy:PRINTER:LIST? (see page 312)	<list> ::= [<printer_spec>] ... <printer_spec> ::= " <index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y N} <name> ::= name of printer
:HARDcopy:START (see page 313)	n/a	n/a

Table 15 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 316)	<binary_block> ::= comma-separated data with newlines at the end of each row
:LISTer:DISPlay {{OFF 0} {SBUS1 ON 1} ALL} (see page 317)	:LISTer:DISPlay? (see page 317)	{OFF SBUS1 ALL}
:LISTer:REference <time_ref> (see page 318)	:LISTer:REference? (see page 318)	<time_ref> ::= {TRIGger PREVIOUS}

Table 16 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 321)	:MARKer:MODE? (see page 321)	<mode> ::= {OFF MEASurement MANual WAVEform}
:MARKer:X1Position <position>[suffix] (see page 322)	:MARKer:X1Position? (see page 322)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 323)	:MARKer:X1Y1source? (see page 323)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 324)	:MARKer:X2Position? (see page 324)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 325)	:MARKer:X2Y2source? (see page 325)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>

Table 16 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MARKer:XDELta? (see page 326)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 327)	:MARKer:XUNits? (see page 327)	<units> ::= {SEConds HERTZ DEGRees PERCent}
:MARKer:XUNits:USE (see page 328)	n/a	n/a
:MARKer:Y1Position <position>[suffix] (see page 329)	:MARKer:Y1Position? (see page 329)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position>[suffix] (see page 330)	:MARKer:Y2Position? (see page 330)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 331)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 332)	:MARKer:YUNits? (see page 332)	<units> ::= {BASE PERCent}
:MARKer:YUNits:USE (see page 333)	n/a	n/a

Table 17 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 344)	n/a	n/a
:MEASure:CLEar (see page 345)	n/a	n/a
:MEASure:DEFine DELay, <delay spec>[,<source>] (see page 346)	:MEASure:DEFine? DELay[,<source>] (see page 347)	<delay spec> ::= <edge_spec1>, <edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ -} <occurrence> ::= integer <source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>}

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DEFine THresholds, <threshold spec>[,<source>] (see page 346)	:MEASure:DEFine? THresholds[,<source>] (see page 347)	<threshold spec> ::= {STANDARD} {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCENT ABSOLUTE} <source> ::= {CHANNEL<n> FUNCTION MATH WMEMORY<r>}
:MEASure:DELay [<source1>] [,<source2>] (see page 349)	:MEASure:DELay? [<source1>] [,<source2>] (see page 349)	<source1,2> ::= {CHANNEL<n> FUNCTION MATH WMEMORY<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 351)	:MEASure:DUTYcycle? [<source>] (see page 351)	<source> ::= {CHANNEL<n> FUNCTION MATH WMEMORY<r>} for DSO models <source> ::= {CHANNEL<n> DIGITAL<d> FUNCTION MATH WMEMORY<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 352)	:MEASure:FALLtime? [<source>] (see page 352)	<source> ::= {CHANNEL<n> FUNCTION MATH WMEMORY<r>} for DSO models <source> ::= {CHANNEL<n> DIGITAL<d> FUNCTION MATH WMEMORY<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FREQuency [<source>] (see page 353)	:MEASure:FREQuency? [<source>] (see page 353)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= frequency in Hertz in NR3 format</p>
:MEASure:NWIDth [<source>] (see page 354)	:MEASure:NWIDth? [<source>] (see page 354)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= negative pulse width in seconds-NR3 format</p>
:MEASure:OVERshoot [<source>] (see page 355)	:MEASure:OVERshoot? [<source>] (see page 355)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><return_value> ::= the percent of the overshoot of the selected waveform in NR3 format</p>

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PERiod [<source>] (see page 357)	:MEASure:PERiod? [<source>] (see page 357)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= waveform period in seconds in NR3 format</p>
:MEASure:PHASE [<source1>] [,<source2>] (see page 358)	:MEASure:PHASE? [<source1>] [,<source2>] (see page 358)	<p><source1,2> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><return_value> ::= the phase angle value in degrees in NR3 format</p>
:MEASure:PREShoot [<source>] (see page 359)	:MEASure:PREShoot? [<source>] (see page 359)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><return_value> ::= the percent of preshoot of the selected waveform in NR3 format</p>
:MEASure:PWIDth [<source>] (see page 360)	:MEASure:PWIDth? [<source>] (see page 360)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= width of positive pulse in seconds in NR3 format</p>

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:RISetime [<source>] (see page 361)	:MEASure:RISetime? [<source>] (see page 361)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SHOW {1 ON} (see page 362)	:MEASure:SHOW? (see page 362)	{1}
:MEASure:SOURce <source1> [,<source2>] (see page 363)	:MEASure:SOURce? (see page 363)	<source1,2> ::= {CHANnel<n> FUNCTion MATH WMEMory<r> EXTERNAL} for DSO models <source1,2> ::= {CHANnel<n> DIGItal<d> FUNCTion MATH WMEMory<r> EXTERNAL} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> NONE}
n/a	:MEASure:TEDGE? <slope><occurrence>[,<source>] (see page 365)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> FUNCTion MATH WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TValue? <value>, [<slope>]<occurrence> [,<source>] (see page 367)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion MATH WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMplitude [<source>] (see page 369)	:MEASure:VAMplitude? [<source>] (see page 369)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<interval>] [,][<sour ce>] (see page 370)	:MEASure:VAverage? [<interval>] [,][<sour ce>] (see page 370)	<interval> ::= {CYCLE DISPLAY} <source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 371)	:MEASure:VBASe? [<source>] (see page 371)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 372)	:MEASure:VMAX? [<source>] (see page 372)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 373)	:MEASure:VMIN? [<source>] (see page 373)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 374)	:MEASure:VPP? [<source>] (see page 374)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<interval>][,] [<type>][,] [<source>] (see page 375)	:MEASure:VRMS? [<interval>][,] [<type>][,] [<source>] (see page 375)	<interval> ::= {CYCLE DISPLAY} <type> ::= {AC DC} <source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:VTIMe? <vtimE>[,<source>] (see page 376)	<vtimE> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> FUNCTion MATH WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format
:MEASure:VTOP [<source>] (see page 377)	:MEASure:VTOP? [<source>] (see page 377)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDOW <type> (see page 378)	:MEASure:WINDOW? (see page 378)	<type> ::= {MAIN ZOOM AUTO}

Table 18 :MTEST Commands Summary

Command	Query	Options and Query Returns
:MTEST:ALL { {0 OFF} {1 ON} } (see page 384)	:MTEST:ALL? (see page 384)	{0 1}
:MTEST:AMASK:CREAtE (see page 385)	n/a	n/a
:MTEST:AMASK:SOURce <source> (see page 386)	:MTEST:AMASK:SOURce? (see page 386)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:MTEST:AMASK:UNITS <units> (see page 387)	:MTEST:AMASK:UNITS? (see page 387)	<units> ::= {CURRent DIVisions}
:MTEST:AMASK:XDELta <value> (see page 388)	:MTEST:AMASK:XDELta? (see page 388)	<value> ::= X delta value in NR3 format

Table 18 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:AMASK:YDELta <value> (see page 389)	:MTEST:AMASK:YDELta? (see page 389)	<value> ::= Y delta value in NR3 format
n/a	:MTEST:COUNT:FWAVEforms? [CHANnel<n>] (see page 390)	<failed> ::= number of failed waveforms in NR1 format
:MTEST:COUNT:RESet (see page 391)	n/a	n/a
n/a	:MTEST:COUNT:TIME? (see page 392)	<time> ::= elapsed seconds in NR3 format
n/a	:MTEST:COUNT:WAVeform s? (see page 393)	<count> ::= number of waveforms in NR1 format
:MTEST:DATA <mask> (see page 394)	:MTEST:DATA? (see page 394)	<mask> ::= data in IEEE 488.2 # format.
:MTEST:DELETE (see page 395)	n/a	n/a
:MTEST:ENABLE {{0 OFF} {1 ON}} (see page 396)	:MTEST:ENABLE? (see page 396)	{0 1}
:MTEST:LOCK {{0 OFF} {1 ON}} (see page 397)	:MTEST:LOCK? (see page 397)	{0 1}
:MTEST:RMODE <rmode> (see page 398)	:MTEST:RMODE? (see page 398)	<rmode> ::= {FORever TIME SIGMa WAveforms}
:MTEST:RMODE:FACTion: MEASure {{0 OFF} {1 ON}} (see page 399)	:MTEST:RMODE:FACTion: MEASure? (see page 399)	{0 1}
:MTEST:RMODE:FACTion: PRINT {{0 OFF} {1 ON}} (see page 400)	:MTEST:RMODE:FACTion: PRINT? (see page 400)	{0 1}
:MTEST:RMODE:FACTion: SAVE {{0 OFF} {1 ON}} (see page 401)	:MTEST:RMODE:FACTion: SAVE? (see page 401)	{0 1}
:MTEST:RMODE:FACTion: STOP {{0 OFF} {1 ON}} (see page 402)	:MTEST:RMODE:FACTion: STOP? (see page 402)	{0 1}
:MTEST:RMODE:SIGMa <level> (see page 403)	:MTEST:RMODE:SIGMa? (see page 403)	<level> ::= from 0.1 to 9.3 in NR3 format

Table 18 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:RMODE:TIME <seconds> (see page 404)	:MTEST:RMODE:TIME? (see page 404)	<seconds> ::= from 1 to 86400 in NR3 format
:MTEST:RMODE:WAVeform s <count> (see page 405)	:MTEST:RMODE:WAVeform s? (see page 405)	<count> ::= number of waveforms in NR1 format
:MTEST:SCALe:BIND {{0 OFF} {1 ON}} (see page 406)	:MTEST:SCALe:BIND? (see page 406)	{0 1}
:MTEST:SCALe:X1 <x1_value> (see page 407)	:MTEST:SCALe:X1? (see page 407)	<x1_value> ::= X1 value in NR3 format
:MTEST:SCALe:XDELta <xdelta_value> (see page 408)	:MTEST:SCALe:XDELta? (see page 408)	<xdelta_value> ::= X delta value in NR3 format
:MTEST:SCALe:Y1 <y1_value> (see page 409)	:MTEST:SCALe:Y1? (see page 409)	<y1_value> ::= Y1 value in NR3 format
:MTEST:SCALe:Y2 <y2_value> (see page 410)	:MTEST:SCALe:Y2? (see page 410)	<y2_value> ::= Y2 value in NR3 format
:MTEST:SOURce <source> (see page 411)	:MTEST:SOURce? (see page 411)	<source> ::= {CHANnel<n> NONE} <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
n/a	:MTEST:TITLe? (see page 412)	<title> ::= a string of up to 128 ASCII characters

Table 19 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 OFF} {1 ON}} (see page 415)	:POD<n>:DISPlay? (see page 415)	{0 1} <n> ::= 1 in NR1 format

Table 19 :POD<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:POD<n>:SIZE <value> (see page 416)	:POD<n>:SIZE? (see page 416)	<value> ::= {SMALL MEDIUM LARGe}
:POD<n>:THreshold <type>[suffix] (see page 417)	:POD<n>:THreshold? (see page 417)	<n> ::= 1 in NR1 format <type> ::= {CMOS ECL TTL <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V mV uV }

Table 20 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILEname <base_name> (see page 421)	:RECall:FILEname? (see page 421)	<base_name> ::= quoted ASCII string
:RECall:MASK[:START] [<file_spec>] (see page 422)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:PWD <path_name> (see page 423)	:RECall:PWD? (see page 423)	<path_name> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see page 424)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:WMEMory<r>[:START] [<file_name>] (see page 425)	n/a	<r> ::= 1-2 in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 21 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILEname <base_name> (see page 430)	:SAVE:FILEname? (see page 430)	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_name>] (see page 431)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGE:FACTOrs { {0 OFF} {1 ON} } (see page 432)	:SAVE:IMAGE:FACTOrs? (see page 432)	{0 1}
:SAVE:IMAGE:FORMAT <format> (see page 433)	:SAVE:IMAGE:FORMAT? (see page 433)	<format> ::= {TIFF {BMP BMP24bit} BMP8bit PNG NONE}
:SAVE:IMAGE:INKSaver { {0 OFF} {1 ON} } (see page 434)	:SAVE:IMAGE:INKSaver? (see page 434)	{0 1}
:SAVE:IMAGE:PALETTE <palette> (see page 435)	:SAVE:IMAGE:PALETTE? (see page 435)	<palette> ::= {COLOR GRAYscale MONochrome}
:SAVE:LISTER[:START] [<file_name>] (see page 436)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:START] [<file_spec>] (see page 437)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:MULTi[:START] [<file_name>] (see page 438)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 439)	:SAVE:PWD? (see page 439)	<path_name> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see page 440)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:START]] [<file_name>] (see page 441)	n/a	<file_name> ::= quoted ASCII string

Table 21 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WAVEform:FORMAT <format> (see page 442)	:SAVE:WAVEform:FORMAT? ? (see page 442)	<format> ::= {ASCIiXY CSV BINary NONE}
:SAVE:WAVEform:LENGTH <length> (see page 443)	:SAVE:WAVEform:LENGTH? ? (see page 443)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVEform:LENGTH :MAX {{0 OFF} {1 ON}} (see page 444)	:SAVE:WAVEform:LENGTH :MAX? (see page 444)	{0 1}
:SAVE:WAVEform:SEGmented <option> (see page 445)	:SAVE:WAVEform:SEGmented? ? (see page 445)	<option> ::= {ALL CURRent}
:SAVE:WMEMory:SOURce <source> (see page 446)	:SAVE:WMEMory:SOURce? ? (see page 446)	<source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:START] [<file_name>] (see page 447)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 22 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPLAY {{0 OFF} {1 ON}} (see page 452)	:SBUS<n>:DISPLAY? ? (see page 452)	{0 1}
:SBUS<n>:MODE <mode> (see page 453)	:SBUS<n>:MODE? (see page 453)	<mode> ::= {CAN IIC LIN SPI UART}

Table 23 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNT:ER Ror? (see page 456)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:OV ERload? (see page 457)	<frame_count> ::= integer in NR1 format
:SBUS<n>:CAN:COUNT:RE Set (see page 458)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNT:TO Tal? (see page 459)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:UT ILization? (see page 460)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:SAMPLEpo int <value> (see page 461)	:SBUS<n>:CAN:SAMPLEpo int? (see page 461)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:CAN:SIGNAl:B AUDrate <baudrate> (see page 462)	:SBUS<n>:CAN:SIGNAl:B AUDrate? (see page 462)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNAl:D EFinition <value> (see page 463)	:SBUS<n>:CAN:SIGNAl:D EFinition? (see page 463)	<value> ::= {CANH CANL RX TX DIFFerential DIFL DIFH}
:SBUS<n>:CAN:SOURce <source> (see page 464)	:SBUS<n>:CAN:SOURce? (see page 464)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 465)	:SBUS<n>:CAN:TRIGger? (see page 466)	<condition> ::= {SOF DATA ERRor IDData IDEither IDRemote ALLerrors OVERload ACKerror}
:SBUS<n>:CAN:TRIGger: PATtern:DATA <string> (see page 467)	:SBUS<n>:CAN:TRIGger: PATtern:DATA? (see page 467)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:CAN:TRIGger: PATtern:DATA:LENGTH <length> (see page 468)	:SBUS<n>:CAN:TRIGger: PATtern:DATA:LENGTH? (see page 468)	<length> ::= integer from 1 to 8 in NR1 format

Table 23 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger: PATtern:ID <string> (see page 469)	:SBUS<n>:CAN:TRIGger: PATtern:ID? (see page 469)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:CAN:TRIGger: PATtern:ID:MODE <value> (see page 470)	:SBUS<n>:CAN:TRIGger: PATtern:ID:MODE? (see page 470)	<value> ::= {STANDARD EXTENDED}

Table 24 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZE <size> (see page 472)	:SBUS<n>:IIC:ASIZE? (see page 472)	<size> ::= {BIT7 BIT8}
:SBUS<n>:IIC[:SOURce] :CLOCk <source> (see page 473)	:SBUS<n>:IIC[:SOURce] :CLOCk? (see page 473)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC[:SOURce] :DATA <source> (see page 474)	:SBUS<n>:IIC[:SOURce] :DATA? (see page 474)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC:TRIGger: PATtern:ADDRess <value> (see page 475)	:SBUS<n>:IIC:TRIGger: PATtern:ADDRess? (see page 475)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATtern:DATA <value> (see page 476)	:SBUS<n>:IIC:TRIGger: PATtern:DATA? (see page 476)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATtern:DATa2 <value> (see page 477)	:SBUS<n>:IIC:TRIGger: PATtern:DATa2? (see page 477)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: QUALifier <value> (see page 478)	:SBUS<n>:IIC:TRIGger: QUALifier? (see page 478)	<value> ::= {EQUAL NOTEQUAL LESSthan GREATERthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 479)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 479)	<type> ::= {START STOP READ7 READEprom WRITE7 WRITE10 NACKnowledge ANACK R7Data2 W7Data2 RESTart}

Table 25 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:PARity { {0 OFF} {1 ON}} (see page 483)	:SBUS<n>:LIN:PARity? (see page 483)	{0 1}
:SBUS<n>:LIN:SAMPLEpo int <value> (see page 484)	:SBUS<n>:LIN:SAMPLEpo int? (see page 484)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:LIN:SIGNAL:B AUDrate <baudrate> (see page 485)	:SBUS<n>:LIN:SIGNAL:B AUDrate? (see page 485)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 486)	:SBUS<n>:LIN:SOURce? (see page 486)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:LIN:STANDARD <std> (see page 487)	:SBUS<n>:LIN:STANDARD ? (see page 487)	<std> ::= {LIN13 LIN20}
:SBUS<n>:LIN:SYNCbre ak <value> (see page 488)	:SBUS<n>:LIN:SYNCbre ak? (see page 488)	<value> ::= integer = {11 12 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 489)	:SBUS<n>:LIN:TRIGger? (see page 489)	<condition> ::= {SYNCbreak ID DATA}
:SBUS<n>:LIN:TRIGger: ID <value> (see page 490)	:SBUS<n>:LIN:TRIGger: ID? (see page 490)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal
:SBUS<n>:LIN:TRIGger: PATtern:DATA <string> (see page 491)	:SBUS<n>:LIN:TRIGger: PATtern:DATA? (see page 491)	<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when <base> = DECimal <string> ::= "nn...n" where n ::= {0 1 X \$} when <base> = BINary <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$} when <base> = HEX

Table 25 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger: PATtern:DATA:LENGTH <length> (see page 493)	:SBUS<n>:LIN:TRIGger: PATtern:DATA:LENGTH? (see page 493)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger: PATtern:FORMAT <base> (see page 494)	:SBUS<n>:LIN:TRIGger: PATtern:FORMAT? (see page 494)	<base> ::= {BINary HEX DECimal}

Table 26 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 497)	:SBUS<n>:SPI:BITorder ? (see page 497)	<order> ::= {LSBFirst MSBFFirst}
:SBUS<n>:SPI:CLOCK:SL OPe <slope> (see page 498)	:SBUS<n>:SPI:CLOCK:SL OPe? (see page 498)	<slope> ::= {NEGative POSitive}
:SBUS<n>:SPI:CLOCK:TI Meout <time_value> (see page 499)	:SBUS<n>:SPI:CLOCK:TI Meout? (see page 499)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:FRAMing <value> (see page 500)	:SBUS<n>:SPI:FRAMing? (see page 500)	<value> ::= {CHIPselect {NCHipselect NOTC} TIMeout}
:SBUS<n>:SPI:SOURce:CL OCK <source> (see page 501)	:SBUS<n>:SPI:SOURce:CL OCK? (see page 501)	<value> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:SOURce:FR AME <source> (see page 502)	:SBUS<n>:SPI:SOURce:FR AME? (see page 502)	<value> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:SOURce:MI SO <source> (see page 503)	:SBUS<n>:SPI:SOURce:MI SO? (see page 503)	<value> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:SOURce:MO SI <source> (see page 504)	:SBUS<n>:SPI:SOURce:MO SI? (see page 504)	<value> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:TRIGger: PATtern:MISO:DATA <string> (see page 505)	:SBUS<n>:SPI:TRIGger: PATtern:MISO:DATA? (see page 505)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}

Table 26 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:TRIGger: PATtern:MISO:WIDTH <width> (see page 506)	:SBUS<n>:SPI:TRIGger: PATtern:MISO:WIDTH? (see page 506)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger: PATtern:MOsi:DATA <string> (see page 507)	:SBUS<n>:SPI:TRIGger: PATtern:MOsi:DATA? (see page 507)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:SPI:TRIGger: PATtern:MOsi:WIDTH <width> (see page 508)	:SBUS<n>:SPI:TRIGger: PATtern:MOsi:WIDTH? (see page 508)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger: TYPE <value> (see page 509)	:SBUS<n>:SPI:TRIGger: TYPE? (see page 509)	<value> ::= {MOsi MISO}
:SBUS<n>:SPI:WIDTH <word_width> (see page 510)	:SBUS<n>:SPI:WIDTH? (see page 510)	<word_width> ::= integer 4-16 in NR1 format

Table 27 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 513)	:SBUS<n>:UART:BASE? (see page 513)	<base> ::= {ASCii BINary HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 514)	:SBUS<n>:UART:BAUDrate? (see page 514)	<baudrate> ::= integer from 100 to 8000000
:SBUS<n>:UART:BITorde r <bitorder> (see page 515)	:SBUS<n>:UART:BITorde r? (see page 515)	<bitorder> ::= {LSBFirst MSBFirst}
n/a	:SBUS<n>:UART:COUNT:E RRor? (see page 516)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNT:R ESet (see page 517)	n/a	n/a
n/a	:SBUS<n>:UART:COUNT:R XFRAMES? (see page 518)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNT:T XFRAMES? (see page 519)	<frame_count> ::= integer in NR1 format

Table 27 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:FRAMing <value> (see page 520)	:SBUS<n>:UART:FRAMing? (see page 520)	<p><value> ::= {OFF <decimal> <nondecimal>}</p> <p><decimal> ::= 8-bit integer from 0-255 (0x00-0xff)</p> <p><nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal</p> <p><nondecimal> ::= #Bnn...n where n ::= {0 1} for binary</p>
:SBUS<n>:UART:PARity <parity> (see page 521)	:SBUS<n>:UART:PARity? (see page 521)	<parity> ::= {EVEN ODD NONE}
:SBUS<n>:UART:POLarit y <polarity> (see page 522)	:SBUS<n>:UART:POLarit y? (see page 522)	<polarity> ::= {HIGH LOW}
:SBUS<n>:UART:SOURce: RX <source> (see page 523)	:SBUS<n>:UART:SOURce: RX? (see page 523)	<p><source> ::= {CHANnel<n> EXTERNAL}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p>
:SBUS<n>:UART:SOURce: TX <source> (see page 524)	:SBUS<n>:UART:SOURce: TX? (see page 524)	<p><source> ::= {CHANnel<n> EXTERNAL}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p>
:SBUS<n>:UART:TRIGger :BASE <base> (see page 525)	:SBUS<n>:UART:TRIGger :BASE? (see page 525)	<base> ::= {ASCII HEX}
:SBUS<n>:UART:TRIGger :BURSt <value> (see page 526)	:SBUS<n>:UART:TRIGger :BURSt? (see page 526)	<value> ::= {OFF 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger :DATA <value> (see page 527)	:SBUS<n>:UART:TRIGger :DATA? (see page 527)	<p><value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format</p> <p><hexadecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal</p> <p><binary> ::= #Bnn...n where n ::= {0 1} for binary</p> <p><quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)</p>
:SBUS<n>:UART:TRIGger :IDLE <time_value> (see page 528)	:SBUS<n>:UART:TRIGger :IDLE? (see page 528)	<time_value> ::= time from 1 us to 10 s in NR3 format

Table 27 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:TRIGger :QUALifier <value> (see page 529)	:SBUS<n>:UART:TRIGger :QUALifier? (see page 529)	<value> ::= {EQUAL NOTequal GREaterthan LESSthan}
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 530)	:SBUS<n>:UART:TRIGger :TYPE? (see page 530)	<value> ::= {RSTArt RSTOP RDATa RD1 RD0 RDX PARityerror TSTArt TSTOP TDATa TD1 TD0 TDX}
:SBUS<n>:UART:WIDTh <width> (see page 531)	:SBUS<n>:UART:WIDTh? (see page 531)	<width> ::= {5 6 7 8 9}

Table 28 General :SEARCh Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARCh:COUNT? (see page 535)	<count> ::= an integer count value
:SEARCh:MODE <value> (see page 536)	:SEARCh:MODE? (see page 536)	<value> ::= {SERial1}
:SEARCh:STATE <value> (see page 537)	:SEARCh:STATE? (see page 537)	<value> ::= {{0 OFF} {1 ON}}

Table 29 :SEARCh:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARCh:SERial:CAN:MODE <value> (see page 539)	:SEARCh:SERial:CAN:MODE? (see page 539)	<value> ::= {DATA IDData IDEither IDRmote ALLerrors OVERload EROR}
:SEARCh:SERial:CAN:ATTern:DATA <string> (see page 540)	:SEARCh:SERial:CAN:ATTern:DATA? (see page 540)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARCh:SERial:CAN:ATTern:DATA:LENGTH <length> (see page 541)	:SEARCh:SERial:CAN:ATTern:DATA:LENGTH? (see page 541)	<length> ::= integer from 1 to 8 in NR1 format
:SEARCh:SERial:CAN:ATTern:ID <string> (see page 542)	:SEARCh:SERial:CAN:ATTern:ID? (see page 542)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARCh:SERial:CAN:ATTern:ID:MODE <value> (see page 543)	:SEARCh:SERial:CAN:ATTern:ID:MODE? (see page 543)	<value> ::= {STANDARD EXTENDED}

Table 30 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MO DE <value> (see page 545)	:SEARch:SERial:IIC:MO DE? (see page 545)	<value> ::= { READ7 WRITE7 NACKnowledge ANACK R7Data2 W7Data2 RESTart READEprom}
:SEARch:SERial:IIC:PA TTern:ADDReSS <value> (see page 547)	:SEARch:SERial:IIC:PA TTern:ADDReSS? (see page 547)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:PA TTern:DATA <value> (see page 548)	:SEARch:SERial:IIC:PA TTern:DATA? (see page 548)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:PA TTern:DATA2 <value> (see page 549)	:SEARch:SERial:IIC:PA TTern:DATA2? (see page 549)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:QU ALifier <value> (see page 550)	:SEARch:SERial:IIC:QU ALifier? (see page 550)	<value> ::= { EQUAL NOTEQUAL LESSthan GREATERthan}

Table 31 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 552)	:SEARch:SERial:LIN:ID ? (see page 552)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal
:SEARch:SERial:LIN:MO DE <value> (see page 553)	:SEARch:SERial:LIN:MO DE? (see page 553)	<value> ::= { ID DATA ERROR}

Table 31 :SEARch:SERial:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:PA TTern:DATA <string> (see page 554)	:SEARch:SERial:LIN:PA TTern:DATA? (see page 554)	When :SEARch:SERial:LIN:PATTern:FORMat DECimal, <string> ::= "n" where n ::= 32-bit integer in unsigned decimal, returns "\$" if data has any don't cares When :SEARch:SERial:LIN:PATTern:FORMat HEX, <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARch:SERial:LIN:PA TTern:DATA:LENGTH <length> (see page 555)	:SEARch:SERial:LIN:PA TTern:DATA:LENGTH? (see page 555)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PA TTern:FORMAT <base> (see page 556)	:SEARch:SERial:LIN:PA TTern:FORMAT? (see page 556)	<base> ::= {HEX DECimal}

Table 32 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MO DE <value> (see page 558)	:SEARch:SERial:SPI:MO DE? (see page 558)	<value> ::= {MOSI MISO}
:SEARch:SERial:SPI:PA TTern:DATA <string> (see page 559)	:SEARch:SERial:SPI:PA TTern:DATA? (see page 559)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARch:SERial:SPI:PA TTern:WIDTH <width> (see page 560)	:SEARch:SERial:SPI:PA TTern:WIDTH? (see page 560)	<width> ::= integer from 1 to 10

Table 33 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:D ATA <value> (see page 562)	:SEARch:SERial:UART:D ATA? (see page 562)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SEARch:SERial:UART:M ODE <value> (see page 563)	:SEARch:SERial:UART:M ODE? (see page 563)	<value> ::= {RDATA RD1 RD0 RDX TDATA TD1 TD0 TDX PARityerror AERRor}
:SEARch:SERial:UART:Q UALifier <value> (see page 564)	:SEARch:SERial:UART:Q UALifier? (see page 564)	<value> ::= {EQUAL NOTEqual GREaterthan LESSthan}

Table 34 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 567)	:SYSTem:DATE? (see page 567)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,...,12 JANuary FEBruary MARch APRil MAY JUNe JULy AUGust SEPtember OCTober NOVember DECember} <day> ::= {1,...31}
:SYSTem:DSP <string> (see page 568)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 569)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 753).
:SYSTem:LOCK <value> (see page 570)	:SYSTem:LOCK? (see page 570)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:MENU <menu> (see page 571)	n/a	<menu> ::= {MASK MEASure SEGmented LISTer}

Table 34 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:PRESet (see page 572)	n/a	See :SYSTem:PRESet (see page 572)
:SYSTem:PROTection:LOCK <value> (see page 575)	:SYSTem:PROTection:LOCK? (see page 575)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:SETup <setup_data> (see page 576)	:SYSTem:SETup? (see page 576)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 578)	:SYSTem:TIME? (see page 578)	<time> ::= hours,minutes,seconds in NR1 format

Table 35 :TIMEbase Commands Summary

Command	Query	Options and Query Returns
:TIMEbase:MODE <value> (see page 581)	:TIMEbase:MODE? (see page 581)	<value> ::= {MAIN WINDOW XY ROLL}
:TIMEbase:POSITION <pos> (see page 582)	:TIMEbase:POSITION? (see page 582)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMEbase:RANGE <range_value> (see page 583)	:TIMEbase:RANGE? (see page 583)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMEbase:REFERENCE {LEFT CENTER RIGHT} (see page 584)	:TIMEbase:REFERENCE? (see page 584)	<return_value> ::= {LEFT CENTER RIGHT}
:TIMEbase:SCALE <scale_value> (see page 585)	:TIMEbase:SCALE? (see page 585)	<scale_value> ::= time/div in seconds in NR3 format
:TIMEbase:VERNier {{0 OFF} {1 ON}} (see page 586)	:TIMEbase:VERNier? (see page 586)	{0 1}
:TIMEbase:WINDOW:POSITION <pos> (see page 587)	:TIMEbase:WINDOW:POSITION? (see page 587)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMEbase:WINDOW:RANGE <range_value> (see page 588)	:TIMEbase:WINDOW:RANGE? (see page 588)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMEbase:WINDOW:SCALE <scale_value> (see page 589)	:TIMEbase:WINDOW:SCALE? (see page 589)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Table 36 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 594)	n/a	n/a
:TRIGger:HFReject {{0 OFF} {1 ON}} (see page 595)	:TRIGger:HFReject? (see page 595)	{0 1}
:TRIGger:HOLDoff <holdoff_time> (see page 596)	:TRIGger:HOLDoff? (see page 596)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:ASETup (see page 597)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 598)	:TRIGger:LEVel:HIGH? <source> (see page 598)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see page 599)	:TRIGger:LEVel:LOW? <source> (see page 599)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:MODE <mode> (see page 600)	:TRIGger:MODE? (see page 600)	<mode> ::= {EDGE GLITch PATTern TV} <return_value> ::= {<mode> <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREJect {{0 OFF} {1 ON}} (see page 601)	:TRIGger:NREJect? (see page 601)	{0 1}
:TRIGger:SWEep <sweep> (see page 602)	:TRIGger:SWEep? (see page 602)	<sweep> ::= {AUTO NORMAl}

Table 37 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC DC LFReject} (see page 604)	:TRIGger[:EDGE]:COUPling? (see page 604)	{AC DC LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 605)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 605)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= +(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> EXTERNAL} for DSO models <source> ::= {CHANnel<n> DIGital<d> EXTERNAL } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE]:REJect {OFF LFReject HFReject} (see page 606)	:TRIGger[:EDGE]:REJect? (see page 606)	{OFF LFReject HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 607)	:TRIGger[:EDGE]:SLOPe? (see page 607)	<polarity> ::= {POSitive NEGative EITHer ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 608)	:TRIGger[:EDGE]:SOURce? (see page 608)	<source> ::= {CHANnel<n> EXTERNAL LINE WGEN} for the DSO models <source> ::= {CHANnel<n> DIGital<d> EXTERNAL LINE WGEN} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 38 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 611)	:TRIGger:GLITch:GREaterthan? (see page 611)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 612)	:TRIGger:GLITch:LESSthan? (see page 612)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 613)	:TRIGger:GLITch:LEVel? (see page 613)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> EXTERNAL} for DSO models <source> ::= {CHANnel<n> DIGItal<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 614)	:TRIGger:GLITch:POLarity? (see page 614)	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 615)	:TRIGger:GLITch:QUALifier? (see page 615)	<qualifier> ::= {GREaterthan LESSthan RANGE}

Table 38 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 616)	:TRIGger:GLITch:RANGE? ? (see page 616)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:SOURce <source> (see page 617)	:TRIGger:GLITch:SOURce? ? (see page 617)	<source> ::= {CHANnel<n> DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 39 :TRIGger:PATTERn Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTERn <string>[,<edge_source>,<edge>] (see page 619)	:TRIGger:PATTERn? ? (see page 620)	<string> ::= "nn...n" where n ::= {0 1 X R F} when <base> = ASCII <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$} when <base> = HEX <edge_source> ::= {CHANnel<n> NONE} for DSO models <edge_source> ::= {CHANnel<n> DIGital<d> NONE} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive NEGative}
:TRIGger:PATTERn:FORM at <base> (see page 621)	:TRIGger:PATTERn:FORM at? ? (see page 621)	<base> ::= {ASCII HEX}
:TRIGger:PATTERn:QUALifier <qualifier> (see page 622)	:TRIGger:PATTERn:QUALifier? ? (see page 622)	<qualifier> ::= ENTERed

Table 40 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 624)	:TRIGger:TV:LINE? (see page 624)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 625)	:TRIGger:TV:MODE? (see page 625)	<tv mode> ::= {FIELD1 FIELD2 AFIELDS ALINES LFIELD1 LFIELD2 LATERNATE}
:TRIGger:TV:POLarity <polarity> (see page 626)	:TRIGger:TV:POLarity? (see page 626)	<polarity> ::= {POSITIVE NEGATIVE}
:TRIGger:TV:SOURce <source> (see page 627)	:TRIGger:TV:SOURce? (see page 627)	<source> ::= {CHANNEL<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANDARD <standard> (see page 628)	:TRIGger:TV:STANDARD? (see page 628)	<standard> ::= {NTSC PAL PALM SECAM}

Table 41 :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see page 637)	:WAVEform:BYTeorder? (see page 637)	<value> ::= {LSBFIRST MSBFIRST}
n/a	:WAVEform:COUNT? (see page 638)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see page 639)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVEform:FORMAT <value> (see page 641)	:WAVEform:FORMAT? (see page 641)	<value> ::= {WORD BYTE ASCII}

Table 41 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:POINTs <# points> (see page 642)	:WAVEform:POINTs? (see page 642)	<# points> ::= {100 250 500 1000 <points_mode>} if waveform points mode is NORMAl <# points> ::= {100 250 500 1000 2000 ... 8000000 in 1-2-5 sequence <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMAl MAXimum RAW}
:WAVEform:POINTs:MODE <points_mode> (see page 644)	:WAVEform:POINTs:MODE? (see page 644)	<points_mode> ::= {NORMAl MAXimum RAW}
n/a	:WAVEform:PREamble? (see page 646)	<preamble_block> ::= <format NR1>, <type NR1>, <points NR1>, <count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>, <yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: <ul style="list-style-type: none">• 0 for BYTE format• 1 for WORD format• 2 for ASCii format <type> ::= an integer in NR1 format: <ul style="list-style-type: none">• 0 for NORMAL type• 1 for PEAK detect type• 3 for AVERAGE type• 4 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format
n/a	:WAVEform:SEGmented:COUNT? (see page 649)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see page 650)	<time_tag> ::= in NR3 format (with Option SGM)

Table 41 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:SOURce <source> (see page 651)	:WAVEform:SOURce? (see page 651)	<p><source> ::= {CHANnel<n> FUNCTION MATH SBUS1} for DSO models</p> <p><source> ::= {CHANnel<n> POD{1 2} BUS{1 2} FUNCTION MATH SBUS1} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p>
:WAVEform:SOURce:SUBS ource <subsource> (see page 655)	:WAVEform:SOURce:SUBS ource? (see page 655)	<subsource> ::= {{SUB0 RX MOSI} {SUB1 TX MISO}}
n/a	:WAVEform:TYPE? (see page 656)	<return_mode> ::= {NORM PEAK AVER HRES}
:WAVEform:UNSIGNED { {0 OFF} {1 ON} } (see page 657)	:WAVEform:UNSIGNED? (see page 657)	{0 1}
:WAVEform:VIEW <view> (see page 658)	:WAVEform:VIEW? (see page 658)	<view> ::= {MAIN}
n/a	:WAVEform:XINCrement? (see page 659)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVEform:XORigin? (see page 660)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVEform:XREFerence? (see page 661)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVEform:YINCrement? (see page 662)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVEform:YORigin? (see page 663)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVEform:YREFerence? (see page 664)	<return_value> ::= y-reference value in the current preamble in NR1 format

Table 42 :WGEN Commands Summary

Command	Query	Options and Query Returns
:WGEN:FREQuency <frequency> (see page 668)	:WGEN:FREQuency? (see page 668)	<frequency> ::= frequency in Hz in NR3 format
:WGEN:FUNCTION <signal> (see page 669)	:WGEN:FUNCTION? (see page 670)	<signal> ::= {SINusoid SQUare RAMP PULSe NOISe DC}
:WGEN:FUNCTION:PULSe: WIDTh <width> (see page 671)	:WGEN:FUNCTION:PULSe: WIDTh? (see page 671)	<width> ::= pulse width in seconds in NR3 format
:WGEN:FUNCTION:RAMP:S YMMetry <percent> (see page 672)	:WGEN:FUNCTION:RAMP:S YMMetry? (see page 672)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format
:WGEN:FUNCTION:SQUare :DCYCle <percent> (see page 673)	:WGEN:FUNCTION:SQUare :DCYCle? (see page 673)	<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
:WGEN:MODulation:AM:D EPTH <percent> (see page 674)	:WGEN:MODulation:AM:D EPTH? (see page 674)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format
:WGEN:MODulation:AM:F REQuency <frequency> (see page 675)	:WGEN:MODulation:AM:F REQuency? (see page 675)	<frequency> ::= modulating waveform frequency in Hz in NR3 format
:WGEN:MODulation:FM:D EViation <frequency> (see page 676)	:WGEN:MODulation:FM:D EViation? (see page 676)	<frequency> ::= frequency deviation in Hz in NR3 format
:WGEN:MODulation:FM:F REQuency <frequency> (see page 677)	:WGEN:MODulation:FM:F REQuency? (see page 677)	<frequency> ::= modulating waveform frequency in Hz in NR3 format
:WGEN:MODulation:FSKe y:FREQuency <percent> (see page 678)	:WGEN:MODulation:FSKe y:FREQuency? (see page 678)	<frequency> ::= hop frequency in Hz in NR3 format
:WGEN:MODulation:FSKe y:RATE <rate> (see page 679)	:WGEN:MODulation:FSKe y:RATE? (see page 679)	<rate> ::= FSK modulation rate in Hz in NR3 format
:WGEN:MODulation:FUNC tion <shape> (see page 680)	:WGEN:MODulation:FUNC tion? (see page 680)	<shape> ::= {SINusoid SQUare RAMP}
:WGEN:MODulation:FUNC tion:RAMP:SYMMetry <percent> (see page 681)	:WGEN:MODulation:FUNC tion:RAMP:SYMMetry? (see page 681)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format

Table 42 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:MODulation:NOISE <percent> (see page 682)	:WGEN:MODulation:NOISE? (see page 682)	<percent> ::= 0 to 100
:WGEN:MODulation:STATE {{0 OFF} {1 ON}} (see page 683)	:WGEN:MODulation:STATE? (see page 683)	{0 1}
:WGEN:MODulation:TYPE <type> (see page 684)	:WGEN:MODulation:TYPE? (see page 684)	<type> ::= {AM FM FSK}
:WGEN:OUTPut {{0 OFF} {1 ON}} (see page 686)	:WGEN:OUTPut? (see page 686)	{0 1}
:WGEN:OUTPut:LOAD <impedance> (see page 687)	:WGEN:OUTPut:LOAD? (see page 687)	<impedance> ::= {ONEMeg FIFTy}
:WGEN:PERiod <period> (see page 688)	:WGEN:PERiod? (see page 688)	<period> ::= period in seconds in NR3 format
:WGEN:RST (see page 689)	n/a	n/a
:WGEN:VOLTage <amplitude> (see page 690)	:WGEN:VOLTage? (see page 690)	<amplitude> ::= amplitude in volts in NR3 format
:WGEN:VOLTage:HIGH <high> (see page 691)	:WGEN:VOLTage:HIGH? (see page 691)	<high> ::= high-level voltage in volts, in NR3 format
:WGEN:VOLTage:LOW <low> (see page 692)	:WGEN:VOLTage:LOW? (see page 692)	<low> ::= low-level voltage in volts, in NR3 format
:WGEN:VOLTage:OFFSet <offset> (see page 693)	:WGEN:VOLTage:OFFSet? (see page 693)	<offset> ::= offset in volts in NR3 format

Table 43 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 697)	n/a	<r> ::= 1-2 in NR1 format
:WMEMory<r>:DISPlay {{0 OFF} {1 ON}} (see page 698)	:WMEMory<r>:DISPlay? (see page 698)	<r> ::= 1-2 in NR1 format {0 1}

Table 43 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:LABEL <string> (see page 699)	:WMEMory<r>:LABEL? (see page 699)	<r> ::= 1-2 in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 700)	n/a	<r> ::= 1-2 in NR1 format <source> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1 to (# analog channels) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 701)	:WMEMory<r>:SKEW? (see page 701)	<r> ::= 1-2 in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMory<r>:YOFFset <offset>[suffix] (see page 702)	:WMEMory<r>:YOFFset? (see page 702)	<r> ::= 1-2 in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V mV}
:WMEMory<r>:YRANGE <range>[suffix] (see page 703)	:WMEMory<r>:YRANGE? (see page 703)	<r> ::= 1-2 in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V mV}
:WMEMory<r>:YScale <scale>[suffix] (see page 704)	:WMEMory<r>:YScale? (see page 704)	<r> ::= 1-2 in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V mV}

Syntax Elements

- "[Number Format](#)" on page 116
- "[<NL> \(Line Terminator\)](#)" on page 116
- "[\[\] \(Optional Syntax Terms\)](#)" on page 116
- "[{ } \(Braces\)](#)" on page 116
- "[::= \(Defined As\)](#)" on page 116
- "[<> \(Angle Brackets\)](#)" on page 117
- "[... \(Ellipsis\)](#)" on page 117
- "[n,...,p \(Value Ranges\)](#)" on page 117
- "[d \(Digits\)](#)" on page 117
- "[Quoted ASCII String](#)" on page 117
- "[Definite-Length Block Response Data](#)" on page 117

Number Format

NR1 specifies integer data.

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

<NL> (Line Terminator)

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

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

[] (Optional Syntax Terms)

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

{ } (Braces)

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

::= (Defined As)

::= means "defined as".

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

< > (Angle Brackets)

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

... (Ellipsis)

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

n,...,p (Value Ranges)

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

d (Digits)

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

Quoted ASCII String

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

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

has a quoted ASCII string of:

```
'one'
```

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

Definite-Length Block Response Data

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

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

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

8 is the number of digits that follow

00001000 is the number of bytes to be transmitted

<1000 bytes of data> is the actual data

5 Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments.
See "[Introduction to Common \(*\) Commands](#)" on page 121.

Table 44 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 123)	n/a	n/a
*ESE <mask> (see page 124)	*ESE? (see page 125)	<mask> ::= 0 to 255; an integer in NR1 format: Bit Weight Name Enables --- ----- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete
n/a	*ESR? (see page 126)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 126)	AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 129)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 130)	*OPC? (see page 130)	ASCII "1" is placed in the output queue when all pending device operations have completed.

Table 44 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns																																				
n/a	*OPT? (see page 131)	<pre> <return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <MSO>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Power Measurements>, <reserved>, <Segmented Memory>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Waveform Generator>, <reserved>, <reserved> <All field> ::= {0 All} <reserved> ::= 0 <MSO> ::= {0 MSO} <Power Measurements> ::= {0 PWR} <Segmented Memory> ::= {0 SGM} <Mask Test> ::= {0 MASK} <Bandwidth> ::= {0 BW10 BW20} <Waveform Generator> ::= {0 WAVEGEN} </pre>																																				
*RCL <value> (see page 133)	n/a	<pre> <value> ::= {0 1 4 5 6 7 8 9} </pre>																																				
*RST (see page 134)	n/a	See *RST (Reset) (see page 134)																																				
*SAV <value> (see page 137)	n/a	<pre> <value> ::= {0 1 4 5 6 7 8 9} </pre>																																				
*SRE <mask> (see page 138)	*SRE? (see page 139)	<p><mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>Enables</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>Operation Status Reg</td> </tr> <tr> <td>6</td> <td>64</td> <td>----</td> <td>(Not used.)</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td>Event Status Bit</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td>Message Available</td> </tr> <tr> <td>3</td> <td>8</td> <td>----</td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td>Message</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td>User</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td>Trigger</td> </tr> </tbody> </table>	Bit	Weight	Name	Enables	7	128	OPER	Operation Status Reg	6	64	----	(Not used.)	5	32	ESB	Event Status Bit	4	16	MAV	Message Available	3	8	----	(Not used.)	2	4	MSG	Message	1	2	USR	User	0	1	TRG	Trigger
Bit	Weight	Name	Enables																																			
7	128	OPER	Operation Status Reg																																			
6	64	----	(Not used.)																																			
5	32	ESB	Event Status Bit																																			
4	16	MAV	Message Available																																			
3	8	----	(Not used.)																																			
2	4	MSG	Message																																			
1	2	USR	User																																			
0	1	TRG	Trigger																																			

Table 44 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns																																													
n/a	*STB? (see page 140)	<p><value> ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>"1"</th> <th>Indicates</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>---</td> <td>Operation status condition occurred.</td> </tr> <tr> <td>6</td> <td>64</td> <td>RQS/</td> <td>MSS</td> <td>Instrument is requesting service.</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td></td> <td>Enabled event status condition occurred.</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td></td> <td>Message available.</td> </tr> <tr> <td>3</td> <td>8</td> <td>----</td> <td></td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td></td> <td>Message displayed.</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td></td> <td>User event condition occurred.</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td></td> <td>A trigger occurred.</td> </tr> </tbody> </table>	Bit	Weight	Name	"1"	Indicates	7	128	OPER	---	Operation status condition occurred.	6	64	RQS/	MSS	Instrument is requesting service.	5	32	ESB		Enabled event status condition occurred.	4	16	MAV		Message available.	3	8	----		(Not used.)	2	4	MSG		Message displayed.	1	2	USR		User event condition occurred.	0	1	TRG		A trigger occurred.
Bit	Weight	Name	"1"	Indicates																																											
7	128	OPER	---	Operation status condition occurred.																																											
6	64	RQS/	MSS	Instrument is requesting service.																																											
5	32	ESB		Enabled event status condition occurred.																																											
4	16	MAV		Message available.																																											
3	8	----		(Not used.)																																											
2	4	MSG		Message displayed.																																											
1	2	USR		User event condition occurred.																																											
0	1	TRG		A trigger occurred.																																											
*TRG (see page 142)	n/a	n/a																																													
n/a	*TST? (see page 143)	<result> ::= 0 or non-zero value; an integer in NR1 format																																													
*WAI (see page 144)	n/a	n/a																																													

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

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

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

NOTE

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

*CLS (Clear Status)

 (see [page 794](#))

Command Syntax *CLS

The *CLS common command clears the status data structures, the device-defined error queue, and the Request-for-OPC flag.

NOTE

If the *CLS command immediately follows a program message terminator, the output queue and the MAV (message available) bit are cleared.

See Also

- "[Introduction to Common \(*\) Commands](#)" on page 121
- "[*STB \(Read Status Byte\)](#)" on page 140
- "[*ESE \(Standard Event Status Enable\)](#)" on page 124
- "[*ESR \(Standard Event Status Register\)](#)" on page 126
- "[*SRE \(Service Request Enable\)](#)" on page 138
- "[:SYSTem:ERRor](#)" on page 569

*ESE (Standard Event Status Enable)

C (see [page 794](#))

Command Syntax *ESE <mask_argument>

<mask_argument> ::= integer from 0 to 255

The *ESE common command sets the bits in the Standard Event Status Enable Register. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A "1" in the Standard Event Status Enable Register enables the corresponding bit in the Standard Event Status Register. A zero disables the bit.

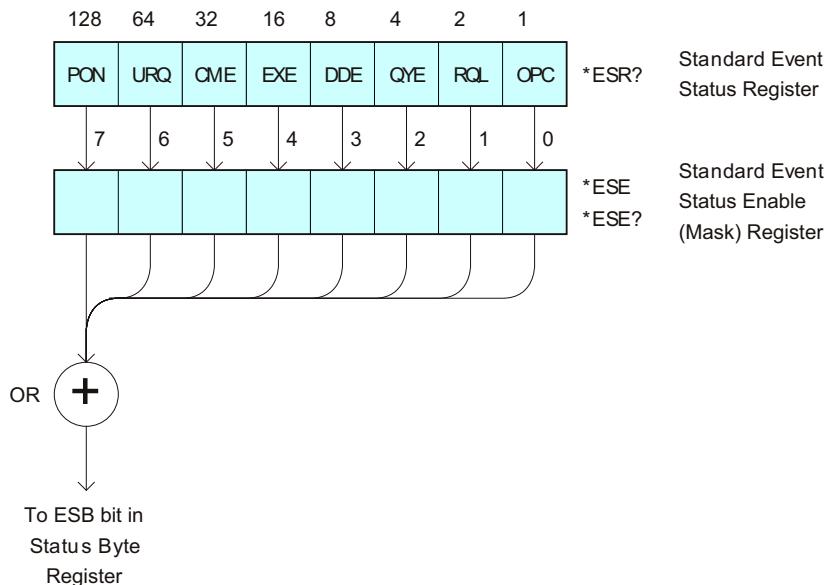


Table 45 Standard Event Status Enable (ESE)

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

Query Syntax *ESE?

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

Return Format <mask_argument><NL>

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

- See Also**
- "[Introduction to Common \(*\) Commands](#)" on page 121
 - "[*ESR \(Standard Event Status Register\)](#)" on page 126
 - "[*OPC \(Operation Complete\)](#)" on page 130
 - "[*CLS \(Clear Status\)](#)" on page 123

*ESR (Standard Event Status Register)

 (see page 794)

Query Syntax *ESR?

The *ESR? query returns the contents of the Standard Event Status Register. When you read the Event Status Register, the value returned is the total bit weights of all of the bits that are high at the time you read the byte. Reading the register clears the Event Status Register.

The following table shows bit weight, name, and condition for each bit.

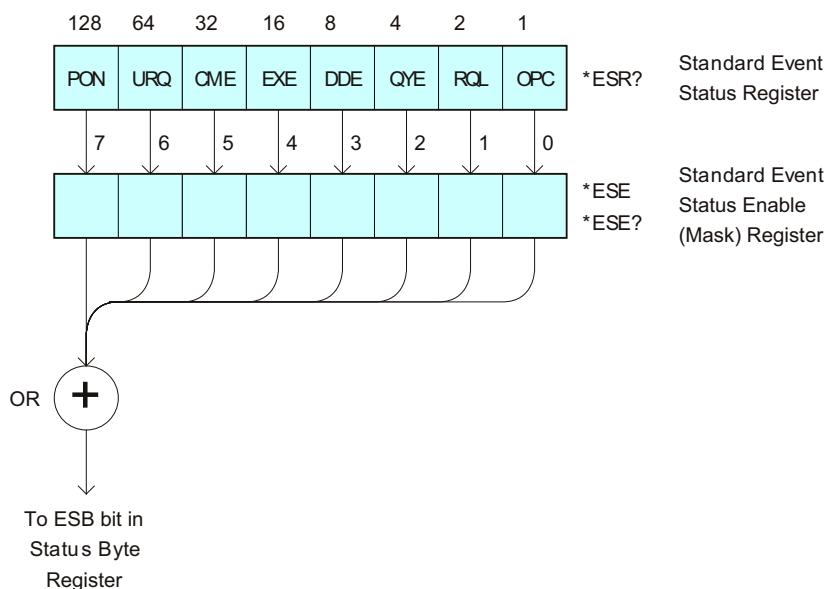


Table 46 Standard Event Status Register (ESR)

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

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

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

- See Also**
- ["Introduction to Common \(*\) Commands"](#) on page 121
 - ["*ESE \(Standard Event Status Enable\)"](#) on page 124
 - ["*OPC \(Operation Complete\)"](#) on page 130
 - ["*CLS \(Clear Status\)"](#) on page 123
 - [":SYSTem:ERRor"](#) on page 569

*IDN (Identification Number)

 (see [page 794](#))

Query Syntax *IDN?

The *IDN? query identifies the instrument type and software version.

Return Format AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <NL>

<model> ::= the model number of the instrument

<serial number> ::= the serial number of the instrument

X.XX.XX ::= the software revision of the instrument

See Also • "[Introduction to Common \(*\) Commands](#)" on page 121
• "["*OPT \(Option Identification\)](#)" on page 131

*LRN (Learn Device Setup)



(see [page 794](#))

Query Syntax

`*LRN?`

The `*LRN?` query result contains the current state of the instrument. This query is similar to the `:SYST:SETUp?` (see [page 576](#)) query, except that it contains `":SYST:SET "` before the binary block data. The query result is a valid command that can be used to restore instrument settings at a later time.

Return Format

```
<learn_string><NL>
<learn_string> ::= :SYST:SET <setup_data>
<setup_data> ::= binary block data in IEEE 488.2 # format
```

`<learn_string>` specifies the current instrument setup. The block size is subject to change with different firmware revisions.

NOTE

The `*LRN?` query return format has changed from previous Keysight oscilloscopes to match the IEEE 488.2 specification which says that the query result must contain `":SYST:SET "` before the binary block data.

See Also

- ["Introduction to Common \(*\) Commands"](#) on page 121
- ["*RCL \(Recall\)"](#) on page 133
- ["*SAV \(Save\)"](#) on page 137
- [":SYST:SETUp"](#) on page 576

*OPC (Operation Complete)

 (see [page 794](#))

Command Syntax *OPC

The *OPC command sets the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

Query Syntax *OPC?

The *OPC? query places an ASCII "1" in the output queue when all pending device operations have completed. The interface hangs until this query returns.

Return Format <complete><NL>

<complete> ::= 1

See Also

- "[Introduction to Common \(*\) Commands](#)" on page 121
- "[*ESE \(Standard Event Status Enable\)](#)" on page 124
- "[*ESR \(Standard Event Status Register\)](#)" on page 126
- "[*CLS \(Clear Status\)](#)" on page 123

*OPT (Option Identification)

C (see page 794)

Query Syntax *OPT?

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

Return Format 0,0,<license info>

```
<license info> ::= <All field>, <reserved>, <MSO>, <reserved>,
                  <Memory>, <Low Speed Serial>, <Automotive Serial>,
                  <reserved>, <reserved>, <reserved>, <reserved>,
                  <RS-232/UART Serial>, <Segmented Memory>,
                  <Mask Test>, <reserved>, <Bandwidth>, <reserved>,
                  <reserved>, <reserved>, <reserved>, <reserved>,
                  <Educator's Kit>, <Waveform Generator>,
                  <reserved>, <reserved>, <reserved>, <reserved>,
                  <reserved>, <reserved>, <Digital Voltmeter>,
                  <reserved>

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

<reserved> ::= 0

<MSO> ::= {0 | MSO}

<Memory> ::= {0 | MEMUP}

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

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

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

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

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

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

<Educator's Kit> ::= {0 | EDK}

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

<Digital Voltmeter> ::= {0 | DVM}
```

The *OPT? query returns the following:

See Also • "Introduction to Common (*) Commands" on page 121

5 Common (*) Commands

- ["*IDN \(Identification Number\)" on page 128](#)

*RCL (Recall)

 (see [page 794](#))

Command Syntax `*RCL <value>`

`<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}`

The *RCL command restores the state of the instrument from the specified save/recall register.

See Also

- ["Introduction to Common \(*\) Commands"](#) on page 121
- ["*SAV \(Save\)"](#) on page 137

*RST (Reset)

 (see [page 794](#))

Command Syntax *RST

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

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

Reset conditions are:

Acquire Menu	
Mode	Normal
Averaging	Off
# Averages	8

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

Cursor Menu	
Source	Channel 1

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

Display Menu	
Persistence	Off
Grid	20%

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

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

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

- See Also**
- "[Introduction to Common \(*\) Commands](#)" on page 121
 - "[:SYSTem:PRESet](#)" on page 572

Example Code

```

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

```

See complete example programs at: [Chapter 38, “Programming Examples,”](#) starting on page 803

*SAV (Save)



(see [page 794](#))

Command Syntax *SAV <value>

```
<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
```

The *SAV command stores the current state of the instrument in a save register. The data parameter specifies the register where the data will be saved.

See Also

- ["Introduction to Common \(*\) Commands"](#) on page 121
- ["*RCL \(Recall\)"](#) on page 133

*SRE (Service Request Enable)

 (see page 794)

Command Syntax *SRE <mask>

<mask> ::= integer with values defined in the following table.

The *SRE command sets the bits in the Service Request Enable Register. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A zero disables the bit.

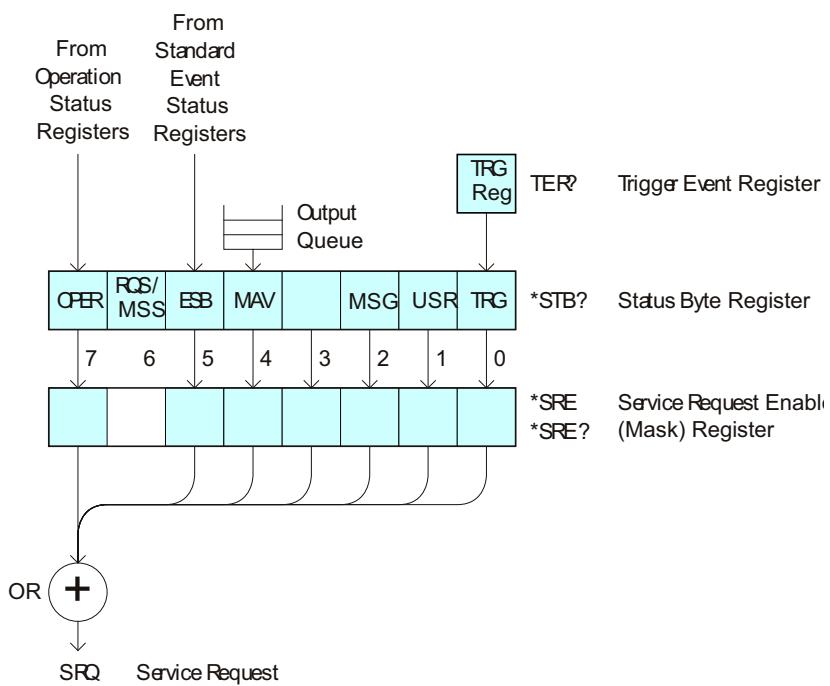


Table 47 Service Request Enable Register (SRE)

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

Query Syntax *SRE?

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

Return Format <mask><NL>

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

See Also

- ["Introduction to Common \(*\) Commands"](#) on page 121
- ["*STB \(Read Status Byte\)"](#) on page 140
- ["*CLS \(Clear Status\)"](#) on page 123

***STB (Read Status Byte)**

C (see page 794)

Query Syntax *STB?

The *STB? query returns the current value of the instrument's status byte. The MSS (Master Summary Status) bit is reported on bit 6 instead of the RQS (request service) bit. The MSS indicates whether or not the device has at least one reason for requesting service.

Return Format <value><NL>

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

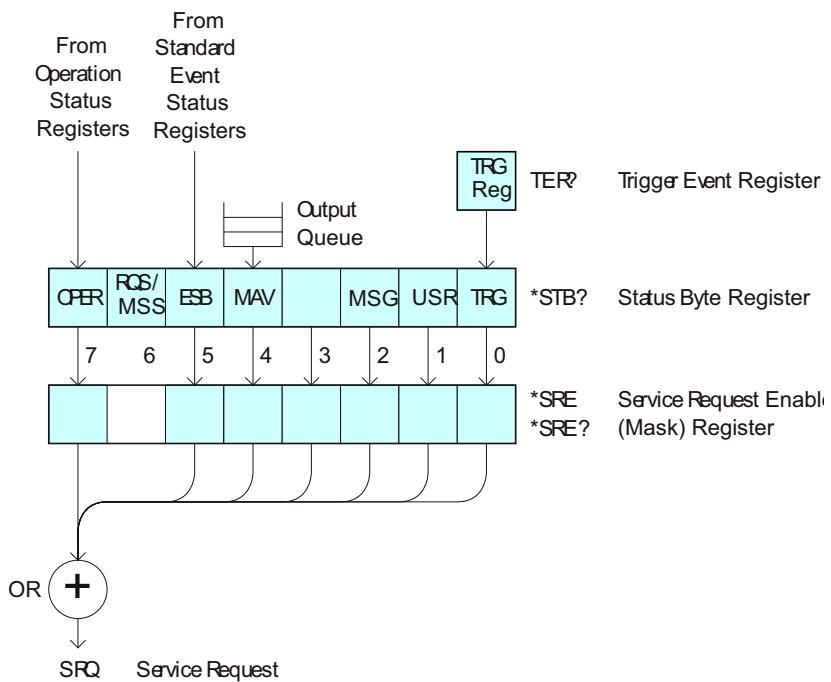


Table 48 Status Byte Register (STB)

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

NOTE

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

See Also

- ["Introduction to Common \(*\) Commands"](#) on page 121
- ["*SRE \(Service Request Enable\)"](#) on page 138

*TRG (Trigger)

 (see [page 794](#))

Command Syntax *TRG

The *TRG command has the same effect as the :DIGITIZE command with no parameters.

See Also

- ["Introduction to Common \(*\) Commands"](#) on page 121
- [":DIGITIZE"](#) on page 157
- [":RUN"](#) on page 174
- [":STOP"](#) on page 178

***TST (Self Test)** (see [page 794](#))**Query Syntax** *TST?

The *TST? query performs a self-test on the instrument. The result of the test is placed in the output queue. A zero indicates the test passed and a non-zero indicates the test failed. If the test fails, refer to the troubleshooting section of the *Service Guide*.

Return Format <result><NL>

<result> ::= 0 or non-zero value; an integer in NR1 format

See Also · ["Introduction to Common \(*\) Commands"](#) on page 121

*WAI (Wait To Continue)

 (see [page 794](#))

Command Syntax *WAI

The *WAI command has no function in the oscilloscope, but is parsed for compatibility with other instruments.

See Also • ["Introduction to Common \(*\) Commands"](#) on page 121

6 Root (:) Commands

Control many of the basic functions of the oscilloscope and reside at the root level of the command tree. See "[Introduction to Root \(:\) Commands](#)" on page 148.

Table 49 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 149)	:ACTivity? (see page 149)	<return value> ::= <edges>,<levels> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
n/a	:AER? (see page 150)	{0 1}; an integer in NR1 format
:AUToscale [<source>[,...,<source>]] (see page 151)	n/a	<source> ::= CHANnel<n> for DSO models <source> ::= {CHANnel<n> DIGItal<d> POD1 POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUToscale:AMODE <value> (see page 153)	:AUToscale:AMODE? (see page 153)	<value> ::= {NORMal CURRent}
:AUToscale:CHANnels <value> (see page 154)	:AUToscale:CHANnels? (see page 154)	<value> ::= {ALL DISPlayed}
:AUToscale:FDEBug {{0 OFF} {1 ON}} (see page 155)	:AUToscale:FDEBug? (see page 155)	{0 1}

Table 49 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:BLANK [<source>] (see page 156)	n/a	<p><source> ::= {CHANnel<n>} FUNCTION MATH SBUS1 WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n>} DIGItal<d> POD{1 2} BUS{1 2} FUNCTION MATH SBUS1 WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p>
:DIGItize [<source>[,...,<source>]] (see page 157)	n/a	<p><source> ::= {CHANnel<n>} FUNCTION MATH} for DSO models</p> <p><source> ::= {CHANnel<n>} DIGItal<d> POD{1 2} BUS{1 2} FUNCTION MATH} for MSO models</p> <p><source> can be repeated up to 5 times</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p>
:MTEenable <n> (see page 159)	:MTEenable? (see page 159)	<n> ::= 16-bit integer in NR1 format
n/a	:MTERegister[:EVENT] ? (see page 161)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 163)	:OPEE? (see page 164)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDiti on? (see page 165)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister[:EVENT] ? (see page 167)	<n> ::= 15-bit integer in NR1 format

Table 49 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns																																	
:OVLenable <mask> (see page 169)	:OVLenable? (see page 170)	<p><mask> ::= 16-bit integer in NR1 format as shown:</p> <table> <thead> <tr> <th>Bit</th><th>Weight</th><th>Input</th></tr> </thead> <tbody> <tr><td>10</td><td>1024</td><td>Ext Trigger Fault</td></tr> <tr><td>9</td><td>512</td><td>Channel 4 Fault</td></tr> <tr><td>8</td><td>256</td><td>Channel 3 Fault</td></tr> <tr><td>7</td><td>128</td><td>Channel 2 Fault</td></tr> <tr><td>6</td><td>64</td><td>Channel 1 Fault</td></tr> <tr><td>4</td><td>16</td><td>Ext Trigger OVL</td></tr> <tr><td>3</td><td>8</td><td>Channel 4 OVL</td></tr> <tr><td>2</td><td>4</td><td>Channel 3 OVL</td></tr> <tr><td>1</td><td>2</td><td>Channel 2 OVL</td></tr> <tr><td>0</td><td>1</td><td>Channel 1 OVL</td></tr> </tbody> </table>	Bit	Weight	Input	10	1024	Ext Trigger Fault	9	512	Channel 4 Fault	8	256	Channel 3 Fault	7	128	Channel 2 Fault	6	64	Channel 1 Fault	4	16	Ext Trigger OVL	3	8	Channel 4 OVL	2	4	Channel 3 OVL	1	2	Channel 2 OVL	0	1	Channel 1 OVL
Bit	Weight	Input																																	
10	1024	Ext Trigger Fault																																	
9	512	Channel 4 Fault																																	
8	256	Channel 3 Fault																																	
7	128	Channel 2 Fault																																	
6	64	Channel 1 Fault																																	
4	16	Ext Trigger OVL																																	
3	8	Channel 4 OVL																																	
2	4	Channel 3 OVL																																	
1	2	Channel 2 OVL																																	
0	1	Channel 1 OVL																																	
n/a	:OVLRegister? (see page 171)	<value> ::= integer in NR1 format. See OVLenable for <value>																																	
:PRINT [<options>] (see page 173)	n/a	<p><options> ::= [<print option>] [,...,<print option>]</p> <p><print option> ::= {COLOR GRAYscale PRINTER0 PRINTER1 BMP8bit BMP PNG NOFactors FACTors}</p> <p><print option> can be repeated up to 5 times.</p>																																	
:RUN (see page 174)	n/a	n/a																																	
n/a	:SERial (see page 175)	<return value> ::= unquoted string containing serial number																																	
:SINGle (see page 176)	n/a	n/a																																	
n/a	:STATus? <display> (see page 177)	<p>{0 1}</p> <p><display> ::= {CHANnel<n> DIGital<d> POD{1 2} BUS{1 2} FUNCTION MATH SBUS1 WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p>																																	
:STOP (see page 178)	n/a	n/a																																	

Table 49 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 179)	{0 1}
:VIEW <source> (see page 180)	n/a	<source> ::= {CHANnel<n> FUNCTION MATH SBUS1 WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCTION MATH SBUS1 WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

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

:ACTivity

N (see [page 794](#))

Command Syntax `:ACTivity`

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

Query Syntax `:ACTivity?`

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

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

Return Format

```
<edges>,<levels><NL>
<edges> ::= presence of edges (16-bit integer in NR1 format).
<levels> ::= logical highs or lows (16-bit integer in NR1 format).
bit 0 ::= DIGital 0
bit 15 ::= DIGital 15
```

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

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

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

See Also

- ["Introduction to Root \(:\) Commands"](#) on page 148
- [":POD<n>:THreshold"](#) on page 417
- [":DIGital<d>:THreshold"](#) on page 249

:AER (Arm Event Register)(see [page 794](#))**Query Syntax**`:AER?`

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

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

Return Format`<value><NL>``<value> ::= {0 | 1}; an integer in NR1 format.`**See Also**

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:OPEE \(Operation Status Enable Register\)](#)" on page 163
- "[:OPERRegister:CONDition \(Operation Status Condition Register\)](#)" on page 165
- "[:OPERRegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 167
- "[*STB \(Read Status Byte\)](#)" on page 140
- "[*SRE \(Service Request Enable\)](#)" on page 138

:AUToscale

 (see [page 794](#))

Command Syntax

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

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

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see "[:AUToscale:CHANnels](#)" on page 154) is set to DISPlayed channels. Then, the autoscale is performed.

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

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

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

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

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

See Also	<ul style="list-style-type: none">· "Introduction to Root (:) Commands" on page 148· "":AUToscale:CHANnels" on page 154· "":AUToscale:AMODE" on page 153
Example Code	<pre>' AUTOSCALE - This command evaluates all the input signals and sets ' the correct conditions to display all of the active signals. myScope.WriteString ":AUToscale" ' Same as pressing Auto Scale key.</pre> <p>See complete example programs at: Chapter 38, “Programming Examples,” starting on page 803</p>

:AUToscale:AMODE

N (see [page 794](#))

Command Syntax :AUToscale:AMODE <value>

<value> ::= {NORMAL | CURR}

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

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

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

Query Syntax :AUToscale:AMODE?

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

Return Format <value><NL>

<value> ::= {NORM | CURR}

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 148
 - "[":AUToscale"](#)" on page 151
 - "[":AUToscale:CHANnels"](#)" on page 154
 - "[":ACQuire:TYPE"](#)" on page 193
 - "[":ACQuire:MODE"](#)" on page 185

:AUToscale:CHANnels**N** (see [page 794](#))

Command Syntax `:AUToscale:CHANnels <value>`
`<value> ::= {ALL | DISPlayed}`

The :AUToscale:CHANnels command specifies which channels will be displayed on subsequent :AUToscales.

- When ALL is selected, all channels that meet the requirements of :AUToscale will be displayed.
- When DISPlayed is selected, only the channels that are turned on are autoscaled.

Use the :VIEW or :BLANK root commands to turn channels on or off.

Query Syntax `:AUToscale:CHANnels?`

The :AUToscale:CHANnels? query returns the autoscale channels setting.

Return Format `<value><NL>`
`<value> ::= {ALL | DISP}`

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[":AUToscale](#)" on page 151
- "[":AUToscale:AMODE](#)" on page 153
- "[":VIEW](#)" on page 180
- "[":BLANK](#)" on page 156

:AUToscale:FDEBug

N (see [page 794](#))

Command Syntax :AUToscale:FDEBug <on_off>

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

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

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

Channel coupling is maintained for easy viewing of oscillating signals.

Query Syntax :AUToscale:FDEBug?

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

Return Format <on_off><NL>

<on_off> ::= {1 | 0}

See Also • ["Introduction to Root \(:\) Commands" on page 148](#)
• [":AUToscale" on page 151](#)

:BLANK

N (see [page 794](#))

Command Syntax

```
:BLANK [<source>]
<source> ::= {CHANnel<n> | FUNCTION | MATH | SBUS1 | WMEMORY<r>}
for the DSO models

<source> ::= {CHANnel<n> | DIGItal<d> | POD{1 | 2}
| BUS{1 | 2} | FUNCTION | MATH | SBUS1 | WMEMORY<r>}
for the MSO models

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

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

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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

NOTE

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

NOTE

MATH is an alias for FUNCTION.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:DISPlay:CLEar](#)" on page 257
- "[:CHANnel<n>:DISPlay](#)" on page 221
- "[:DIGItal<d>:DISPlay](#)" on page 245
- "[:FUNCTION:DISPlay](#)" on page 282
- "[:POD<n>:DISPlay](#)" on page 415
- "[:WMEMORY<r>:DISPlay](#)" on page 698
- "[:STATUs](#)" on page 177
- "[:VIEW](#)" on page 180

Example Code

- "[Example Code](#)" on page 180

:DIGItize

C (see page 794)

Command Syntax

```
:DIGItize [<source>[, . . . , <source>]]
<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS1}
for the DSO models

<source> ::= {CHANnel<n> | DIGItal<d> | POD{1 | 2}
| BUS{1 | 2} | FUNCtion | MATH | SBUS1}
for the MSO models

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

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

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

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

NOTE

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

NOTE

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

NOTE

MATH is an alias for FUNCtion.

See Also

- ["Introduction to Root \(:\) Commands"](#) on page 148
- [":RUN"](#) on page 174
- [":SINGle"](#) on page 176
- [":STOP"](#) on page 178
- [":TIMEbase:MODE"](#) on page 581
- [Chapter 7](#), “:ACQuire Commands,” starting on page 181
- [Chapter 30](#), “:WAVeform Commands,” starting on page 629

Example Code

```
' Capture an acquisition using :DIGItize.
' -----
myScope.WriteString ":DIGItize CHANnel1"
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:MTEenable (Mask Test Event Enable Register)

N (see [page 794](#))

Command Syntax

```
:MTEenable <mask>
<mask> ::= 16-bit integer
```

The :MTEenable command sets a mask in the Mask Test Event Enable register. Set any of the following bits to "1" to enable bit 9 in the Operation Status Condition Register and potentially cause an SRQ (Service Request interrupt) to be generated.

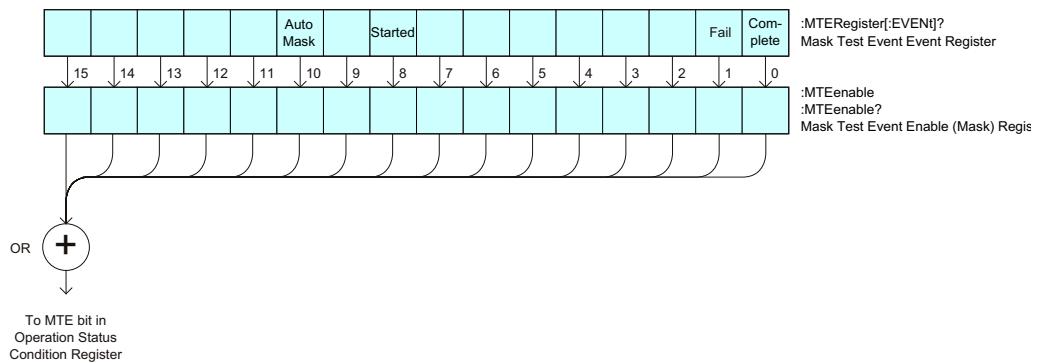


Table 50 Mask Test Event Enable Register (MTEenable)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	---	---	(Not used.)
8	Started	Mask Testing Started	Mask testing started.
7-2	---	---	(Not used.)
1	Fail	Mask Test Fail	Mask test failed.
0	Complete	Mask Test Complete	Mask test is complete.

Query Syntax

```
:MTEenable?
```

The :MTEenable? query returns the current value contained in the Mask Test Event Enable register as an integer number.

Return Format

```
<value><NL>
<value> ::= integer in NR1 format.
```

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 148
 - "[:AER \(Arm Event Register\)](#)" on page 150
 - "[:CHANnel<n>:PROTection](#)" on page 231
 - "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 167
 - "[:OVLenable \(Overload Event Enable Register\)](#)" on page 169
 - "[:OVLRegister \(Overload Event Register\)](#)" on page 171
 - "[!*STB \(Read Status Byte\)](#)" on page 140
 - "[!*SRE \(Service Request Enable\)](#)" on page 138

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

N (see [page 794](#))

Query Syntax :MTERegister [:EVENT] ?

The :MTERegister[:EVENT]? query returns the integer value contained in the Mask Test Event Event Register and clears the register.

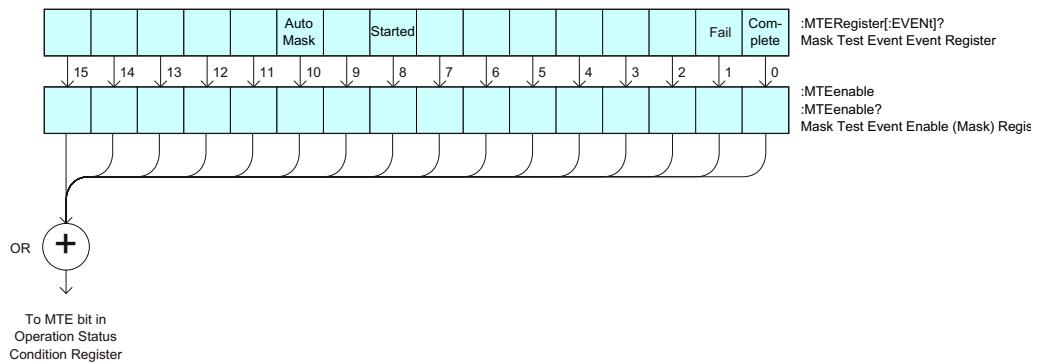


Table 51 Mask Test Event Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	---	---	(Not used.)
8	Started	Mask Testing Started	Mask testing started.
7-2	---	---	(Not used.)
1	Fail	Mask Test Fail	The mask test failed.
0	Complete	Mask Test Complete	The mask test is complete.

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

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 148
 - "[":CHANnel<n>:PROtection](#)" on page 231
 - "[":OPEE \(Operation Status Enable Register\)](#)" on page 163
 - "[":OPERregister:CONDition \(Operation Status Condition Register\)"](#) on page 165
 - "[":OVLenable \(Overload Event Enable Register\)"](#) on page 169

- [":OVLRegister \(Overload Event Register\)" on page 171](#)
- ["*STB \(Read Status Byte\)" on page 140](#)
- ["*SRE \(Service Request Enable\)" on page 138](#)

:OPEE (Operation Status Enable Register)

C (see page 794)

Command Syntax :OPEE <mask>

<mask> ::= 15-bit integer

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

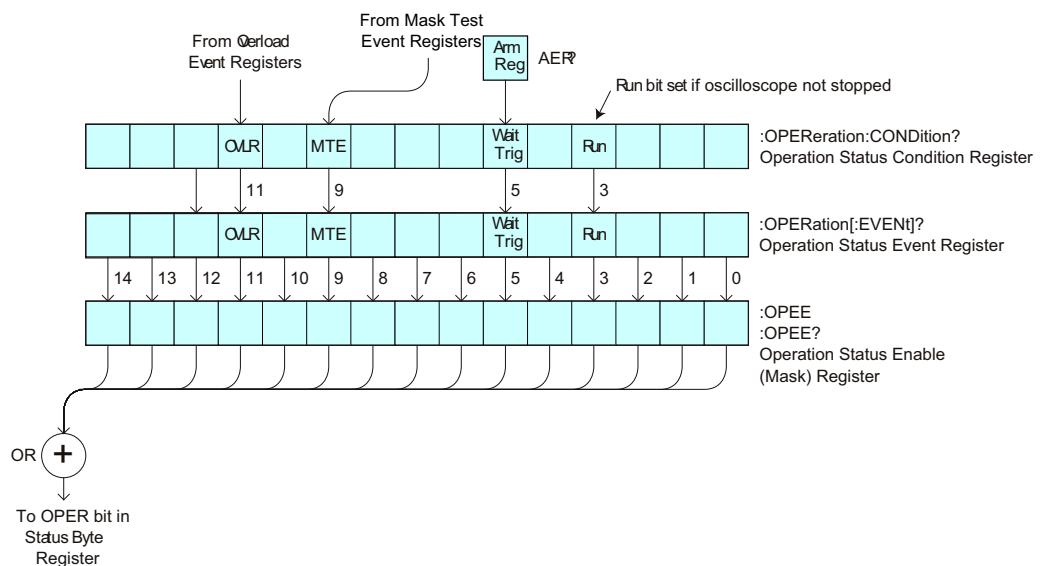


Table 52 Operation Status Enable Register (OPEE)

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

Query Syntax :OPEE?

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:AER \(Arm Event Register\)](#)" on page 150
- "[:CHANnel<n>:PROTection](#)" on page 231
- "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 167
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 169
- "[:OVLRegister \(Overload Event Register\)](#)" on page 171
- "[*STB \(Read Status Byte\)](#)" on page 140
- "[*SRE \(Service Request Enable\)](#)" on page 138

:OPERegister:CONDition (Operation Status Condition Register)

C (see page 794)

Query Syntax :OPERegister:CONDition?

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

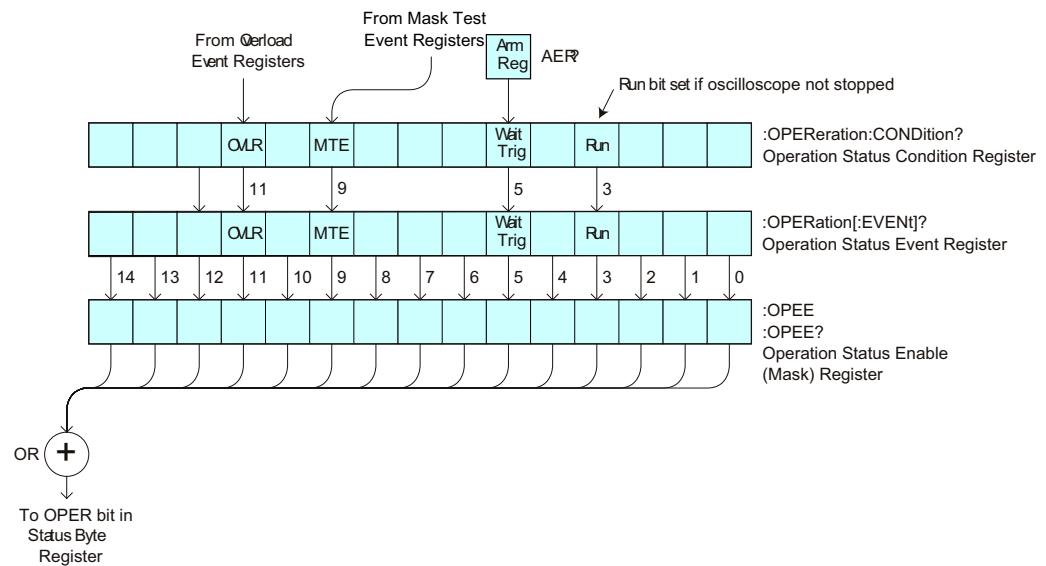


Table 53 Operation Status Condition Register

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

- See Also
- "[Introduction to Root \(:\) Commands](#)" on page 148
 - "[":CHANnel<n>:PROTection](#)" on page 231
 - "[":OPEE \(Operation Status Enable Register\)](#)" on page 163
 - "[":OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 167
 - "[":OVLenable \(Overload Event Enable Register\)](#)" on page 169
 - "[":OVLRegister \(Overload Event Register\)](#)" on page 171
 - "[":*STB \(Read Status Byte\)](#)" on page 140
 - "[":*SRE \(Service Request Enable\)](#)" on page 138
 - "[":MTERegister\[:EVENT\] \(Mask Test Event Event Register\)](#)" on page 161
 - "[":MTEenable \(Mask Test Event Enable Register\)](#)" on page 159

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

C (see page 794)

Query Syntax :OPERegister [:EVENT] ?

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

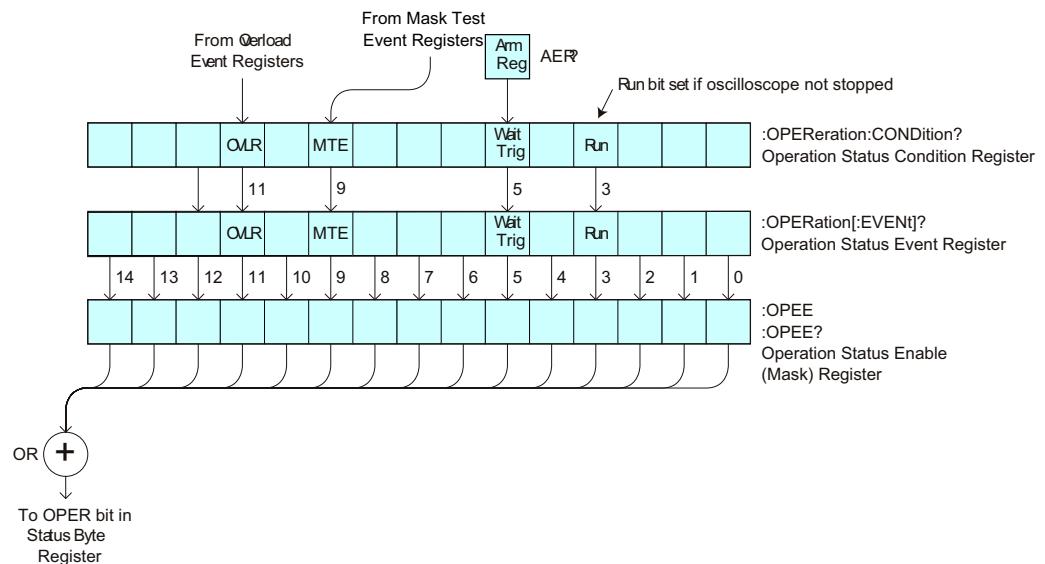


Table 54 Operation Status Event Register

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:CHANnel<n>:PROTection](#)" on page 231
- "[:OPEE \(Operation Status Enable Register\)](#)" on page 163
- "[:OPERegister:CONDition \(Operation Status Condition Register\)](#)" on page 165
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 169
- "[:OVLRegister \(Overload Event Register\)](#)" on page 171
- "[:*STB \(Read Status Byte\)](#)" on page 140
- "[:*SRE \(Service Request Enable\)](#)" on page 138
- "[:MTERegister\[:EVENT\] \(Mask Test Event Event Register\)](#)" on page 161
- "[:MTEenable \(Mask Test Event Enable Register\)](#)" on page 159

:OVLenable (Overload Event Enable Register)

C (see [page 794](#))

Command Syntax :OVLenable <enable_mask>

<enable_mask> ::= 16-bit integer

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

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

NOTE

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

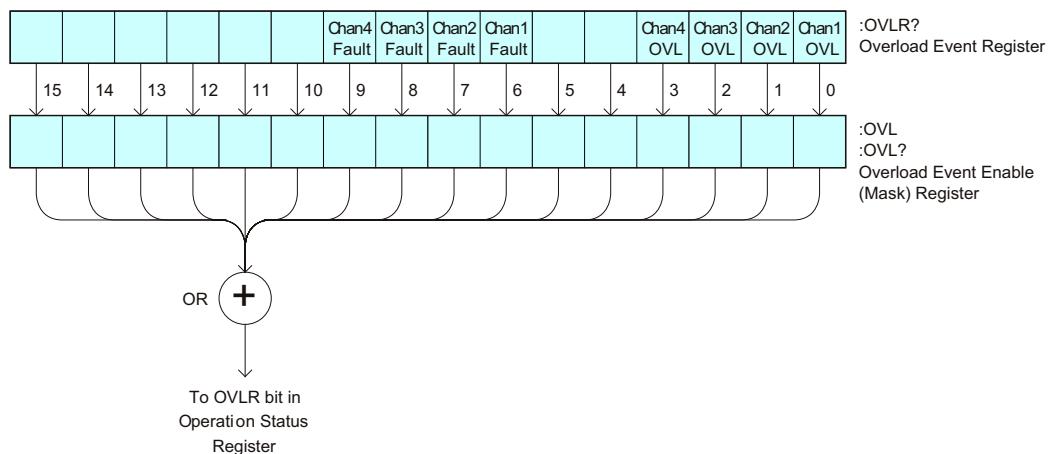


Table 55 Overload Event Enable Register (OVL)

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

Table 55 Overload Event Enable Register (OVL) (continued)

Bit	Description	When Set (1 = High = True), Enables:
3	Channel 4 OVL	Event when overload occurs on Channel 4 input.
2	Channel 3 OVL	Event when overload occurs on Channel 3 input.
1	Channel 2 OVL	Event when overload occurs on Channel 2 input.
0	Channel 1 OVL	Event when overload occurs on Channel 1 input.

Query Syntax :OVLenable?

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

Return Format <enable_mask><NL>

<enable_mask> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[":CHANnel<n>:PROTection](#)" on page 231
- "[":OPEE \(Operation Status Enable Register\)](#)" on page 163
- "[":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 165
- "[":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 167
- "[":OVLRegister \(Overload Event Register\)"](#) on page 171
- "[":*STB \(Read Status Byte\)"](#) on page 140
- "[":*SRE \(Service Request Enable\)"](#) on page 138

:OVLRegister (Overload Event Register)

C (see page 794)

Query Syntax :OVLRegister?

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

NOTE

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

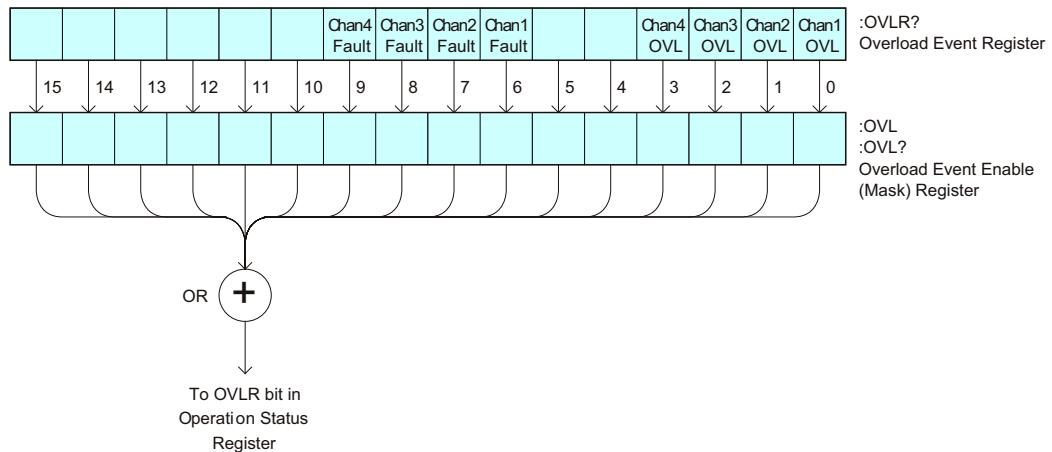


Table 56 Overload Event Register (OCLR)

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

Table 56 Overload Event Register (OVLR) (continued)

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:CHANnel<n>:PROTection](#)" on page 231
- "[:OPEE \(Operation Status Enable Register\)](#)" on page 163
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 169
- "[*STB \(Read Status Byte\)](#)" on page 140
- "[*SRE \(Service Request Enable\)](#)" on page 138

:PRINT**C** (see page 794)

Command Syntax	<pre>:PRINT [<options>] <options> ::= [<print option>] [,...<print option>] <print option> ::= {COLOR GRAYscale PRINTER0 PRINTER1 BMP8bit BMP PNG NOFactors FACTors}</pre>
	<p>The <print option> parameter may be repeated up to 5 times.</p> <p>The PRINT command formats the output according to the currently selected format (device). If an option is not specified, the value selected in the Print Config menu is used.</p>
See Also	<ul style="list-style-type: none">· "Introduction to Root () Commands" on page 148· "Introduction to :HARDcopy Commands" on page 298· ":HARDcopy:FACTors" on page 301· ":HARDcopy:GRAYscale" on page 724· ":DISPLAY:DATA" on page 258

:RUN (see [page 794](#))**Command Syntax**`:RUN`

The :RUN command starts repetitive acquisitions. This is the same as pressing the Run key on the front panel.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[":SINGle"](#) on page 176
- "[":STOP"](#) on page 178

Example Code

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

See complete example programs at: [Chapter 38, “Programming Examples,”](#) starting on page 803

:SERial

N (see [page 794](#))

Query Syntax :SERial?

The :SERial? query returns the serial number of the instrument.

Return Format: Unquoted string<NL>

See Also • ["Introduction to Root \(:\) Commands" on page 148](#)

:SINGle

 (see [page 794](#))

Command Syntax

`:SINGle`

The :SINGle command causes the instrument to acquire a single trigger of data. This is the same as pressing the Single key on the front panel.

See Also

- ["Introduction to Root \(:\) Commands" on page 148](#)
- [":RUN" on page 174](#)
- [":STOP" on page 178](#)

:STATus

N (see [page 794](#))

Query Syntax

```
:STATus? <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS1 | WMEMory<r>}
for the DSO models

<source> ::= {CHANnel<n> | DIGItal<d> | POD{1 | 2}
| BUS{1 | 2} | FUNCtion | MATH | SBUS1 | WMEMory<r>}
for the MSO models

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

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

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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

NOTE

MATH is an alias for FUNCtion.

Return Format

```
<value><NL>
```

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

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:BLANK](#)" on page 156
- "[:CHANnel<n>:DISPlay](#)" on page 221
- "[:DIGItal<d>:DISPlay](#)" on page 245
- "[:FUNCtion:DISPlay](#)" on page 282
- "[:POD<n>:DISPlay](#)" on page 415
- "[:WMEMory<r>:DISPlay](#)" on page 698
- "[:VIEW](#)" on page 180

:STOP

 (see [page 794](#))

Command Syntax :STOP

The :STOP command stops the acquisition. This is the same as pressing the Stop key on the front panel.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[":RUN"](#) on page 174
- "[":SINGLe"](#) on page 176

Example Code

- "[Example Code](#)" on page 174

:TER (Trigger Event Register)

 (see [page 794](#))

Query Syntax :TER?

The :TER? query reads the Trigger Event Register. After the Trigger Event Register is read, it is cleared. A one indicates a trigger has occurred. A zero indicates a trigger has not occurred.

The Trigger Event Register is summarized in the TRG bit of the Status Byte Register (STB). A Service Request (SRQ) can be generated when the TRG bit of the Status Byte transitions, and the TRG bit is set in the Service Request Enable register. The Trigger Event Register must be cleared each time you want a new service request to be generated.

Return Format <value><NL>

<value> ::= {1 | 0}; a 16-bit integer in NR1 format.

See Also

- ["Introduction to Root \(:\) Commands" on page 148](#)
- ["*SRE \(Service Request Enable\)" on page 138](#)
- ["*STB \(Read Status Byte\)" on page 140](#)

:VIEW

N (see [page 794](#))

Command Syntax

```
:VIEW <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS1 | WMEMORY<r>}
for DSO models

<source> ::= {CHANnel<n> | DIGItal<d> | POD{1 | 2}
| BUS{1 | 2} | FUNCtion | MATH | SBUS1 | WMEMORY<r>}
for MSO models

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

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

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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

NOTE

MATH is an alias for FUNCtion.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 148
- "[:BLANK](#)" on page 156
- "[:CHANnel<n>:DISPlay](#)" on page 221
- "[:DIGItal<d>:DISPlay](#)" on page 245
- "[:FUNCtion:DISPlay](#)" on page 282
- "[:POD<n>:DISPlay](#)" on page 415
- "[:WMEMORY<r>:DISPlay](#)" on page 698
- "[:STATus](#)" on page 177

Example Code

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

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

7 :ACQuire Commands

Set the parameters for acquiring and storing data. See "[Introduction to :ACQuire Commands](#)" on page 181.

Table 57 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:COMPLETE <complete> (see page 183)	:ACQuire:COMPLETE? (see page 183)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNT <count> (see page 184)	:ACQuire:COUNT? (see page 184)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:MODE <mode> (see page 185)	:ACQuire:MODE? (see page 185)	<mode> ::= {RTIMe SEGmented}
n/a	:ACQuire:POINTS? (see page 186)	<# points> ::= an integer in NR1 format
:ACQuire:SEGMENTed:ANALyze (see page 187)	n/a	n/a (with Option SGM)
:ACQuire:SEGMENTed:COUNT <count> (see page 188)	:ACQuire:SEGMENTed:COUNT? (see page 188)	<count> ::= an integer from 2 to 25 in NR1 format (with Option SGM)
:ACQuire:SEGMENTed:INDEX <index> (see page 189)	:ACQuire:SEGMENTed:INDEX? (see page 189)	<index> ::= an integer from 1 to 25 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATE? (see page 192)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 193)	:ACQuire:TYPE? (see page 193)	<type> ::= {NORMAL AVERage HRESolution PEAK}

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

Normal

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

Averaging

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

High-Resolution

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

Peak Detect

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

Reporting the Setup

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

Return Format

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

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

:ACQuire:COMplete

C (see [page 794](#))

Command Syntax :ACQuire:COMplete <complete>

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

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

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

Query Syntax :ACQuire:COMplete?

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

Return Format <completion_criteria><NL>

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

- See Also**
- "[Introduction to :ACQuire Commands](#)" on page 181
 - "[":ACQuire:TYPE](#)" on page 193
 - "[":DIGITIZE](#)" on page 157
 - "[":WAVEFORM:POINTS](#)" on page 642

Example Code

```
' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
' an acquisition. The parameter determines the percentage of time
' buckets needed to be "full" before an acquisition is considered
' to be complete.
myScope.WriteString ":ACQuire:COMplete 100"
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:ACQuire:COUNt

C (see [page 794](#))

Command Syntax `:ACQuire:COUNt <count>`

`<count> ::= integer in NR1 format`

In averaging mode, the :ACQuire:COUNt command specifies the number of values to be averaged for each time bucket before the acquisition is considered to be complete for that time bucket. When :ACQuire:TYPE is set to AVERage, the count can be set to any value from 2 to 65536.

NOTE

The :ACQuire:COUNt 1 command has been deprecated. The AVERage acquisition type with a count of 1 is functionally equivalent to the HRESolution acquisition type; however, you should select the high-resolution acquisition mode with the :ACQuire:TYPE HRESolution command instead.

Query Syntax `:ACQuire:COUNT?`

The :ACQuire:COUNT? query returns the currently selected count value for averaging mode.

Return Format `<count_argument><NL>`

`<count_argument> ::= an integer from 2 to 65536 in NR1 format`

See Also

- "[Introduction to :ACQuire Commands](#)" on page 181
- "[":ACQuire:TYPE"](#) on page 193
- "[":DIGitize"](#) on page 157
- "[":WAVeform:COUNt"](#) on page 638

:ACQuire:MODE

 (see [page 794](#))

Command Syntax `:ACQuire:MODE <mode>`

`<mode> ::= {RTIMe | SEGmented}`

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

- The :ACQuire:MODE RTIMe command sets the oscilloscope in real time mode.

NOTE

The obsolete command ACQuire:TYPE:REALtime is functionally equivalent to sending ACQuire:MODE RTIMe; TYPE NORMAl.

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

Query Syntax `:ACQuire:MODE?`

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

Return Format `<mode_argument><NL>`

`<mode_argument> ::= {RTIM | SEGM}`

See Also · ["Introduction to :ACQuire Commands"](#) on page 181
 · [":ACQuire:TYPE"](#) on page 193

:ACQuire:POINts (see [page 794](#))**Query Syntax** `:ACQuire:POINts?`

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

Return Format `<points_argument><NL>`

`<points_argument>` ::= an integer in NR1 format

See Also

- ["Introduction to :ACQuire Commands"](#) on page 181

- [":DIGitize"](#) on page 157

- [":WAVeform:POINts"](#) on page 642

:ACQuire:SEGmented:ANALyze

N (see [page 794](#))

Command Syntax :ACQuire:SEGmented:ANALyze

NOTE

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

This command calculates measurement statistics and/or infinite persistence over all segments that have been acquired. It corresponds to the front panel **Analyze Segments** softkey which appears in both the Measurement Statistics and Segmented Memory Menus.

In order to use this command, the oscilloscope must be stopped and in segmented acquisition mode, with either quick measurements or infinite persistence on.

See Also

- [":ACQuire:MODE"](#) on page 185
- [":ACQuire:SEGmented:COUNt"](#) on page 188
- ["Introduction to :ACQuire Commands"](#) on page 181

:ACQuire:SEGmented:COUNt

N (see [page 794](#))

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

NOTE

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

The :ACQuire:SEGmented:COUNt command sets the number of memory segments to acquire.

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

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

Query Syntax `:ACQuire:SEGmented:COUNt?`

The :ACQuire:SEGmented:COUNt? query returns the current count setting.

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

See Also

- "[:ACQuire:MODE](#)" on page 185
- "[:DIGITIZE](#)" on page 157
- "[:SINGLE](#)" on page 176
- "[:RUN](#)" on page 174
- "[:WAVeform:SEGmented:COUNt](#)" on page 649
- "[:ACQuire:SEGmented:ANALyze](#)" on page 187
- "[Introduction to :ACQuire Commands](#)" on page 181

Example Code

- "["Example Code"](#) on page 189

:ACQuire:SEGmented:INDEX

N (see [page 794](#))

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

NOTE

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

The :ACQuire:SEGmented:INDEX command sets the index into the memory segments that have been acquired.

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGmented:COUNt command, and data is acquired using the :DIGitize, :SINGle, or :RUN commands. The number of memory segments that have been acquired is returned by the :WAVeform:SEGmented:COUNt? query. The time tag of the currently indexed memory segment is returned by the :WAVeform:SEGmented:TTAG? query.

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

Query Syntax :ACQuire:SEGmented:INDEX?

The :ACQuire:SEGmented:INDEX? query returns the current segmented memory index setting.

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

- See Also**
- "[:ACQuire:MODE](#)" on page 185
 - "[:ACQuire:SEGmented:COUNt](#)" on page 188
 - "[:DIGitize](#)" on page 157
 - "[:SINGle](#)" on page 176
 - "[:RUN](#)" on page 174
 - "[:WAVeform:SEGmented:COUNt](#)" on page 649
 - "[:WAVeform:SEGmented:TTAG](#)" on page 650
 - "[:ACQuire:SEGmented:ANALyze](#)" on page 187
 - "[Introduction to :ACQuire Commands](#)" on page 181

Example Code

```
' Segmented memory commands example.  

' -----
```

```

Option Explicit

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

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

Sub Main()

On Error GoTo VisaComError

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

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

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

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

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

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

Dim lngSegments As Long
lngSegments = varQueryResult

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

```

```
For lngI = lngSegments To 1 Step -1

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

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

    Next lngI

    Exit Sub

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

End Sub
```

:ACQuire:SRATe

N (see [page 794](#))

Query Syntax :ACQuire:SRATe?

The :ACQuire:SRATe? query returns the current oscilloscope acquisition sample rate. The sample rate is not directly controllable.

Return Format <sample_rate><NL>

<sample_rate> ::= sample rate in NR3 format

See Also • ["Introduction to :ACQuire Commands"](#) on page 181
• [":ACQuire:POINts"](#) on page 186

:ACQuire:TYPE

C (see [page 794](#))

Command Syntax :ACQuire:TYPE <type>

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

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

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

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

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

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

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

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

NOTE

The obsolete command ACQuire:TYPE:REALtime is functionally equivalent to sending ACQuire:MODE RTIME; TYPE NORMAL.

Query Syntax :ACQuire:TYPE?

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

Return Format <acq_type><NL>

<acq_type> ::= {NORM | AVER | HRES | PEAK}

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

- "[:ACQuire:COUNT](#)" on page 184
- "[:ACQuire:MODE](#)" on page 185
- "[:DIGitize](#)" on page 157
- "[:WAVeform:FORMat](#)" on page 641
- "[:WAVeform:TYPE](#)" on page 656
- "[:WAVeform:PREamble](#)" on page 646

Example Code

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

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

8 :BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See "[Introduction to :BUS<n> Commands](#)" on page 196.

Table 58 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 OFF} {1 ON}} (see page 197)	:BUS<n>:BIT<m>? (see page 197)	{0 1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 OFF} {1 ON}} (see page 198)	:BUS<n>:BITS? (see page 198)	<channel_list>, {0 1} <channel_list> ::= (@<m>,<m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:CLEar (see page 200)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 201)	:BUS<n>:DISPlay? (see page 201)	{0 1} <n> ::= 1 or 2; an integer in NR1 format

Table 58 :BUS<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see page 202)	:BUS<n>:LABel? (see page 202)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 203)	:BUS<n>:MASK? (see page 203)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

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

<n> ::= {1 | 2}

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

NOTE

These commands are only valid for the MSO models.

Reporting the Setup

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

Return Format

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

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

:BUS<n>:BIT<m>

N (see [page 794](#))

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

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

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

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

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

NOTE

This command is only valid for the MSO models.

Query Syntax

:BUS<n>:BIT<m>?

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

Return Format

<display><NL>

<display> ::= {0 | 1}

See Also

- "[Introduction to :BUS<n> Commands](#)" on page 196
- "[":BUS<n>:BITS](#)" on page 198
- "[":BUS<n>:CLEar](#)" on page 200
- "[":BUS<n>:DISPlay](#)" on page 201
- "[":BUS<n>:LABEL](#)" on page 202
- "[":BUS<n>:MASK](#)" on page 203

Example Code

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

:BUS<n>:BITS

N (see [page 794](#))

Command Syntax	<code>:BUS<n>:BITS <channel_list>, <display></code>
	<code><channel_list> ::= (@<m>,<m>:<m>, ...)</code> where commas separate bits and colons define bit ranges.
	<code><m> ::= An integer, 0,...,7, defines a digital channel affected by the command.</code>
	<code><display> ::= {{1 ON} {0 OFF}}</code>
	<code><n> ::= An integer, 1 or 2, is attached as a suffix to BUS and defines the bus that ?s affected by the command.</code>

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

NOTE

This command is only valid for the MSO models.

Query Syntax	<code>:BUS<n>:BITS?</code>
	The :BUS<n>:BITS? query returns the definition for the specified bus.
Return Format	<code><channel_list>, <display><NL></code>
	<code><channel_list> ::= (@<m>,<m>:<m>, ...)</code> where commas separate bits and colons define bit ranges.
	<code><display> ::= {0 1}</code>
See Also	<ul style="list-style-type: none"> • "Introduction to :BUS<n> Commands" on page 196 • "":BUS<n>:BIT<m>" on page 197 • "":BUS<n>:CLEar" on page 200 • "":BUS<n>:DISPlay" on page 201 • "":BUS<n>:LABEL" on page 202 • "":BUS<n>:MASK" on page 203
Example Code	<pre>' Include digital channels 1, 2, 4, 5, 6, and 7 in bus 1: myScope.WriteString ":BUS1:BITS (@1,2,4:7), ON" ' Include digital channels 1, 5, and 7 in bus 1: myScope.WriteString ":BUS1:BITS (@1,5,7), ON" ' Include digital channels 1 through 7 in bus 1: myScope.WriteString ":BUS1:BITS (@1:7), ON"</pre>

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

:BUS<n>:CLEar

N (see [page 794](#))

Command Syntax :BUS<n>:CLEar

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

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

NOTE

This command is only valid for the MSO models.

See Also

- "[Introduction to :BUS<n> Commands](#)" on page 196
- "[":BUS<n>:BIT<m>"](#) on page 197
- "[":BUS<n>:BITS"](#) on page 198
- "[":BUS<n>:DISPlay"](#) on page 201
- "[":BUS<n>:LABEL"](#) on page 202
- "[":BUS<n>:MASK"](#) on page 203

:BUS<n>:DISPlay

N (see [page 794](#))

Command Syntax :BUS<n>:DISPlay <value>

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

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

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

NOTE

This command is only valid for the MSO models.

Query Syntax :BUS<n>:DISPlay?

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

Return Format <value><NL>

<value> ::= {0 | 1}

See Also • "[Introduction to :BUS<n> Commands](#)" on page 196

- "[:BUS<n>:BIT<m>](#)" on page 197
- "[:BUS<n>:BITS](#)" on page 198
- "[:BUS<n>:CLEar](#)" on page 200
- "[:BUS<n>:LABEL](#)" on page 202
- "[:BUS<n>:MASK](#)" on page 203

:BUS<n>:LABEL

N (see [page 794](#))

Command Syntax :BUS<n>:LABEL <quoted_string>

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

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

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

NOTE

This command is only valid for the MSO models.

NOTE

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

Query Syntax

:BUS<n>:LABEL?

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

Return Format

<quoted_string><NL>

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

See Also

- "[Introduction to :BUS<n> Commands](#)" on page 196
- "[":BUS<n>:BIT<m>"](#) on page 197
- "[":BUS<n>:BITS"](#) on page 198
- "[":BUS<n>:CLEAR"](#) on page 200
- "[":BUS<n>:DISPLAY"](#) on page 201
- "[":BUS<n>:MASK"](#) on page 203
- "[":CHANnel<n>:LABEL"](#) on page 224
- "[":DISPLAY:LABELList"](#) on page 261
- "[":DIGital<d>:LABEL"](#) on page 246

Example Code

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

:BUS<n>:MASK

N (see [page 794](#))

Command Syntax

```
:BUS<n>:MASK <mask>
<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string>
<nondecimal> ::= #Hnn...n where n ::= {0,...,9 | A,...,F} for hexadecimal
<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F} for hexadecimal
<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.
```

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

NOTE

This command is only valid for the MSO models.

Query Syntax

:BUS<n>:MASK?

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

Return Format

<mask><NL> in decimal format

See Also

- "[Introduction to :BUS<n> Commands](#)" on page 196
- "[":BUS<n>:BIT<m>"](#) on page 197
- "[":BUS<n>:BITS"](#) on page 198
- "[":BUS<n>:CLEar"](#) on page 200
- "[":BUS<n>:DISPlay"](#) on page 201
- "[":BUS<n>:LABEL"](#) on page 202

9 :CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "[Introduction to :CALibrate Commands](#)" on page 206.

Table 59 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 207)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABEL <string> (see page 208)	:CALibrate:LABEL? (see page 208)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 209)	:CALibrate:OUTPut? (see page 209)	<signal> ::= {TRIGgers MASK WAVEgen}
n/a	:CALibrate:PROTected? (see page 210)	{"PROTected" "UNPROtected"}
:CALibrate:START (see page 211)	n/a	n/a
n/a	:CALibrate:STATUS? (see page 212)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPeratur e? (see page 213)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 214)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

- Introduction to :CALibrate Commands**
- The CALibrate subsystem provides utility commands for:
- Determining the state of the calibration factor protection switch (CAL PROTECT).
 - Saving and querying the calibration label string.
 - Reporting the calibration time and date.
 - Reporting changes in the temperature since the last calibration.
 - Starting the user calibration procedure.

:CALibrate:DATE

N (see [page 794](#))

Query Syntax :CALibrate:DATE?

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

Return Format <date><NL>

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

See Also • ["Introduction to :CALibrate Commands"](#) on page 206

:CALibrate:LABEL

N (see [page 794](#))

Command Syntax :CALibrate:LABEL <string>

<string> ::= quoted ASCII string of up to 32 characters in length,
not including the quotes

The CALibrate:LABEL command saves a string that is up to 32 characters in length into the instrument's non-volatile memory. The string may be used to record calibration dates or other information as needed.

Query Syntax :CALibrate:LABEL?

The :CALibrate:LABEL? query returns the contents of the calibration label string.

Return Format <string><NL>

<string> ::= unquoted ASCII string of up to 32 characters in length

See Also • ["Introduction to :CALibrate Commands"](#) on page 206

:CALibrate:OUTPut

N (see [page 794](#))

Command Syntax

```
:CALibrate:OUTPut <signal>
<signal> ::= {TRIGgers | MASK | WAVEgen}
```

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

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

Waveform Type	Sync Signal Characteristics
SINusoid, SQuare, RAMP, PULSe	The Sync signal is a TTL positive pulse that occurs when the waveform rises above zero volts (or the DC offset value).
DC, NOISe	N/A

Query Syntax

```
:CALibrate:OUTPut?
```

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

Return Format

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

See Also

- "[Introduction to :CALibrate Commands](#)" on page 206
- "[":WGEN:FUNCTION](#)" on page 669

:CALibrate:PROTected

N (see [page 794](#))

Query Syntax :CALibrate:PROTected?

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

Return Format <switch><NL>

<switch> ::= { "PROTected" | "UNPROtected" }

See Also • ["Introduction to :CALibrate Commands"](#) on page 206

:CALibrate:STARt

N (see [page 794](#))

Command Syntax :CALibrate:STARt

The CALibrate:STARt command starts the user calibration procedure.

NOTE

Before starting the user calibration procedure, you must set the rear panel CALIBRATION switch to UNPROTECTED, and you must connect BNC cables from the TRIG OUT connector to the analog channel inputs. See the *User's Guide* for details.

See Also

- "[Introduction to :CALibrate Commands](#)" on page 206
- "[":CALibrate:PROTected](#)" on page 210

:CALibrate:STATus

N (see [page 794](#))

Query Syntax :CALibrate:STATus?

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

Return Format <return value><NL>

<return value> ::= <status_code>,<status_string>

<status_code> ::= an integer status code

<status_string> ::= an ASCII status string

See Also • ["Introduction to :CALibrate Commands"](#) on page 206

:CALibrate:TEMPerature

N (see [page 794](#))

Query Syntax :CALibrate:TEMPerature?

The :CALibrate:TEMPerature? query returns the change in temperature since the last user calibration procedure.

Return Format <return value><NL>

<return value> ::= degrees C delta since last cal in NR3 format

See Also · "Introduction to :CALibrate Commands" on page 206

:CALibrate:TIME

N (see [page 794](#))

Query Syntax :CALibrate:TIME?

The :CALibrate:TIME? query returns the time of the last calibration.

Return Format <date><NL>

<date> ::= hour,minutes,seconds in NR1 format

See Also • ["Introduction to :CALibrate Commands"](#) on page 206

10 :CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "[Introduction to :CHANnel<n> Commands](#)" on page 217.

Table 60 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BANDwidth <limit> (see page 218)	:CHANnel<n>:BANDwidth ? [MAXimum] (see page 218)	<limit> ::= 25E6 in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:BWLimit { {0 OFF} {1 ON} } (see page 219)	:CHANnel<n>:BWLimit? (see page 219)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 220)	:CHANnel<n>:COUPling? (see page 220)	<coupling> ::= {AC DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay { {0 OFF} {1 ON} } (see page 221)	:CHANnel<n>:DISPlay? (see page 221)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 222)	:CHANnel<n>:IMPedance ? (see page 222)	<impedance> ::= ONEMeg <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert { {0 OFF} {1 ON} } (see page 223)	:CHANnel<n>:INVert? (see page 223)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABEL <string> (see page 224)	:CHANnel<n>:LABEL? (see page 224)	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 225)	:CHANnel<n>:OFFSet? (see page 225)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4; in NR1 format

Table 60 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe <attenuation> (see page 226)	:CHANnel<n>:PROBe? (see page 226)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
:CHANnel<n>:PROBe:HEA D [:TYPE] <head_param> (see page 227)	:CHANnel<n>:PROBe:HEA D [:TYPE]? (see page 227)	<head_param> ::= {SEND0 SEND6 SEND12 SEND20 DIFF0 DIFF6 DIFF12 DIFF20 NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 228)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKE W <skew_value> (see page 229)	:CHANnel<n>:PROBe:SKE W? (see page 229)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STY Pe <signal type> (see page 230)	:CHANnel<n>:PROBe:STY Pe? (see page 230)	<signal type> ::= {DIFFerential SINGLE} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 231)	:CHANnel<n>:PROTection? (see page 231)	NORM <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGE <range>[suffix] (see page 232)	:CHANnel<n>:RANGE? (see page 232)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 233)	:CHANnel<n>:SCALe? (see page 233)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITS <units> (see page 234)	:CHANnel<n>:UNITS? (see page 234)	<units> ::= {VOLT AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier { {0 OFF} {1 ON} } (see page 235)	:CHANnel<n>:VERNier? (see page 235)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format

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

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

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

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

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

NOTE

The obsolete CHANnel subsystem is supported.

Reporting the Setup

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

Return Format

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

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

:CHANnel<n>:BANDwidth

N (see [page 794](#))

Command Syntax `:CHANnel<n>:BANDwidth <limit>`

`<limit> ::= 25E6 in NR3 format`

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

The :CHANnel<n>:BANDwidth command sets the bandwidth limit value and turns on bandwidth limiting (see the :CHANnel<n>:BWLimit command).

For waveforms with frequencies below the bandwidth limit, turning the bandwidth limit on removes unwanted high frequency noise from the waveform.

Bandwidth limit also limits the trigger signal path of the channel.

While you can request any limit; the oscilloscope will choose the only bandwidth limit available, 25 MHz.

Query Syntax `:CHANnel<n>:BANDwidth? [MAXimum]`

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

If the bandwidth limit is off, the query returns the full bandwidth of the oscilloscope.

When the MAXimum parameter is used, the oscilloscope's maximum possible bandwidth is returned.

Return Format `<limit><NL>`

`<limit> ::= 25E6 or full bandwidth in NR3 format`

See Also · [":CHANnel<n>:BWLimit" on page 219](#)

:CHANnel<n>:BWLimit**C** (see [page 794](#))**Command Syntax** `:CHANnel<n>:BWLimit <bwlimit>` `<bwlimit> ::= {{1 | ON} | {0 | OFF}}` `<n> ::= 1 to (# analog channels) in NR1 format`

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

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

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

Return Format `<bwlimit><NL>` `<bwlimit> ::= {1 | 0}`**See Also** • "Introduction to :CHANnel<n> Commands" on page 217

:CHANnel<n>:COUpling

 (see [page 794](#))

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

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

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

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

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

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

Return Format `<coupling value><NL>`

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 217

:CHANnel<n>:DISPlay

C (see [page 794](#))

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

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

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

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

Query Syntax :CHANnel<n>:DISPlay?

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

Return Format <display value><NL>

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

- See Also**
- "[Introduction to :CHANnel<n> Commands](#)" on page 217
 - "[:VIEW](#)" on page 180
 - "[:BLANK](#)" on page 156
 - "[:STATus](#)" on page 177
 - "[:POD<n>:DISPlay](#)" on page 415
 - "[:DIGital<d>:DISPlay](#)" on page 245

:CHANnel<n>:IMPedance (see [page 794](#))**Command Syntax** `:CHANnel<n>:IMPedance <impedance>` `<impedance> ::= ONEMeg` `<n> ::= 1 to (# analog channels) in NR1 format`

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

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

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

Return Format `<impedance value><NL>` `<impedance value> ::= ONEM`**See Also** • "Introduction to :CHANnel<n> Commands" on page 217

:CHANnel<n>:INVert**N** (see [page 794](#))**Command Syntax** `:CHANnel<n>:INVert <invert value>` `<invert value> ::= {{1 | ON} | {0 | OFF}}` `<n> ::= 1 to (# analog channels) in NR1 format`

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

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

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

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

- "Introduction to :CHANnel<n> Commands" on page 217

:CHANnel<n>:LABel

N (see [page 794](#))

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

NOTE

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

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

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

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

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

See Also

- "[Introduction to :CHANnel<n> Commands](#)" on page 217
- "[":DISPlay:LABel](#)" on page 260
- "[":DIGItal<d>:LABel](#)" on page 246
- "[":DISPlay:LABList](#)" on page 261
- "[":BUS<n>:LABel](#)" on page 202

Example Code

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

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:CHANnel<n>:OFFSet



(see [page 794](#))

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

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

<suffix> ::= {V | mV}

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

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

Query Syntax :CHANnel<n>:OFFSet?

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

Return Format <offset><NL>

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

- See Also**
- ["Introduction to :CHANnel<n> Commands" on page 217](#)
 - [":CHANnel<n>:RANGE" on page 232](#)
 - [":CHANnel<n>:SCALE" on page 233](#)
 - [":CHANnel<n>:PROBe" on page 226](#)

:CHANnel<n>:PROBe

(see [page 794](#))

Command Syntax

```
:CHANnel<n>:PROBe <attenuation>
<attenuation> ::= probe attenuation ratio in NR3 format
<n> ::= 1 to (# analog channels) in NR1 format
The obsolete attenuation values X1, X10, X20, X100 are also supported.
```

The :CHANnel<n>:PROBe command specifies the probe attenuation factor for the selected channel. The probe attenuation factor may be 0.1 to 10000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors, for making automatic measurements, and for setting trigger levels.

Query Syntax

```
:CHANnel<n>:PROBe?
```

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

Return Format

```
<attenuation><NL>
```

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

- See Also**
- ["Introduction to :CHANnel<n> Commands" on page 217](#)
 - [":CHANnel<n>:RANGE" on page 232](#)
 - [":CHANnel<n>:SCALe" on page 233](#)
 - [":CHANnel<n>:OFFSet" on page 225](#)

Example Code

```
' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
' channel. The probe attenuation factor may be set from 0.1 to 10000
'
myScope.WriteString ":CHANnel1:PROBe 10" ' Set Probe to 10:1.
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

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

 (see [page 794](#))

Command Syntax

NOTE

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

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

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

- SEND0 – Single-ended, 0dB.
- SEND6 – Single-ended, 6dB.
- SEND12 – Single-ended, 12dB.
- SEND20 – Single-ended, 20dB.
- DIFF0 – Differential, 0dB.
- DIFF6 – Differential, 6dB.
- DIFF12 – Differential, 12dB.
- DIFF20 – Differential, 20dB.

Query Syntax :CHANnel<n>:PROBe:HEAD [:TYPE] ?

The :CHANnel<n>:PROBe:HEAD[:TYPE]? query returns the current probe head type setting for the selected channel.

Return Format <head_param><NL>

```
<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6
                  | DIFF12 | DIFF20 | NONE}
```

See Also

- "[Introduction to :CHANnel<n> Commands](#)" on page 217
- "[":CHANnel<n>:PROBe"](#) on page 226
- "[":CHANnel<n>:PROBe:ID"](#) on page 228
- "[":CHANnel<n>:PROBe:SKEW"](#) on page 229
- "[":CHANnel<n>:PROBe:STYPe"](#) on page 230

:CHANnel<n>:PROBe:ID(see [page 794](#))**Query Syntax** `:CHANnel<n>:PROBe:ID?``<n> ::= 1 to (# analog channels) in NR1 format`

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

Return Format `<probe id><NL>``<probe id> ::= unquoted ASCII string up to 11 characters`

Some of the possible returned values are:

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

See Also ["Introduction to :CHANnel<n> Commands" on page 217](#)

:CHANnel<n>:PROBe:SKEW



(see [page 794](#))

Command Syntax :CHANnel<n>:PROBe:SKEW <skew value>

<skew value> ::= skew time in NR3 format

<skew value> ::= -100 ns to +100 ns

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

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

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

The :CHANnel<n>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

Return Format <skew value><NL>

<skew value> ::= skew value in NR3 format

See Also • ["Introduction to :CHANnel<n> Commands" on page 217](#)

:CHANnel<n>:PROBe:STYPe

(see [page 794](#))

Command Syntax

NOTE

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

```
:CHANnel<n>:PROBe:STYPe <signal type>
<signal type> ::= {DIFFerential | SINGLE}
<n> ::= 1 to (# analog channels) in NR1 format
```

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

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

Query Syntax

```
:CHANnel<n>:PROBe:STYPe?
```

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

Return Format

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

See Also

- ["Introduction to :CHANnel<n> Commands" on page 217](#)
- [":CHANnel<n>:OFFSet" on page 225](#)

:CHANnel<n>:PROTection

N (see [page 794](#))

Command Syntax :CHANnel<n>:PROTection[:CLEar]

```
<n> ::= 1 to (# analog channels) in NR1 format | 4}
```

With the 2000 X-Series oscilloscopes, the analog channel input impedance is always $1\text{ M}\Omega$, so automatic overvoltage protection is not necessary (as it is for channels with 50Ω input impedance). There are no protection settings to clear, so the :CHANnel<n>:PROTection[:CLEar] command does nothing.

Query Syntax :CHANnel<n>:PROTection?

The :CHANnel<n>:PROTection query always returns NORM (normal).

Return Format NORM<NL>

- See Also**
- "[Introduction to :CHANnel<n> Commands](#)" on page 217
 - "[":CHANnel<n>:COUPling](#)" on page 220
 - "[":CHANnel<n>:PROBe](#)" on page 226

:CHANnel<n>:RANGE

(see [page 794](#))

Command Syntax `:CHANnel<n>:RANGE <range>[<suffix>]`

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

`<suffix>` ::= {V | mV}

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

The :CHANnel<n>:RANGE command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, legal values for the range are from 8 mV to 40 V.

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

Query Syntax `:CHANnel<n>:RANGE?`

The :CHANnel<n>:RANGE? query returns the current full-scale range setting for the specified channel.

Return Format `<range_argument><NL>`

`<range_argument>` ::= vertical full-scale range value in NR3 format

- See Also**
- "[Introduction to :CHANnel<n> Commands](#)" on page 217
 - "[":CHANnel<n>:SCALe](#)" on page 233
 - "[":CHANnel<n>:PROBe](#)" on page 226

Example Code

```
' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
' range value is 8 times the volts per division.
myScope.WriteString ":CHANnel1:RANGE 8"    ' Set the vertical range to
8 volts.
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:CHANnel<n>:SCALe

N (see [page 794](#))

Command Syntax	<code>:CHANnel<n>:SCALe <scale>[<suffix>]</code>
	<code><scale></code> ::= vertical units per division in NR3 format
	<code><suffix></code> ::= {V mV}
	<code><n></code> ::= 1 to (# analog channels) in NR1 format
	The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel.
	If the probe attenuation is changed, the scale value is multiplied by the probe's attenuation factor.
Query Syntax	<code>:CHANnel<n>:SCALe?</code>
	The :CHANnel<n>:SCALe? query returns the current scale setting for the specified channel.
Return Format	<code><scale value><NL></code>
	<code><scale value></code> ::= vertical units per division in NR3 format
See Also	<ul style="list-style-type: none"> • "Introduction to :CHANnel<n> Commands" on page 217 • "":CHANnel<n>:RANGE" on page 232 • "":CHANnel<n>:PROBe" on page 226

:CHANnel<n>:UNITS**N** (see [page 794](#))**Command Syntax** `:CHANnel<n>:UNITS <units>` `<units> ::= {VOLT | AMPere}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :CHANnel<n>:UNITS command sets the measurement units for the connected probe. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

Query Syntax `:CHANnel<n>:UNITS?`

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

Return Format `<units><NL>` `<units> ::= {VOLT | AMP}`**See Also**

- "[Introduction to :CHANnel<n> Commands](#)" on page 217
- "[":CHANnel<n>:RANGE](#)" on page 232
- "[":CHANnel<n>:PROBe](#)" on page 226
- "[":EXTernal:UNITS](#)" on page 277

:CHANnel<n>:VERNier**N** (see [page 794](#))**Command Syntax** `:CHANnel<n>:VERNier <vernier value>` `<vernier value> ::= {{1 | ON} | {0 | OFF}}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :CHANnel<n>:VERNier command specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).

Query Syntax `:CHANnel<n>:VERNier?`

The :CHANnel<n>:VERNier? query returns the current state of the channel's vernier setting.

Return Format `<vernier value><NL>` `<vernier value> ::= {0 | 1}`**See Also**

- "Introduction to :CHANnel<n> Commands" on page 217

11 :DEMO Commands

When the education kit is licensed (Option EDU), you can output demonstration signals on the oscilloscope's Demo 1 and Demo 2 terminals. See "["Introduction to :DEMO Commands"](#) on page 237.

Table 61 :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCTION <signal> (see page 238)	:DEMO:FUNCTION? (see page 239)	<signal> ::= {SINusoid NOISy PHASe RINGing SINGle AM CLK GLITch BURSt MSO RFBurst LFSine FMBurst}
:DEMO:FUNCTION:PHASE: PHASE <angle> (see page 240)	:DEMO:FUNCTION:PHASE: PHASE? (see page 240)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0 OFF} {1 ON}} (see page 241)	:DEMO:OUTPut? (see page 241)	{0 1}

Introduction to :DEMO Commands The :DEMO subsystem provides commands to output demonstration signals on the oscilloscope's Demo 1 and Demo 2 terminals.

Reporting the Setup

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

Return Format

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

```
:DEMO:FUNC SIN;OUTP 0
```

:DEMO:FUNCTION

N (see [page 794](#))

Command Syntax `:DEMO:FUNCTION <signal>`

```
<signal> ::= {SINusoid | NOISy | PHASE | RINGing | SINGle | AM | CLK
               | GLITch | BURSt | MSO | RFBurst | LFSine | FMBurst}
```

The :DEMO:FUNCTION command selects the type of demo signal:

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
SINusoid	5 MHz sine wave @ ~ 6 Vpp, 0 V offset	Off
NOISy	1 kHz sine wave @ ~ 2.4 Vpp, 0.0 V offset, with ~ 0.5 Vpp of random noise added	Off
PHASE	1 kHz sine wave @ 2.4 Vpp, 0.0 V offset	1 kHz sine wave @ 2.4 Vpp, 0.0 V offset , phase shifted by the amount entered using the " ":DEMO:FUNCTION:PHASE:PHASE " on page 240 command
RINGing	500 kHz digital pulse @ ~ 3 Vpp, 1.5 V offset, and ~500 ns pulse width with ringing	Off
SINGle	~500 ns wide digital pulse with ringing @ ~ 3 Vpp, 1.5 V offset Press the front panel Set Off Single-Shot softkey to cause the selected single-shot signal to be output.	Off
AM	26 kHz sine wave, ~ 7 Vpp, 0 V offset	Amplitude modulated signal, ~ 3 Vpp, 0 V offset, with ~13 MHz carrier and sine envelope
CLK	500 kHz clock @ ~2 Vpp, 1 V offset, with infrequent glitch (1 glitch per 50,000 clocks)	Off
GLITch	Burst of 6 digital pulses (plus infrequent glitch) that occurs once every 80 µs @ ~3.6 Vpp, ~1.8 V offset	Off
BURSt	Burst of digital pulses that occur every 50 µs @ ~ 3.6 Vpp, ~1.5 V offset	Off

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
MSO	3.1 kHz stair-step sine wave output of DAC @ ~1.5 Vpp, 0.75 V offset DAC input signals are internally routed to digital channels D0 through D7	~3.1 kHz sine wave filtered from DAC output @ ~ 600 mVpp, 300 mV offset
RFBurst	5-cycle burst of a 10 MHz amplitude modulated sine wave @ ~ 2.6 Vpp, 0 V offset occurring once every 4 ms	Off
LFSine	30 Hz sine wave @ ~2.7 Vpp, 0 V offset, with very narrow glitch near each positive peak	Off
FMBurst	FM burst, modulated from ~100 kHz to ~1 MHz, ~5.0 Vpp, ~600 mV offset.	Off

Query Syntax :DEMO:FUNCTION?

The :DEMO:FUNCTION? query returns the currently selected demo signal type.

Return Format

```
<signal><NL>
<signal> ::= {SIN | NOIS | PHAS | RING | SING | AM | CLK | GLIT | BURS
               | MSO | RFB | LFS | FMB}
```

See Also

- "Introduction to :DEMO Commands" on page 237

:DEMO:FUNCTION:PHASE:PHASE

N (see [page 794](#))

Command Syntax `:DEMO:FUNCTION:PHASE:PHASE <angle>`

`<angle>` ::= angle in degrees from 0 to 360 in NR3 format

For the phase shifted sine demo signals, the :DEMO:FUNCTION:PHASE:PHASE command specifies the phase shift in the second sine waveform.

Query Syntax `:DEMO:FUNCTION:PHASE:PHASE?`

The :DEMO:FUNCTION:PHASE:PHASE? query returns the currently set phase shift.

Return Format `<angle><NL>`

`<angle>` ::= angle in degrees from 0 to 360 in NR3 format

See Also

- "Introduction to :DEMO Commands" on page 237
- "[:DEMO:FUNCTION](#)" on page 238

:DEMO:OUTPut

N (see [page 794](#))

Command Syntax `:DEMO:OUTPut <on_off>`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :DEMO:OUTPut command specifies whether the demo signal output is ON (1) or OFF (0).

Query Syntax `:DEMO:OUTPut?`

The :DEMO:OUTPut? query returns the current state of the demo signal output setting.

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also

- "Introduction to :DEMO Commands" on page 237
- ":DEMO:FUNCTION" on page 238

12 :DIGItal<d> Commands

Control all oscilloscope functions associated with individual digital channels. See "[Introduction to :DIGItal<d> Commands](#)" on page 244.

Table 62 :DIGItal<d> Commands Summary

Command	Query	Options and Query Returns
:DIGItal<d>:DISPlay { {0 OFF} {1 ON}} (see page 245)	:DIGItal<d>:DISPlay? (see page 245)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 1}
:DIGItal<d>:LABel <string> (see page 246)	:DIGItal<d>:LABel? (see page 246)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGItal<d>:POSIon <position> (see page 247)	:DIGItal<d>:POSITION? (see page 247)	<d> ::= 0 to (# digital channels - 1) in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small Returns -1 when there is no space to display the digital waveform.
:DIGItal<d>:SIZE <value> (see page 248)	:DIGItal<d>:SIZE? (see page 248)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMAll MEDium LARGe}
:DIGItal<d>:THreshold <value>[suffix] (see page 249)	:DIGItal<d>:THreshold? (see page 249)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {CMOS ECL TTL <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V mV uV}

Introduction to :DIGITAL<d> Commands	<p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p>The DIGITAL subsystem commands control the viewing, labeling, and positioning of digital channels. They also control threshold settings for groups of digital channels, or <i>pods</i>.</p>
---	---

NOTE

These commands are only valid for the MSO models.

Reporting the Setup

Use :DIGITAL<d>? to query setup information for the DIGITAL subsystem.

Return Format

The following is a sample response from the :DIGITAL0? query. In this case, the query was issued following a *RST command.

```
:DIG0:DISP 0;THR +1.40E+00;LAB 'D0';POS +0
```

:DIGItal<d>:DISPlay

N (see [page 794](#))

Command Syntax :DIGItal<d>:DISPlay <display>

```
<d> ::= 0 to (# digital channels - 1) in NR1 format
<display> ::= {{1 | ON} | {0 | OFF}}
```

The :DIGItal<d>:DISPlay command turns digital display on or off for the specified channel.

NOTE

This command is only valid for the MSO models.

Query Syntax :DIGItal<d>:DISPlay?

The :DIGItal<d>:DISPlay? query returns the current digital display setting for the specified channel.

Return Format <display><NL>

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

See Also

- "[Introduction to :DIGItal<d> Commands](#)" on page 244
- "[":POD<n>:DISPlay](#)" on page 415
- "[":CHANnel<n>:DISPlay](#)" on page 221
- "[":VIEW](#)" on page 180
- "[":BLANK](#)" on page 156
- "[":STATus](#)" on page 177

:DIGital<d>:LABel

N (see [page 794](#))

Command Syntax

```
:DIGital<d>:LABel <string>
<d> ::= 0 to (# digital channels - 1) in NR1 format
<string> ::= any series of 10 or less characters as quoted ASCII string.
```

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

NOTE

This command is only valid for the MSO models.

NOTE

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

Query Syntax

```
:DIGital<d>:LABel?
```

The :DIGital<d>:LABel? query returns the name of the specified channel.

Return Format

```
<label string><NL>
<label string> ::= any series of 10 or less characters as a quoted
                  ASCII string.
```

See Also

- "[Introduction to :DIGital<d> Commands](#)" on page 244
- "[:CHANnel<n>:LABel](#)" on page 224
- "[:DISPlay:LABList](#)" on page 261
- "[:BUS<n>:LABel](#)" on page 202

:DIGItal<d>:POsition

N (see [page 794](#))

Command Syntax :DIGItal<d>:POsition <position>

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

Channel Size	Position	Top	Bottom
Large	0-7	7	0
Medium	0-15	15	0
Small	0-31	31	0

The :DIGItal<d>:POsition command sets the position of the specified channel. Note that bottom positions might not be valid depending on whether digital buses, serial decode waveforms, or the zoomed time base are displayed.

NOTE

This command is only valid for the MSO models.

Query Syntax :DIGItal<d>:POsition?

The :DIGItal<d>:POsition? query returns the position of the specified channel.

If the returned value is "-1", this indicates there is no space to display the digital waveform (for example, when all serial lanes, digital buses, and the zoomed time base are displayed).

Return Format

```
<position><NL>
<position> ::= integer in NR1 format.
```

See Also

- ["Introduction to :DIGItal<d> Commands" on page 244](#)

:DIGital<d>:SIZE

N (see [page 794](#))

Command Syntax `:DIGital<d>:SIZE <value>`

```
<d> ::= 0 to (# digital channels - 1) in NR1 format
<value> ::= {SMALL | MEDium | LARGe}
```

The :DIGital<d>:SIZE command specifies the size of digital channels on the display. Sizes are set for all digital channels. Therefore, if you set the size on digital channel 0 (for example), the same size is set on all other as well.

NOTE

This command is only valid for the MSO models.

Query Syntax `:DIGital<d>:SIZE?`

The :DIGital<d>:SIZE? query returns the size setting for the specified digital channels.

Return Format `<size_value><NL>`

```
<size_value> ::= {SMAL | MED | LARG}
```

See Also

- "[Introduction to :DIGital<d> Commands](#)" on page 244
- "[":POD<n>:SIZE](#)" on page 416
- "[":DIGital<d>:POStion](#)" on page 247

:DIGItal<d>:THreshold

N (see [page 794](#))

Command Syntax :DIGItal<d>:THreshold <value>

```

<d> ::= 0 to (# digital channels - 1) in NR1 format
<value> ::= {CMOS | ECL | TTL | <user defined value>[<suffix>] }
<user defined value> ::= -8.00 to +8.00 in NR3 format
<suffix> ::= {V | mV | uV}
  • TTL = 1.4V
  • CMOS = 2.5V
  • ECL = -1.3V

```

The :DIGItal<d>:THreshold command sets the logic threshold value for all channels in the same *pod* as the specified channel. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE

This command is only valid for the MSO models.

Query Syntax :DIGItal<d>:THreshold?

The :DIGItal<d>:THreshold? query returns the threshold value for the specified channel.

Return Format <value><NL>

```
<value> ::= threshold value in NR3 format
```

See Also

- "[Introduction to :DIGItal<d> Commands](#)" on page 244
- "[":POD<n>:THreshold](#)" on page 417
- "[":TRIGger\[:EDGE\]:LEVel](#)" on page 605

13 :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "[Introduction to :DISPlay Commands](#)" on page 252.

Table 63 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation {{0 OFF} {1 ON}} (see page 253)	:DISPlay:ANNotation? (see page 253)	{0 1}
:DISPlay:ANNotation:B ACKground <mode> (see page 254)	:DISPlay:ANNotation:B ACKground? (see page 254)	<mode> ::= {OPAQue INVerted TRANsparent}
:DISPlay:ANNotation:C OLor <color> (see page 255)	:DISPlay:ANNotation:C OLor? (see page 255)	<color> ::= {CH1 CH2 CH3 CH4 DIG MATH REF MARKer WHITe RED}
:DISPlay:ANNotation:T EXT <string> (see page 256)	:DISPlay:ANNotation:T EXT? (see page 256)	<string> ::= quoted ASCII string (up to 254 characters)
:DISPlay:CLEar (see page 257)	n/a	n/a
n/a	:DISPlay:DATA? [<format>] [,] [<palett e>] (see page 258)	<format> ::= {BMP BMP8bit PNG} <palette> ::= {COLor GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:INTensity:WA Veform <value> (see page 259)	:DISPlay:INTensity:WA Veform? (see page 259)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:LABEL {{0 OFF} {1 ON}} (see page 260)	:DISPlay:LABEL? (see page 260)	{0 1}

Table 63 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:LABList <binary block> (see page 261)	:DISPlay:LABList? (see page 261)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:PERSistence <value> (see page 262)	:DISPlay:PERSistence? (see page 262)	<value> ::= {MINimum INFinite <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {1 ON} (see page 263)	:DISPlay:VECTors? (see page 263)	1

Introduction to :DISPlay Commands The DISPlay subsystem is used to control the display storage and retrieval of waveform data, labels, and text. This subsystem allows the following actions:

- Clear the waveform area on the display.
- Turn vectors on or off.
- Set waveform persistence.
- Specify labels.
- Save and Recall display data.

Reporting the Setup

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

Return Format

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

```
:DISP:LAB 0;VECT 1;PERS MIN
```

:DISPlay:ANAnnotation

N (see [page 794](#))

Command Syntax `:DISPlay:ANAnnotation <setting>`
`<setting> ::= {{1 | ON} | {0 | OFF}}`

The :DISPlay:ANAnnotation command turns the annotation on and off. When on, the annotation appears in the upper left corner of the oscilloscope's display.

The annotation is useful for documentation purposes, to add notes before capturing screens.

Query Syntax `:DISPlay:ANAnnotation?`

The :DISPlay:ANAnnotation? query returns the annotation setting.

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

See Also

- "[:DISPlay:ANAnnotation:TEXT](#)" on page 256
- "[:DISPlay:ANAnnotation:COLor](#)" on page 255
- "[:DISPlay:ANAnnotation:BACKground](#)" on page 254
- "[Introduction to :DISPlay Commands](#)" on page 252

:DISPlay:ANNotation:BACKground**N** (see [page 794](#))**Command Syntax** `:DISPlay:ANNotation:BACKground <mode>``<mode> ::= {OPAQue | INVerted | TRANsparent}`

The :DISPlay:ANNotation:BACKground command specifies the background of the annotation:

- OPAQue – the annotation has a solid background.
- INVerted – the annotation's foreground and background colors are switched.
- TRANsparent – the annotation has a transparent background.

Query Syntax `:DISPlay:ANNotation:BACKground?`

The :DISPlay:ANNotation:BACKground? query returns the specified annotation background mode.

Return Format `<mode><NL>``<mode> ::= {OPAQ | INV | TRAN}`**See Also**

- "[:DISPlay:ANNotation](#)" on page 253
- "[:DISPlay:ANNotation:TEXT](#)" on page 256
- "[:DISPlay:ANNotation:COLor](#)" on page 255
- "[Introduction to :DISPlay Commands](#)" on page 252

:DISPlay:ANNotation:COLor

N (see [page 794](#))

Command Syntax `:DISPlay:ANNotation:COLor <color>`

$$<\text{color}> ::= \{\text{CH1} \mid \text{CH2} \mid \text{CH3} \mid \text{CH4} \mid \text{DIG} \mid \text{MATH} \mid \text{REF} \mid \text{MARKer} \mid \text{WHITE} \mid \text{RED}\}$$

The :DISPlay:ANNotation:COLor command specifies the annotation color. You can choose white, red, or colors that match analog channels, digital channels, math waveforms, reference waveforms, or markers.

Query Syntax `:DISPlay:ANNotation:COLor?`

The :DISPlay:ANNotation:COLor? query returns the specified annotation color.

Return Format `<color><NL>`

$$<\text{color}> ::= \{\text{CH1} \mid \text{CH2} \mid \text{CH3} \mid \text{CH4} \mid \text{DIG} \mid \text{MATH} \mid \text{REF} \mid \text{MARK} \mid \text{WHIT} \mid \text{RED}\}$$

See Also

- "[:DISPlay:ANNotation](#)" on page 253
- "[:DISPlay:ANNotation:TEXT](#)" on page 256
- "[:DISPlay:ANNotation:BACKground](#)" on page 254
- "[Introduction to :DISPlay Commands](#)" on page 252

:DISPlay:ANAnnotation:TEXT

N (see [page 794](#))

Command Syntax `:DISPlay:ANAnnotation:TEXT <string>`

`<string> ::= quoted ASCII string (up to 254 characters)`

The :DISPlay:ANAnnotation:TEXT command specifies the annotation string. The annotation string can contain as many characters as will fit in the Edit Annotation box on the oscilloscope's screen, up to 254 characters.

You can include a carriage return in the annotation string using the characters "\n". Note that this is not a new line character but the actual "\" (backslash) and "n" characters in the string. Carriage returns lessen the number of characters available for the annotation string.

Use :DISPlay:ANAnnotation:TEXT "" to remotely clear the annotation text. (Two sets of quote marks without a space between them creates a NULL string.)

Query Syntax `:DISPlay:ANAnnotation:TEXT?`

The :DISPlay:ANAnnotation:TEXT? query returns the specified annotation text.

When carriage returns are present in the annotation text, they are returned as the actual carriage return character (ASCII 0x0D).

Return Format `<string><NL>`

`<string> ::= quoted ASCII string`

See Also

- [":DISPlay:ANAnnotation"](#) on page 253

- [":DISPlay:ANAnnotation:COLor"](#) on page 255

- [":DISPlay:ANAnnotation:BACKground"](#) on page 254

- ["Introduction to :DISPlay Commands"](#) on page 252

:DISPlay:CLEar

N (see [page 794](#))

Command Syntax :DISPlay:CLEar

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

See Also • ["Introduction to :DISPlay Commands"](#) on page 252

:DISPlay:DATA

N (see [page 794](#))

Query Syntax `:DISPlay:DATA? [<format> [,] [<palette>]`
`<format> ::= {BMP | BMP8bit | PNG}`
`<palette> ::= {COLOR | GRAYscale}`

The `:DISPlay:DATA?` query reads screen image data. You can choose 24-bit BMP, 8-bit BMP8bit, or 24-bit PNG formats in color or grayscale.

If no format or palette option is specified, the screen image is returned in BMP, COLOR format.

Screen image data is returned in the IEEE-488.2 # binary block data format.

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

See Also

- "[Introduction to :DISPlay Commands](#)" on page 252
- "[":HARDcopy:INKSaver](#)" on page 303
- "[":PRINT](#)" on page 173
- "[":RCL \(Recall\)](#)" on page 133
- "[":SAV \(Save\)](#)" on page 137
- "[":VIEW](#)" on page 180

Example Code

```
' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPlay:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPlay:DATA? BMP, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
    Kill strPath
End If
Close #1      ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1      ' Open file for
or output.
Put #1, , byteData      ' Write data.
Close #1      ' Close file.
myScope.IO.Timeout = 5000
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:DISPlay:INTensity:WAveform

N (see [page 794](#))

Command Syntax :DISPlay:INTensity:WAveform <value>

<value> ::= an integer from 0 to 100 in NR1 format.

The :DISPlay:INTensity:WAveform command sets the waveform intensity.

This is the same as adjusting the front panel [**Intensity**] knob.

Query Syntax :DISPlay:INTensity:WAveform?

The :DISPlay:INTensity:WAveform? query returns the waveform intensity setting.

Return Format <value><NL>

<value> ::= an integer from 0 to 100 in NR1 format.

See Also • "Introduction to :DISPlay Commands" on page 252

:DISPlay:LABel**N** (see [page 794](#))**Command Syntax** `:DISPlay:LABel <value>``<value> ::= {{1 | ON} | {0 | OFF}}`

The :DISPlay:LABel command turns the analog and digital channel labels on and off.

Query Syntax `:DISPlay:LABel?`

The :DISPlay:LABel? query returns the display mode of the analog and digital labels.

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

- See Also**
- "[Introduction to :DISPlay Commands](#)" on page 252
 - "[":CHANnel<n>:LABel](#)" on page 224

Example Code

```
' DISP_LABEL  
' - Turns label names ON or OFF on the analyzer display.  
myScope.WriteString ":DISPlay:LABel ON" ' Turn on labels.
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:DISPlay:LABList

N (see [page 794](#))

Command Syntax `:DISPlay:LABList <binary block data>`
`<binary block> ::= an ordered list of up to 75 labels, a maximum of 10 characters each, separated by newline characters.`

The :DISPlay:LABList command adds labels to the label list. Labels are added in alphabetical order.

NOTE

Labels that begin with the same alphabetic base string followed by decimal digits are considered duplicate labels. Duplicate labels are not added to the label list. For example, if label "A0" is in the list and you try to add a new label called "A123456789", the new label is not added.

Query Syntax `:DISPlay:LABList?`

The :DISPlay:LABList? query returns the label list.

Return Format `<binary block><NL>`

`<binary block> ::= an ordered list of up to 75 labels, a maximum of 10 characters each, separated by newline characters.`

See Also

- "[Introduction to :DISPlay Commands](#)" on page 252
- "[":DISPlay:LABel](#)" on page 260
- "[":CHANnel<n>:LABel](#)" on page 224
- "[":DIGital<d>:LABel](#)" on page 246
- "[":BUS<n>:LABel](#)" on page 202

:DISPlay:PERSistence

N (see [page 794](#))

Command Syntax `:DISPlay:PERSistence <value>`

`<value> ::= {MINimum | INFinite | <time>}`

`<time> ::= seconds in in NR3 format from 100E-3 to 60E0`

The :DISPlay:PERSistence command specifies the persistence setting:

- MINimum – indicates zero persistence.
- INFinite – indicates infinite persistence.
- <time> – for variable persistence, that is, you can specify how long acquisitions remain on the screen.

Use the :DISPlay:CLEar command to erase points stored by persistence.

Query Syntax `:DISPlay:PERSistence?`

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

Return Format `<value><NL>`

`<value> ::= {MIN | INF | <time>}`

See Also

- "[Introduction to :DISPlay Commands](#)" on page 252
- "[":DISPlay:CLEar](#)" on page 257

:DISPlay:VECTors

N (see [page 794](#))

Command Syntax `:DISPlay:VECTors <vectors>`
 `<vectors> ::= {1 | ON}`

The only legal value for the :DISPlay:VECTors command is ON (or 1). This specifies that lines are drawn between acquired data points on the screen.

Query Syntax `:DISPlay:VECTors?`
The :DISPlay:VECTors? query returns the vectors setting.

Return Format `<vectors><NL>`
 `<vectors> ::= 1`

See Also • "Introduction to :DISPlay Commands" on page 252

14 :DVM Commands

When the optional DSOXDVM digital voltmeter analysis feature is licensed, these commands control the digital voltmeter (DVM) feature.

Table 64 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARAnge {{0 OFF} {1 ON}} (see page 266)	:DVM:ARAnge? (see page 266)	{0 1}
n/a	:DVM:CURREnt? (see page 267)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABLE {{0 OFF} {1 ON}} (see page 268)	:DVM:ENABLE? (see page 268)	{0 1}
n/a	:DVM:FREQuency? (see page 267)	<freq_value> ::= floating-point number in NR3 format
:DVM:MODE <mode> (see page 270)	:DVM:MODE? (see page 270)	<dvm_mode> ::= {ACRMs DC DCRMs FREQuency}
:DVM:SOURce <source> (see page 271)	:DVM:SOURce? (see page 271)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format

:DVM:ARAnge

N (see [page 794](#))

Command Syntax `:DVM:ARAnge <setting>`

`<setting> ::= {{OFF | 0} | {ON | 1}}`

If the selected digital voltmeter (DVM) source channel is not used in oscilloscope triggering, the :DVM:ARAnge command turns the digital voltmeter's Auto Range capability on or off.

- When on, the DVM channel's vertical scale, vertical (ground level) position, and trigger (threshold voltage) level (used for the counter frequency measurement) are automatically adjusted.

The Auto Range capability overrides attempted adjustments of the channel's vertical scale and position.

- When off, you can adjust the channel's vertical scale and position normally.

Query Syntax `:DVM:ARAnge?`

The :DVM:ARAnge? query returns a flag indicating whether the digital voltmeter's Auto Range capability is on or off.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also [":DVM:SOURce"](#) on page 271
[":DVM:ENABLE"](#) on page 268
[":DVM:MODE"](#) on page 270

:DVM:CURREnt**N** (see [page 794](#))**Query Syntax** `:DVM:CURREnt?`

The :DVM:CURREnt? query returns the displayed 3-digit DVM value based on the current mode.

Return Format `<dvm_value><NL>`

`<dvm_value>` ::= floating-point number in NR3 format

- See Also**
- "[:DVM:SOURce](#)" on page 271
 - "[:DVM:ENABLE](#)" on page 268
 - "[:DVM:MODE](#)" on page 270
 - "[:DVM:FREQuency](#)" on page 269

:DVM:ENABLE**N** (see [page 794](#))**Command Syntax** `:DVM:ENABLE <setting>``<setting> ::= {{OFF | 0} | {ON | 1}}`

The :DVM:ENABLE command turns the digital voltmeter (DVM) analysis feature on or off.

Query Syntax `:DVM:ENABLE?`

The :DVM:ENABLE? query returns a flag indicating whether the digital voltmeter (DVM) analysis feature is on or off.

Return Format `<setting><NL>``<setting> ::= {0 | 1}`**See Also**

- [":DVM:SOURce" on page 271](#)
- [":DVM:MODE" on page 270](#)
- [":DVM:ARAnge" on page 266](#)

:DVM:FREQuency

N (see [page 794](#))

Query Syntax :DVM:FREQuency?

The :DVM:FREQuency? query returns the displayed 5-digit frequency value that is displayed below the main DVM value.

Return Format <freq_value><NL>

<freq_value> ::= floating-point number in NR3 format

See Also

- "[:DVM:SOURce](#)" on page 271
- "[:DVM:ENABLE](#)" on page 268
- "[:DVM:MODE](#)" on page 270
- "[:DVM:CURREnt](#)" on page 267

:DVM:MODE**N** (see [page 794](#))**Command Syntax** `:DVM:MODE <dvm_mode>``<dvm_mode> ::= {ACRMs | DC | DCRMs | FREQuency}`

The :DVM:MODE command sets the digital voltmeter (DVM) mode:

- ACRMs – displays the root-mean-square value of the acquired data, with the DC component removed.
- DC – displays the DC value of the acquired data.
- DCRMs – displays the root-mean-square value of the acquired data.
- FREQuency – displays the frequency counter measurement.

Query Syntax `:DVM:MODE?`

The :DVM:MODE? query returns the selected DVM mode.

Return Format `<dvm_mode><NL>``<dvm_mode> ::= {ACRM | DC | DCRM | FREQ}`**See Also**

- "[:DVM:ENABLE](#)" on page 268
- "[:DVM:SOURce](#)" on page 271
- "[:DVM:ARAnge](#)" on page 266
- "[:DVM:CURRent](#)" on page 267
- "[:DVM:FREQuency](#)" on page 269

:DVM:SOURce

N (see [page 794](#))

Command Syntax :DVM:SOURce <source>

```
<source> ::= {CHANnel<n>}
<n> ::= 1-2 or 1-4 in NR1 format
```

The :DVM:SOURce command sets the select the analog channel on which digital voltmeter (DVM) measurements are made.

The selected channel does not have to be on (displaying a waveform) in order for DVM measurements to be made.

Query Syntax :DVM:SOURce?

The :DVM:SOURce? query returns the selected DVM input source.

Return Format <source><NL>

```
<source> ::= {CHAN<n>}
<n> ::= 1-2 or 1-4 in NR1 format
```

- See Also**
- "[:DVM:ENABLE](#)" on page 268
 - "[:DVM:MODE](#)" on page 270
 - "[:DVM:ARAnge](#)" on page 266
 - "[:DVM:CURRent](#)" on page 267
 - "[:DVM:FREQuency](#)" on page 269

15 :EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See "[Introduction to :EXTernal Trigger Commands](#)" on page 273.

Table 65 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLIMIT <bwlimit> (see page 274)	:EXTernal:BWLIMIT? (see page 274)	<bwlimit> ::= {0 OFF}
:EXTernal:PROBE <attenuation> (see page 275)	:EXTernal:PROBE? (see page 275)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGE <range>[<suffix>] (see page 276)	:EXTernal:RANGE? (see page 276)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V mV}
:EXTernal:UNITS <units> (see page 277)	:EXTernal:UNITS? (see page 277)	<units> ::= {VOLT AMPere}

Introduction to :EXTernal Trigger Commands The EXTernal trigger subsystem commands control the input characteristics of the external trigger input. The probe factor, impedance, input range, input protection state, units, and bandwidth limit settings may all be queried. Depending on the instrument type, some settings may be changeable.

Reporting the Setup

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

Return Format

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

```
:EXT:BWL 0;RANG +8E+00;UNIT VOLT;PROB +1.000E+00
```

:EXTernal:BWLimits (see [page 794](#))

Command Syntax `:EXTernal:BWLimits <bwlimits>`
 `<bwlimits> ::= {0 | OFF}`

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

Query Syntax `:EXTernal:BWLimits?`

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

Return Format `<bwlimits><NL>`
 `<bwlimits> ::= 0`

See Also

- "[Introduction to :EXTernal Trigger Commands](#)" on page 273
- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:HFReject](#)" on page 595

:EXTernal:PROBe

 (see [page 794](#))

Command Syntax `:EXTernal:PROBe <attenuation>`

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

The :EXTernal:PROBe command specifies the probe attenuation factor for the external trigger. The probe attenuation factor may be 0.1 to 10000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors and for setting trigger levels.

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

Query Syntax `:EXTernal:PROBe?`

The :EXTernal:PROBe? query returns the current probe attenuation factor for the external trigger.

Return Format `<attenuation><NL>`

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

See Also

- "[Introduction to :EXTernal Trigger Commands](#)" on page 273
- "[":EXTernal:RANGE](#)" on page 276
- "[":INTroduction to :TRIGger Commands](#)" on page 591
- "[":CHANnel<n>:PROBe](#)" on page 226

:EXTernal:RANGE(see [page 794](#))**Command Syntax** `:EXTernal:RANGE <range>[<suffix>]` `<range> ::= vertical full-scale range value in NR3 format` `<suffix> ::= {V | mV}`

The :EXTernal:RANGE command is provided for product compatibility. When using 1:1 probe attenuation, the range can only be set to 8.0 V.

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

Query Syntax `:EXTernal:RANGE?`

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

Return Format `<range_argument><NL>` `<range_argument> ::= external trigger range value in NR3 format`**See Also**

- "[Introduction to :EXTernal Trigger Commands](#)" on page 273

- "[:EXTernal:PROBe](#)" on page 275

- "[Introduction to :TRIGger Commands](#)" on page 591

:EXTernal:UNITS**N** (see [page 794](#))

Command Syntax `:EXTernal:UNITS <units>`
 `<units> ::= {VOLT | AMPere}`

The :EXTernal:UNITS command sets the measurement units for the probe connected to the external trigger input. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

Query Syntax `:EXTernal:UNITS?`

The :CHANnel<n>:UNITS? query returns the current units setting for the external trigger.

Return Format `<units><NL>`
 `<units> ::= {VOLT | AMP}`

See Also

- "[Introduction to :EXTernal Trigger Commands](#)" on page 273
- "[Introduction to :TRIGger Commands](#)" on page 591
- "[":EXTernal:RANGE](#)" on page 276
- "[":EXTernal:PROBe](#)" on page 275
- "[":CHANnel<n>:UNITS](#)" on page 234

16 :FUNCTION Commands

Control functions in the measurement/storage module. See "[Introduction to :FUNCTION Commands](#)" on page 280.

Table 66 :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:DISPLAY {{0 OFF} {1 ON}} (see page 282)	:FUNCTION:DISPLAY? (see page 282)	{0 1}
:FUNCTION[:FFT]:CENTEr <frequency> (see page 283)	:FUNCTION[:FFT]:CENTEr? (see page 283)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCTION[:FFT]:SPAN (see page 284)	:FUNCTION[:FFT]:SPAN? (see page 284)	 ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCTION[:FFT]:VTPe <units> (see page 285)	:FUNCTION[:FFT]:VTPe? (see page 285)	<units> ::= {DECibel VRMS}
:FUNCTION[:FFT]:WINDOW <>window> (see page 286)	:FUNCTION[:FFT]:WINDOW? (see page 286)	<window> ::= {RECTangular HANNing FLATtop BHARris}
:FUNCTION:GOFT:OPERation <operation> (see page 287)	:FUNCTION:GOFT:OPERation? (see page 287)	<operation> ::= {ADD SUBTract MULTiply}
:FUNCTION:GOFT:SOURce 1 <source> (see page 288)	:FUNCTION:GOFT:SOURce 1? (see page 288)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see page 289)	:FUNCTION:GOFT:SOURce 2? (see page 289)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models

Table 66 :FUNCTION Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION:OFFSet <offset> (see page 290)	:FUNCTION:OFFSet? (see page 290)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:OPERation <operation> (see page 291)	:FUNCTION:OPERation? (see page 291)	<operation> ::= {ADD SUBTract MULTiply FFT}
:FUNCTION:RANGE <range> (see page 292)	:FUNCTION:RANGE? (see page 292)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the FFT function is 8 to 800 dBV.
:FUNCTION:REFERENCE <level> (see page 293)	:FUNCTION:REFERENCE? (see page 293)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:SCALE <scale value>[<suffix>] (see page 294)	:FUNCTION:SCALE? (see page 294)	<scale value> ::= integer in NR1 format <suffix> ::= {V dB}
:FUNCTION:SOURce1 <source> (see page 295)	:FUNCTION:SOURce1? (see page 295)	<source> ::= {CHANnel<n> GOFT} <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models GOFT is only for FFT operation.
:FUNCTION:SOURce2 <source> (see page 296)	:FUNCTION:SOURce2? (see page 296)	<source> ::= {CHANnel<n> NONE} <n> ::= {{1 2} {3 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 2} for 2ch models

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

The SOURce1, DISPLAY, RANGE, and OFFSet commands apply to any function.

The SPAN, CENTER, VTYPE, and WINDOW commands are only useful for FFT functions. When FFT is selected, the horizontal cursors change from time to frequency (Hz), and the vertical cursors change from volts to decibel (dB).

Reporting the Setup

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

Return Format

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

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

:FUNCTION:DISPLAY**N** (see [page 794](#))**Command Syntax** `:FUNCTION:DISPLAY <display>``<display> ::= {{1 | ON} | {0 | OFF}}`

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

Query Syntax `:FUNCTION:DISPLAY?`

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

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

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[:VIEW](#)" on page 180
- "[:BLANK](#)" on page 156
- "[:STATus](#)" on page 177

:FUNCTION[:FFT]:CENTer

N (see [page 794](#))

Command Syntax `:FUNCTION [:FFT] :CENTer <frequency>`

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

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

Query Syntax `:FUNCTION [:FFT] :CENTer?`

The :FUNCTION[:FFT]:CENTer? query returns the current center frequency in Hertz.

Return Format `<frequency><NL>`

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

NOTE

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION\[:FFT\]:SPAN"](#) on page 284
- "[":TIMEbase:RANGE"](#) on page 583
- "[":TIMEbase:SCALE"](#) on page 585

:FUNCTION[:FFT]:SPAN

N (see [page 794](#))

Command Syntax `:FUNCTION[:FFT]:SPAN `

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

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

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

Query Syntax `:FUNCTION[:FFT]:SPAN?`

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

NOTE

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

Return Format `<NL>`

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION\[:FFT\]:CENTer](#)" on page 283
- "[":TIMEbase:RANGE](#)" on page 583
- "[":TIMEbase:SCALE](#)" on page 585

:FUNCTION[:FFT]:VTYPe

N (see [page 794](#))

Command Syntax `:FUNCTION [:FFT] :VTYPe <units>`
`<units> ::= {DECibel | VRMS}`

The :FUNCTION[:FFT]:VTYPe command specifies FFT vertical units as DECibel or VRMS.

Query Syntax `:FUNCTION [:FFT] :VTYPe?`

The :FUNCTION[:FFT]:VTYPe? query returns the current FFT vertical units.

Return Format `<units><NL>`
`<units> ::= {DEC | VRMS}`

See Also

- ["Introduction to :FUNCTION Commands"](#) on page 280
- [":FUNCTION:OPERation"](#) on page 291

:FUNCTION[:FFT]:WINDOW

N (see [page 794](#))

Command Syntax `:FUNCTION [:FFT] :WINDOW <window>`
`<window> ::= {RECTangular | HANNing | FLATtop | BHARRis}`

The :FUNCTION[:FFT]:WINDOW command allows the selection of four different windowing transforms or operations for the FFT (Fast Fourier Transform) function.

The FFT operation assumes that the time record repeats. Unless an integral number of sampled waveform cycles exist in the record, a discontinuity is created between the end of one record and the beginning of the next. This discontinuity introduces additional frequency components about the peaks into the spectrum. This is referred to as leakage. To minimize leakage, windows that approach zero smoothly at the start and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input signals.

- RECTangular – useful for transient signals, and signals where there are an integral number of cycles in the time record.
- HANNing – useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements. This is the default window.
- FLATtop – best for making accurate amplitude measurements of frequency peaks.
- BHARRis (Blackman-Harris) – reduces time resolution compared to the rectangular window, but it improves the capacity to detect smaller impulses due to lower secondary lobes (provides minimal spectral leakage).

Query Syntax `:FUNCTION [:FFT] :WINDOW?`

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

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

See Also • "Introduction to :FUNCTION Commands" on page 280

:FUNCTION:GOFT:OPERation

N (see [page 794](#))

Command Syntax `:FUNCTION:GOFT:OPERation <operation>`

`<operation> ::= {ADD | SUBTract | MULTiply}`

The :FUNCTION:GOFT:OPERation command sets the math operation for the g(t) source that can be used as the input to the FFT function:

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

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

Query Syntax `:FUNCTION:GOFT:OPERation?`

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

Return Format `<operation><NL>`

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION:GOFT:SOURce1](#)" on page 288
- "[":FUNCTION:GOFT:SOURce2](#)" on page 289
- "[":FUNCTION:SOURce1](#)" on page 295

:FUNCTION:GOFT:SOURce1**N** (see [page 794](#))**Command Syntax** `:FUNCTION:GOFT:SOURce1 <value>`

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

The :FUNCTION:GOFT:SOURce1 command selects the first input channel for the g(t) source that can be used as the input to the FFT function.

Query Syntax `:FUNCTION:GOFT:SOURce1?`

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

Return Format `<value><NL>`

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION:GOFT:SOURce2"](#) on page 289
- "[":FUNCTION:GOFT:OPERation"](#) on page 287

:FUNCTION:GOFT:SOURce2

N (see [page 794](#))

Command Syntax `:FUNCTION:GOFT:SOURce2 <value>`

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

The :FUNCTION:GOFT:SOURce2 command selects the second input channel for the g(t) source that can be used as the input to the FFT function.

Query Syntax `:FUNCTION:GOFT:SOURce2?`

The :FUNCTION:GOFT:SOURce2? query returns the current selection for the second input channel for the g(t) source.

Return Format `<value><NL>`

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION:GOFT:SOURce1](#)" on page 288
- "[":FUNCTION:GOFT:OPERation](#)" on page 287

:FUNCTION:OFFSet

N (see [page 794](#))

Command Syntax `:FUNCTION:OFFSet <offset>`

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

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

NOTE

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

Query Syntax `:FUNCTION:OFFSet?`

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

Return Format `<offset><NL>`

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

See Also

- "Introduction to :FUNCTION Commands" on page 280
- ":FUNCTION:RANGE" on page 292
- ":FUNCTION:REFerence" on page 293
- ":FUNCTION:SCALe" on page 294

:FUNCTION:OPERation

N (see [page 794](#))

Command Syntax `:FUNCTION:OPERation <operation>`

`<operation> ::= {ADD | SUBTract | MULTIply | FFT}`

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

- ADD – Source1 + source2.
- SUBTract – Source1 - source2.
- MULTIply – Source1 * source2.
- FFT – Fast Fourier Transform on the selected waveform source.

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

Query Syntax `:FUNCTION:OPERation?`

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

Return Format `<operation><NL>`

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION:SOURce1](#)" on page 295
- "[":FUNCTION:SOURce2](#)" on page 296

:FUNCTION:RANGE

N (see [page 794](#))

Command Syntax `:FUNCTION:RANGE <range>`

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

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

Query Syntax `:FUNCTION:RANGE?`

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

Return Format `<range><NL>`

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

See Also

- "Introduction to :FUNCTION Commands" on page 280
- ":FUNCTION:SCALE" on page 294

:FUNCTION:REFerence

N (see [page 794](#))

Command Syntax `:FUNCTION:REFerence <level>`

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

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

NOTE

The FUNCTION:REFerence command is equivalent to the :FUNCTION:OFFSet command.

Query Syntax `:FUNCTION:REFerence?`

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

Return Format `<level><NL>`

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

See Also

- "Introduction to :FUNCTION Commands" on page 280
- ":FUNCTION:OFFSet" on page 290
- ":FUNCTION:RANGE" on page 292
- ":FUNCTION:SCALE" on page 294

:FUNCTION:SCALe**N** (see [page 794](#))

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

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

Query Syntax `:FUNCTION:SCALe?`

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

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

See Also

- "Introduction to :FUNCTION Commands" on page 280
- ":FUNCTION:RANGE" on page 292

:FUNCTION:SOURce1

N (see [page 794](#))

Command Syntax `:FUNCTION:SOURce1 <value>`

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

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

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

NOTE

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

Query Syntax `:FUNCTION:SOURce1?`

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

Return Format `<value><NL>`

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[:FUNCTION:OPERation](#)" on page 291
- "[:FUNCTION:GOFT:OPERation](#)" on page 287
- "[:FUNCTION:GOFT:SOURce1](#)" on page 288
- "[:FUNCTION:GOFT:SOURce2](#)" on page 289

:FUNCTION:SOURce2

N (see [page 794](#))

Command Syntax

```
:FUNCTION:SOURce2 <value>
<value> ::= {CHANnel<n> | NONE}
<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
      selection
<n> ::= {1 | 2} for 2ch models
```

The :FUNCTION:SOURce2 command specifies the second source for math operations that have two sources (see the :FUNCTION:OPERation command), in other words, ADD, SUBTract, or MULTIply. (The :FUNCTION:SOURce1 command specifies the first source.)

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

The :FUNCTION:SOURce2 setting is not used when the :FUNCTION:OPERation is FFT (Fast Fourier Transform).

Query Syntax

```
:FUNCTION:SOURce2?
```

The :FUNCTION:SOURce2? query returns the currently specified second source for math operations.

Return Format

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

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION:OPERation](#)" on page 291
- "[":FUNCTION:SOURce1](#)" on page 295

17 :HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "[Introduction to :HARDcopy Commands](#)" on page 298.

Table 67 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 299)	:HARDcopy:AREA? (see page 299)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 300)	:HARDcopy:APRinter? (see page 300)	<active_printer> ::= {<index> <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 OFF} {1 ON}} (see page 301)	:HARDcopy:FACTors? (see page 301)	{0 1}
:HARDcopy:FFEed {{0 OFF} {1 ON}} (see page 302)	:HARDcopy:FFEed? (see page 302)	{0 1}
:HARDcopy:INKSaver { {0 OFF} {1 ON}} (see page 303)	:HARDcopy:INKSaver? (see page 303)	{0 1}
:HARDcopy:LAYout <layout> (see page 304)	:HARDcopy:LAYout? (see page 304)	<layout> ::= {LANDscape PORTRait}
:HARDcopy:NETWork:ADD Ress <address> (see page 305)	:HARDcopy:NETWork:ADD Ress? (see page 305)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APP Ly (see page 306)	n/a	n/a
:HARDcopy:NETWork:DOM ain <domain> (see page 307)	:HARDcopy:NETWork:DOM ain? (see page 307)	<domain> ::= quoted ASCII string

Table 67 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:NETWork:PAS Sword <password> (see page 308)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLO T <slot> (see page 309)	:HARDcopy:NETWork:SLO T? (see page 309)	<slot> ::= {NET0 NET1}
:HARDcopy:NETWork:USE Rname <username> (see page 310)	:HARDcopy:NETWork:USE Rname? (see page 310)	<username> ::= quoted ASCII string
:HARDcopy:PAlette <palette> (see page 311)	:HARDcopy:PAlette? (see page 311)	<palette> ::= {COLOR GRAYscale NONE}
n/a	:HARDcopy:PRINTER:LIS T? (see page 312)	<list> ::= [<printer_spec>] ... [<printer_spec>] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y N} <name> ::= name of printer
:HARDcopy:STARt (see page 313)	n/a	n/a

Introduction to :HARDcopy Commands The HARDcopy subsystem provides commands to set and query the selection of hardcopy device and formatting options such as inclusion of instrument settings (FACTors) and generation of formfeed (FFEed).

:HARDC is an acceptable short form for :HARDcopy.

Reporting the Setup

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

Return Format

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

```
:HARD:APR "";AREA SCR;FACT 0;FFE 0;INKS 1;PAL NONE;LAY PORT
```

:HARDcopy:AREA

N (see [page 794](#))

Command Syntax `:HARDcopy:AREA <area>`
 `<area> ::= SCReen`

The :HARDcopy:AREA command controls what part of the display area is printed. Currently, the only legal choice is SCReen.

Query Syntax `:HARDcopy:AREA?`
The :HARDcopy:AREA? query returns the selected display area.

Return Format `<area><NL>`
 `<area> ::= SCR`

See Also

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[":HARDcopy:STARt](#)" on page 313
- "[":HARDcopy:APRinter](#)" on page 300
- "[":HARDcopy:PRINTER:LIST](#)" on page 312
- "[":HARDcopy:FACTors](#)" on page 301
- "[":HARDcopy:FFEed](#)" on page 302
- "[":HARDcopy:INKSaver](#)" on page 303
- "[":HARDcopy:LAYout](#)" on page 304
- "[":HARDcopy:PALETTE](#)" on page 311

:HARDcopy:APRinter

N (see [page 794](#))

Command Syntax `:HARDcopy:APRinter <active_printer>`

`<active_printer> ::= {<index> | <name>}`

`<index> ::= integer index of printer in list`

`<name> ::= name of printer in list`

The :HARDcopy:APRinter command sets the active printer.

Query Syntax `:HARDcopy:APRinter?`

The :HARDcopy:APRinter? query returns the name of the active printer.

Return Format `<name><NL>`

`<name> ::= name of printer in list`

See Also

- "Introduction to :HARDcopy Commands" on page 298

- ":HARDcopy:PRINTER:LIST" on page 312

- ":HARDcopy:STARt" on page 313

:HARDcopy:FACTors

N (see [page 794](#))

Command Syntax `:HARDcopy:FACTors <factors>`

`<factors> ::= {{OFF | 0} | {ON | 1}}`

The HARDcopy:FACTors command controls whether the scale factors are output on the hardcopy dump.

Query Syntax `:HARDcopy:FACTors?`

The :HARDcopy:FACTors? query returns a flag indicating whether oscilloscope instrument settings are output on the hardcopy.

Return Format `<factors><NL>`

`<factors> ::= {0 | 1}`

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 298
 - "[":HARDcopy:STARt](#)" on page 313
 - "[":HARDcopy:FFEed](#)" on page 302
 - "[":HARDcopy:INKSaver](#)" on page 303
 - "[":HARDcopy:LAYout](#)" on page 304
 - "[":HARDcopy:PAlette](#)" on page 311

:HARDcopy:FFEed**N** (see [page 794](#))**Command Syntax** `:HARDcopy:FFEed <ffeed>``<ffeed> ::= {{OFF | 0} | {ON | 1}}`

The HARDcopy:FFEed command controls whether a formfeed is output between the screen image and factors of a hardcopy dump.

Query Syntax `:HARDcopy:FFEed?`

The :HARDcopy:FFEed? query returns a flag indicating whether a formfeed is output at the end of the hardcopy dump.

Return Format `<ffeed><NL>``<ffeed> ::= {0 | 1}`**See Also**

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[:HARDcopy:STARt](#)" on page 313
- "[:HARDcopy:FACTors](#)" on page 301
- "[:HARDcopy:INKSaver](#)" on page 303
- "[:HARDcopy:LAYout](#)" on page 304
- "[:HARDcopy:PAlette](#)" on page 311

:HARDcopy:INKSaver**N** (see [page 794](#))**Command Syntax** `:HARDcopy:INKSaver <value>``<value> ::= {{OFF | 0} | {ON | 1}}`

The HARDcopy:INKSaver command controls whether the graticule colors are inverted or not.

Query Syntax `:HARDcopy:INKSaver?`

The :HARDcopy:INKSaver? query returns a flag indicating whether graticule colors are inverted or not.

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

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 298
 - "[":HARDcopy:STARt](#)" on page 313
 - "[":HARDcopy:FACTors](#)" on page 301
 - "[":HARDcopy:FFEed](#)" on page 302
 - "[":HARDcopy:LAYout](#)" on page 304
 - "[":HARDcopy:PAlette](#)" on page 311

:HARDcopy:LAYout

N (see [page 794](#))

Command Syntax `:HARDcopy:LAYout <layout>`

`<layout> ::= {LANDscape | PORTrait}`

The :HARDcopy:LAYout command sets the hardcopy layout mode.

Query Syntax `:HARDcopy:LAYout?`

The :HARDcopy:LAYout? query returns the selected hardcopy layout mode.

Return Format `<layout><NL>`

`<layout> ::= {LAND | PORT}`

See Also

- ["Introduction to :HARDcopy Commands"](#) on page 298

- [":HARDcopy:STARt"](#) on page 313

- [":HARDcopy:FACTors"](#) on page 301

- [":HARDcopy:PALETTE"](#) on page 311

- [":HARDcopy:FFEed"](#) on page 302

- [":HARDcopy:INKSaver"](#) on page 303

:HARDcopy:NETWork:ADDRess

N (see [page 794](#))

Command Syntax `:HARDcopy:NETWork:ADDRess <address>`
`<address> ::= quoted ASCII string`

The :HARDcopy:NETWork:ADDRess command sets the address for a network printer slot. The address is the server/computer name and the printer's share name in the \\server\share format.

The network printer slot is selected by the :HARDcopy:NETWork:SLOT command.

To apply the entered address, use the :HARDcopy:NETWork:APPLy command.

Query Syntax `:HARDcopy:NETWork:ADDRess?`

The :HARDcopy:NETWork:ADDRess? query returns the specified address for the currently selected network printer slot.

Return Format `<address><NL>`
`<address> ::= quoted ASCII string`

See Also

- "Introduction to :HARDcopy Commands" on page 298
- "[:HARDcopy:NETWork:SLOT](#)" on page 309
- "[:HARDcopy:NETWork:APPLy](#)" on page 306
- "[:HARDcopy:NETWork:DOMAIN](#)" on page 307
- "[:HARDcopy:NETWork:USERname](#)" on page 310
- "[:HARDcopy:NETWork:PASSword](#)" on page 308

:HARDcopy:NETWork:APPLy

N (see [page 794](#))

Command Syntax :HARDcopy:NETWork:APPLy

The :HARDcopy:NETWork:APPLy command applies the network printer settings and makes the printer connection.

See Also

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[":HARDcopy:NETWork:SLOT](#)" on page 309
- "[":HARDcopy:NETWork:ADDRess](#)" on page 305
- "[":HARDcopy:NETWork:DOMain](#)" on page 307
- "[":HARDcopy:NETWork:USERname](#)" on page 310
- "[":HARDcopy:NETWork:PASSword](#)" on page 308

:HARDcopy:NETWork:DOMain

N (see [page 794](#))

Command Syntax `:HARDcopy:NETWork:DOMain <domain>`
 `<domain> ::= quoted ASCII string`

The :HARDcopy:NETWork:DOMain command sets the Windows network domain name.

The domain name setting is a common setting for both network printer slots.

Query Syntax `:HARDcopy:NETWork:DOMain?`

The :HARDcopy:NETWork:DOMain? query returns the current Windows network domain name.

Return Format `<domain><NL>`
 `<domain> ::= quoted ASCII string`

See Also

- "Introduction to [:HARDcopy Commands](#)" on page 298
- "[:HARDcopy:NETWork:SLOT](#)" on page 309
- "[:HARDcopy:NETWork:APPLy](#)" on page 306
- "[:HARDcopy:NETWork:ADDReSS](#)" on page 305
- "[:HARDcopy:NETWork:USERname](#)" on page 310
- "[:HARDcopy:NETWork:PASStword](#)" on page 308

:HARDcopy:NETWork:PASSword

N (see [page 794](#))

Command Syntax `:HARDcopy:NETWork:PASSword <password>`

`<password>` ::= quoted ASCII string

The :HARDcopy:NETWork:PASSword command sets the password for the specified Windows network domain and user name.

The password setting is a common setting for both network printer slots.

See Also

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[:HARDcopy:NETWork:USERname](#)" on page 310
- "[:HARDcopy:NETWork:DOMain](#)" on page 307
- "[:HARDcopy:NETWork:SLOT](#)" on page 309
- "[:HARDcopy:NETWork:APPLy](#)" on page 306
- "[:HARDcopy:NETWork:ADDRess](#)" on page 305

:HARDcopy:NETWork:SLOT

N (see [page 794](#))

Command Syntax `:HARDcopy:NETWork:SLOT <slot>`
`<slot> ::= {NET0 | NET1}`

The :HARDcopy:NETWork:SLOT command selects the network printer slot used for the address and apply commands. There are two network printer slots to choose from.

Query Syntax `:HARDcopy:NETWork:SLOT?`

The :HARDcopy:NETWork:SLOT? query returns the currently selected network printer slot.

Return Format `<slot><NL>`
`<slot> ::= {NET0 | NET1}`

See Also

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[":HARDcopy:NETWork:APPLy](#)" on page 306
- "[":HARDcopy:NETWork:ADDRess](#)" on page 305
- "[":HARDcopy:NETWork:DOMain](#)" on page 307
- "[":HARDcopy:NETWork:USERname](#)" on page 310
- "[":HARDcopy:NETWork:PASSword](#)" on page 308

:HARDcopy:NETWork:USERname**N** (see [page 794](#))

Command Syntax `:HARDcopy:NETWork:USERname <username>`
 `<username> ::= quoted ASCII string`

The :HARDcopy:NETWork:USERname command sets the user name to use when connecting to the Windows network domain.

The user name setting is a common setting for both network printer slots.

Query Syntax `:HARDcopy:NETWork:USERname?`

The :HARDcopy:NETWork:USERname? query returns the currently set user name.

Return Format `<username><NL>`
 `<username> ::= quoted ASCII string`

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 298
 - "[":HARDcopy:NETWork:DOMain](#)" on page 307
 - "[":HARDcopy:NETWork:PASSword](#)" on page 308
 - "[":HARDcopy:NETWork:SLOT](#)" on page 309
 - "[":HARDcopy:NETWork:APPLy](#)" on page 306
 - "[":HARDcopy:NETWork:ADDRess](#)" on page 305

:HARDcopy:PALETTE

N (see [page 794](#))

Command Syntax :HARDcopy:PALETTE <palette>

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

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

The oscilloscope's print driver cannot print color images to color laser printers, so the COLOR option is not available when connected to laser printers.

Query Syntax :HARDcopy:PALETTE?

The :HARDcopy:PALETTE? query returns the selected hardcopy palette color.

Return Format <palette><NL>

<palette> ::= {COL | GRAY | NONE}

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[:HARDcopy:STARt](#)" on page 313
- "[:HARDcopy:FACTors](#)" on page 301
- "[:HARDcopy:LAYout](#)" on page 304
- "[:HARDcopy:FFeed](#)" on page 302
- "[:HARDcopy:INKSaver](#)" on page 303

:HARDcopy:PRINter:LIST

N (see [page 794](#))

Query Syntax :HARDcopy:PRINter:LIST?

The :HARDcopy:PRINter:LIST? query returns a list of available printers. The list can be empty.

Return Format

```
<list><NL>
<list> ::= [<printer_spec>] ... [<printer_spec>]
<printer_spec> ::= "<index>,<active>,<name>;"
<index> ::= integer index of printer
<active> ::= {Y | N}
<name> ::= name of printer (for example "DESKJET 950C")
```

See Also

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[":HARDcopy:APRinter](#)" on page 300
- "[":HARDcopy:STARt](#)" on page 313

:HARDcopy:STARt

N (see [page 794](#))

Command Syntax :HARDcopy:STARt

The :HARDcopy:STARt command starts a print job.

See Also

- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[:HARDcopy:APRINTER](#)" on page 300
- "[:HARDcopy:PRINTER:LIST](#)" on page 312
- "[:HARDcopy:FACTors](#)" on page 301
- "[:HARDcopy:FFEed](#)" on page 302
- "[:HARDcopy:INKSaver](#)" on page 303
- "[:HARDcopy:LAYOUT](#)" on page 304
- "[:HARDcopy:PALETTE](#)" on page 311

18 :LISTer Commands

Table 68 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 316)	<binary_block> ::= comma-separated data with newlines at the end of each row
:LISTer:DISPlay {{OFF 0} {SBUS1 ON 1} ALL} (see page 317)	:LISTer:DISPlay? (see page 317)	{OFF SBUS1 ALL}
:LISTer:REFerence <time_ref> (see page 318)	:LISTer:REFERENCE? (see page 318)	<time_ref> ::= {TRIGger PREVIOUS}

Introduction to :LISTer Commands The LISTER subsystem is used to turn on/off the serial decode Lister display and return data from the Lister display.

:LISTer:DATA

N (see [page 794](#))

Query Syntax :LISTer:DATA?

The :LISTer:DATA? query returns the lister data.

Return Format <binary block><NL>

<binary_block> ::= comma-separated data with newlines at the
end of each row

See Also

- "[Introduction to :LISTer Commands](#)" on page 315
- "[":LISTer:DISPLAY](#)" on page 317
- "[Definite-Length Block Response Data](#)" on page 117

:LISTer:DISPlay

N (see [page 794](#))

Command Syntax `:LISTer:DISPLAY <value>`

`<value> ::= {{OFF | 0} | {SBUS1 | ON | 1} | ALL}`

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 Syntax `:LISTer:DISPLAY?`

The :LISTer:DISPLAY? query returns the Lister display setting.

Return Format `<value><NL>`

`<value> ::= {OFF | SBUS1 | ALL}`

See Also

- "[Introduction to :LISTer Commands](#)" on page 315
- "[":SBUS<n>:DISPLAY](#)" on page 452
- "[":LISTer:DATA](#)" on page 316

:LISTER:REFerence**N** (see [page 794](#))

Command Syntax `:LISTER:REFerence <time_ref>`
`<time_ref> ::= {TRIGger | PREVIOUS}`

The :LISTER:REFerence command selects whether the time value for a Lister row is relative to the trigger or the previous Lister row.

Query Syntax `:LISTER:REFerence?`

The :LISTER:REFerence? query returns the Lister time reference setting.

Return Format `<time_ref><NL>`
`<time_ref> ::= {TRIGger | PREVIOUS}`

See Also

- "[Introduction to :LISTER Commands](#)" on page 315
- "[":SBUS<n>:DISPLAY](#)" on page 452
- "[":LISTER:DATA](#)" on page 316
- "[":LISTER:DISPLAY](#)" on page 317

19 :MARKer Commands

Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). See "[Introduction to :MARKer Commands](#)" on page 320.

Table 69 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 321)	:MARKer:MODE? (see page 321)	<mode> ::= {OFF MEASurement MANual WAVEform}
:MARKer:X1Position <position>[suffix] (see page 322)	:MARKer:X1Position? (see page 322)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 323)	:MARKer:X1Y1source? (see page 323)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 324)	:MARKer:X2Position? (see page 324)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 325)	:MARKer:X2Y2source? (see page 325)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 326)	<return_value> ::= X cursors delta value in NR3 format

Table 69 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:XUnits <mode> (see page 327)	:MARKer:XUnits? (see page 327)	<units> ::= {SEConds HERTZ DEGRees PERCent}
:MARKer:XUnits:USE (see page 328)	n/a	n/a
:MARKer:Y1Position <position>[suffix] (see page 329)	:MARKer:Y1Position? (see page 329)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position>[suffix] (see page 330)	:MARKer:Y2Position? (see page 330)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 331)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUnits <mode> (see page 332)	:MARKer:YUnits? (see page 332)	<units> ::= {BASE PERCent}
:MARKer:YUnits:USE (see page 333)	n/a	n/a

Introduction to :MARKer Commands The MARKer subsystem commands set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). You can set and query the marker mode and source, the position of the X and Y cursors, and query delta X and delta Y cursor values.

Reporting the Setup

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

Return Format

The following is a sample response from the :MARKer? query. In this case, the query was issued following a *RST and ":MARKer:MODE MANual" command.

```
:MARK:X1Y1 CHAN1;X2Y2 CHAN1;MODE MAN
```

:MARKer:MODE

N (see [page 794](#))

Command Syntax `:MARKer:MODE <mode>`

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

The :MARKer:MODE command sets the cursors mode:

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

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

- MEASurement – cursors track the most recent measurement.

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

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

Query Syntax `:MARKer:MODE?`

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

Return Format `<mode><NL>`

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

See Also ["Introduction to :MARKer Commands" on page 320](#)

- [":MARKer:X1Y1source" on page 323](#)
- [":MARKer:X2Y2source" on page 325](#)
- [":MEASure:SOURce" on page 363](#)
- [":MARKer:X1Position" on page 322](#)
- [":MARKer:X2Position" on page 324](#)
- [":MARKer:Y1Position" on page 329](#)
- [":MARKer:Y2Position" on page 330](#)

:MARKer:X1Position

N (see [page 794](#))

Command Syntax `:MARKer:X1Position <position> [suffix]`

`<position> ::= X1 cursor position in NR3 format`

`<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}`

The :MARKer:X1Position command:

- Sets :MARKer:MODE to MANual if it is not currently set to WAVerform (see "[:MARKer:MODE](#)" on page 321).
- Sets the X1 cursor position to the specified value.

X cursor units are set by the :MARKer:XUNits command.

Query Syntax `:MARKer:X1Position?`

The :MARKer:X1Position? query returns the current X1 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

NOTE

If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVerform to put the cursors in the front-panel Normal mode.

Return Format `<position><NL>`

`<position> ::= X1 cursor position in NR3 format`

See Also ["Introduction to :MARKer Commands"](#) on page 320

- "[:MARKer:MODE](#)" on page 321
- "[:MARKer:X2Position](#)" on page 324
- "[:MARKer:X1Y1source](#)" on page 323
- "[:MARKer:X2Y2source](#)" on page 325
- "[:MARKer:XUNits](#)" on page 327
- "[:MEASure:TSTArt](#)" on page 731

:MARKer:X1Y1source

N (see [page 794](#))

Command Syntax :MARKer:X1Y1source <source>

```
<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= {1 | 2}
```

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVEform (see "[:MARKer:MODE](#)" on page 321):

- Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X1Y1) sets the source for the other (for example, X2Y2).

If the marker mode is currently WAVEform, the X1Y1 source can be set separate from the X2Y2 source.

If :MARKer:MODE is set to OFF or MANUAL, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, MATH, or WMEMory<r> will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

NOTE

MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

:MARKer:X1Y1source?

The :MARKer:X1Y1source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>
<source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}
```

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[:MARKer:MODE](#)" on page 321
- "[:MARKer:X2Y2source](#)" on page 325
- "[:MEASure:SOURce](#)" on page 363

:MARKer:X2Position

N (see [page 794](#))

Command Syntax `:MARKer:X2Position <position> [suffix]`

`<position> ::= X2 cursor position in NR3 format`

`<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}`

The :MARKer:X2Position command:

- Sets :MARKer:MODE to MANual if it is not currently set to WAVerform (see "[:MARKer:MODE](#)" on page 321).
- Sets the X2 cursor position to the specified value.

X cursor units are set by the :MARKer:XUNits command.

Query Syntax `:MARKer:X2Position?`

The :MARKer:X2Position? query returns current X2 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTOP command/query.

NOTE

If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVerform to put the cursors in the front-panel Normal mode.

Return Format `<position><NL>`

`<position> ::= X2 cursor position in NR3 format`

See Also ["Introduction to :MARKer Commands"](#) on page 320

- "[:MARKer:MODE](#)" on page 321
- "[:MARKer:X1Position](#)" on page 322
- "[:MARKer:X2Y2source](#)" on page 325
- "[:MARKer:XUNits](#)" on page 327
- "[:MEASure:TSTOP](#)" on page 732

:MARKer:X2Y2source

N (see [page 794](#))

Command Syntax

```
:MARKer:X2Y2source <source>
<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= {1 | 2}
```

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVEform (see "[:MARKer:MODE](#)" on page 321):

- Sending a :MARKer:X2Y2source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X2Y2) sets the source for the other (for example, X1Y1).

If the marker mode is currently WAVEform, the X2Y2 source can be set separate from the X1Y1 source.

If :MARKer:MODE is set to OFF or MANUAL, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, MATH, or WMEMory<r> will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

NOTE

MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

```
:MARKer:X2Y2source?
```

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>
<source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}
```

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[:MARKer:MODE](#)" on page 321
- "[:MARKer:X1Y1source](#)" on page 323
- "[:MEASure:SOURce](#)" on page 363

:MARKer:XDELta

N (see [page 794](#))

Query Syntax `:MARKer:XDELta?`

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

Xdelta = (Value at X2 cursor) - (Value at X1 cursor)

X cursor units are set by the :MARKer:XUNits command.

NOTE

If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAveform to put the cursors in the front-panel Normal mode.

Return Format `<value><NL>`

`<value>` ::= difference value in NR3 format.

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[":MARKer:MODE](#)" on page 321
- "[":MARKer:X1Position](#)" on page 322
- "[":MARKer:X2Position](#)" on page 324
- "[":MARKer:X1Y1source](#)" on page 323
- "[":MARKer:X2Y2source](#)" on page 325
- "[":MARKer:XUNits](#)" on page 327

:MARKer:XUNits

N (see [page 794](#))

Command Syntax `:MARKer:XUNits <units>`

```
<units> ::= {SEConds | HERTz | DEGRees | PERCent}
```

The :MARKer:XUNits command sets the X cursors units:

- SEConds – for making time measurements.
- HERTz – for making frequency measurements.
- DEGRees – for making phase measurements. Use the :MARKer:XUNits:USE command to set the current X1 location as 0 degrees and the current X2 location as 360 degrees.
- PERCent – for making ratio measurements. Use the :MARKer:XUNits:USE command to set the current X1 location as 0 percent and the current X2 location as 100 percent.

Changing X units affects the input and output values of the :MARKer:X1Position, :MARKer:X2Position, and :MARKer:XDELta commands/queries.

Query Syntax `:MARKer:XUNits?`

The :MARKer:XUNits? query returns the current X cursors units.

Return Format `<units><NL>`

```
<units> ::= {SEC | HERT | DEGR | PERC}
```

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 320
 - "[:MARKer:XUNits:USE](#)" on page 328
 - "[:MARKer:X1Y1source](#)" on page 323
 - "[:MARKer:X2Y2source](#)" on page 325
 - "[:MEASure:SOURce](#)" on page 363
 - "[:MARKer:X1Position](#)" on page 322
 - "[:MARKer:X2Position](#)" on page 324

:MARKer:XUNits:USE

N (see [page 794](#))

Command Syntax `:MARKer:XUNits:USE`

When DEGRees is selected for :MARKer:XUNits, the :MARKer:XUNits:USE command sets the current X1 location as 0 degrees and the current X2 location as 360 degrees.

When PERCent is selected for :MARKer:XUNits, the :MARKer:XUNits:USE command sets the current X1 location as 0 percent and the current X2 location as 100 percent.

Once the 0 and 360 degree or 0 and 100 percent locations are set, inputs to and outputs from the :MARKer:X1Position, :MARKer:X2Position, and :MARKer:XDELta commands/queries are relative to the set locations.

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 320
 - [":MARKer:XUNits"](#) on page 327
 - [":MARKer:X1Y1source"](#) on page 323
 - [":MARKer:X2Y2source"](#) on page 325
 - [":MEASure:SOURce"](#) on page 363
 - [":MARKer:X1Position"](#) on page 322
 - [":MARKer:X2Position"](#) on page 324
 - [":MARKer:XDELta"](#) on page 326

:MARKer:Y1Position

N (see [page 794](#))

Command Syntax :MARKer:Y1Position <position> [suffix]

<position> ::= Y1 cursor position in NR3 format

<suffix> ::= {mV | v | dB}

If the :MARKer:MODE is not currently set to WAVEform (see "[:MARKer:MODE](#)" on page 321), the :MARKer:Y1Position command:

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

Y cursor units are set by the :MARKer:YUNits command.

When the :MARKer:MODE is set to WAVEform, Y positions cannot be set.

Query Syntax :MARKer:Y1Position?

The :MARKer:Y1Position? query returns current Y1 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTARt command/query.

NOTE

If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVEform to put the cursors in the front-panel Normal mode.

Return Format <position><NL>

<position> ::= Y1 cursor position in NR3 format

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[:MARKer:MODE](#)" on page 321
- "[:MARKer:X1Y1source](#)" on page 323
- "[:MARKer:X2Y2source](#)" on page 325
- "[:MARKer:Y2Position](#)" on page 330
- "[:MARKer:YUNits](#)" on page 332
- "[:MEASure:VSTARt](#)" on page 736

:MARKer:Y2Position

N (see [page 794](#))

Command Syntax `:MARKer:Y2Position <position> [<suffix>]`

`<position> ::= Y2 cursor position in NR3 format`

`<suffix> ::= {mV | v | dB}`

If the :MARKer:MODE is not currently set to WAVEform (see "[:MARKer:MODE](#)" on page 321), the :MARKer:Y1Position command:

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

Y cursor units are set by the :MARKer:YUNits command.

When the :MARKer:MODE is set to WAVEform, Y positions cannot be set.

Query Syntax `:MARKer:Y2Position?`

The :MARKer:Y2Position? query returns current Y2 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

NOTE

If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVEform to put the cursors in the front-panel Normal mode.

Return Format `<position><NL>`

`<position> ::= Y2 cursor position in NR3 format`

See Also ["Introduction to :MARKer Commands"](#) on page 320

- [":MARKer:MODE"](#) on page 321
- [":MARKer:X1Y1source"](#) on page 323
- [":MARKer:X2Y2source"](#) on page 325
- [":MARKer:Y1Position"](#) on page 329
- [":MARKer:YUNits"](#) on page 332
- [":MEASure:VSTOp"](#) on page 737

:MARKer:YDELta

N (see [page 794](#))

Query Syntax `:MARKer:YDELta?`

The `:MARKer:YDELta?` query returns the value difference between the current Y1 and Y2 cursor positions.

$Y_{delta} = (\text{Value at Y2 cursor}) - (\text{Value at Y1 cursor})$

NOTE

If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set `:MARKer:MODE` to MANual or WAVEform to put the cursors in the front-panel Normal mode.

Y cursor units are set by the `:MARKer:YUNits` command.

Return Format `<value><NL>`

`<value>` ::= difference value in NR3 format

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[":MARKer:MODE](#)" on page 321
- "[":MARKer:X1Y1source](#)" on page 323
- "[":MARKer:X2Y2source](#)" on page 325
- "[":MARKer:Y1Position](#)" on page 329
- "[":MARKer:Y2Position](#)" on page 330
- "[":MARKer:YUNits](#)" on page 332

:MARKer:YUNits

N (see [page 794](#))

Command Syntax `:MARKer:YUNits <units>`

`<units> ::= {BASE | PERCent}`

The :MARKer:YUNits command sets the Y cursors units:

- BASE – for making measurements in the units associated with the cursors source.
- PERCent – for making ratio measurements. Use the :MARKer:YUNits:USE command to set the current Y1 location as 0 percent and the current Y2 location as 100 percent.

Changing Y units affects the input and output values of the :MARKer:Y1Position, :MARKer:Y2Position, and :MARKer:YDELta commands/queries.

Query Syntax `:MARKer:YUNits?`

The :MARKer:YUNits? query returns the current Y cursors units.

Return Format `<units><NL>`

`<units> ::= {BASE | PERC}`

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 320
 - "[":MARKer:YUNits:USE](#)" on page 333
 - "[":MARKer:X1Y1source](#)" on page 323
 - "[":MARKer:X2Y2source](#)" on page 325
 - "[":MEASure:SOURce](#)" on page 363
 - "[":MARKer:Y1Position](#)" on page 329
 - "[":MARKer:Y2Position](#)" on page 330
 - "[":MARKer:YDELta](#)" on page 331

:MARKer:YUNits:USE

N (see [page 794](#))

Command Syntax :MARKer:YUNits:USE

When PERCent is selected for :MARKer:YUNits, the :MARKer:YUNits:USE command sets the current Y1 location as 0 percent and the current Y2 location as 100 percent.

Once the 0 and 100 percent locations are set, inputs to and outputs from the :MARKer:Y1Position, :MARKer:Y2Position, and :MARKer:YDELta commands/queries are relative to the set locations.

See Also • ["Introduction to :MARKer Commands"](#) on page 320

- [":MARKer:YUNits"](#) on page 332
- [":MARKer:X1Y1source"](#) on page 323
- [":MARKer:X2Y2source"](#) on page 325
- [":MEASure:SOURce"](#) on page 363
- [":MARKer:Y1Position"](#) on page 329
- [":MARKer:Y2Position"](#) on page 330
- [":MARKer:YDELta"](#) on page 331

20 :MEASure Commands

Select automatic measurements to be made and control time markers. See "[Introduction to :MEASure Commands](#)" on page 342.

Table 70 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 344)	n/a	n/a
:MEASure:CLEar (see page 345)	n/a	n/a
:MEASure:DEFine DELay, <delay spec>[,<source>] (see page 346)	:MEASure:DEFine? DELay[,<source>] (see page 347)	<delay spec> ::= <edge_spec1>, <edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ -} <occurrence> ::= integer <source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>}
:MEASure:DEFine THresholds, <threshold spec>[,<source>] (see page 346)	:MEASure:DEFine? THresholds[,<source>] (see page 347)	<threshold spec> ::= {STANDARD} {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCent ABSolute} <source> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>}
:MEASure:DElay [<source1>] [,<source2>] (see page 349)	:MEASure:DElay? [<source1>] [,<source2>] (see page 349)	<source1,2> ::= {CHANnel<n> FUNCTION MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 351)	:MEASure:DUTYcycle? [<source>] (see page 351)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= ratio of positive pulse width to period in NR3 format</p>
:MEASure:FALLtime [<source>] (see page 352)	:MEASure:FALLtime? [<source>] (see page 352)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= time in seconds between the lower and upper thresholds in NR3 format</p>
:MEASure:FREQuency [<source>] (see page 353)	:MEASure:FREQuency? [<source>] (see page 353)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= frequency in Hertz in NR3 format</p>

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NWIDth [<source>] (see page 354)	:MEASure:NWIDth? [<source>] (see page 354)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= negative pulse width in seconds-NR3 format</p>
:MEASure:OVERshoot [<source>] (see page 355)	:MEASure:OVERshoot? [<source>] (see page 355)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><return_value> ::= the percent of the overshoot of the selected waveform in NR3 format</p>
:MEASure:PERiod [<source>] (see page 357)	:MEASure:PERiod? [<source>] (see page 357)	<p><source> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= waveform period in seconds in NR3 format</p>
:MEASure:PHASE [<source1>] [,<source2>] (see page 358)	:MEASure:PHASE? [<source1>] [,<source2>] (see page 358)	<p><source1,2> ::= {CHANnel<n> FUNCTION MATH WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><return_value> ::= the phase angle value in degrees in NR3 format</p>

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PREShoot [<source>] (see page 359)	:MEASure:PREShoot? [<source>] (see page 359)	<source> ::= {CHANnel<n> FUNCTION MATH WMEMORY<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 360)	:MEASure:PWIDth? [<source>] (see page 360)	<source> ::= {CHANnel<n> FUNCTION MATH WMEMORY<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMORY<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISetime [<source>] (see page 361)	:MEASure:RISetime? [<source>] (see page 361)	<source> ::= {CHANnel<n> FUNCTION MATH WMEMORY<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SHOW {1 ON} (see page 362)	:MEASure:SHOW? (see page 362)	{1}

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SOURce <source1> [,<source2>] (see page 363)	:MEASure:SOURce? (see page 363)	<p><source1,2> ::= {CHANnel<n> FUNCTION MATH WMEMORY<r> EXTERNAL} for DSO models</p> <p><source1,2> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMORY<r> EXTERNAL} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= {<source> NONE}</p>
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see page 365)	<p><slope> ::= direction of the waveform</p> <p><occurrence> ::= the transition to be reported</p> <p><source> ::= {CHANnel<n> FUNCTION MATH WMEMORY<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION MATH WMEMORY<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1-2 in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= time in seconds of the specified transition</p>

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TValue? <value>, [<slope>]<occurrence> [,<source>] (see page 367)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion MATH WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMplitude [<source>] (see page 369)	:MEASure:VAMplitude? [<source>] (see page 369)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<interval>] [,][<sour ce>] (see page 370)	:MEASure:VAverage? [<interval>] [,][<sour ce>] (see page 370)	<interval> ::= {CYCLE DISPLAY} <source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 371)	:MEASure:VBASe? [<source>] (see page 371)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 372)	:MEASure:VMAX? [<source>] (see page 372)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 373)	:MEASure:VMIN? [<source>] (see page 373)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 374)	:MEASure:VPP? [<source>] (see page 374)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<interval>][,] [<type>][,] [<source>] (see page 375)	:MEASure:VRMS? [<interval>][,] [<type>][,] [<source>] (see page 375)	<interval> ::= {CYCLE DISPLAY} <type> ::= {AC DC} <source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format

Table 70 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:VTIMe? <vtim>[,<source>] (see page 376)	<vtim> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCTion MATH WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format
:MEASure:VTOP [<source>] (see page 377)	:MEASure:VTOP? [<source>] (see page 377)	<source> ::= {CHANnel<n> FUNCTion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDOW <type> (see page 378)	:MEASure:WINDOW? (see page 378)	<type> ::= {MAIN ZOOM AUTO}

Introduction to :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

Measurement Error

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the zoomed (delayed) time base mode (:TIMEbase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the zoomed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNCtion source.

Not all measurements are available on the digital channels or FFT (Fast Fourier Transform).

Reporting the Setup

Use the :MEASure? query to obtain setup information for the MEASure subsystem. (Currently, this is only :MEASure:SOURce.)

Return Format

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

```
:MEAS:SOUR CHAN1,CHAN2;STAT ON
```

:MEASure:ALL

N (see [page 794](#))

Command Syntax :MEASure:ALL

This command installs a Snapshot All measurement on the screen.

See Also • ["Introduction to :MEASure Commands"](#) on page 342

:MEASure:CLEar

N (see [page 794](#))

Command Syntax :MEASure:CLEar

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

See Also · ["Introduction to :MEASure Commands"](#) on page 342

:MEASure:DEFine

N (see [page 794](#))

Command Syntax `:MEASure:DEFine <meas_spec>[,<source>]`

```
<meas_spec> ::= {DEDelay | THRESHolds}
<source> ::= {CHANnel<n> | FUNCTion | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= {1 | 2}
```

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DEDelay specification or the THRESHolds values. For example, changing the THRESHolds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

MEASure Command	DEDelay	THRESHolds
DUTYcycle		x
DEDelay	x	x
FALLtime		x
FREQuency		x
NWIDth		x
OVERshoot		x
PERiod		x
PHASe		x
PRESHoot		x
PWIDth		x
RISetime		x
VAVerage		x
VRMS		x

**:MEASure:DEFine
DEDelay Command
Syntax**

```
:MEASure:DEFine DELay,<delay spec>[,<source>]
<delay spec> ::= <edge_spec1>,<edge_spec2>
<edge_spec1> ::= [<slope>]<occurrence>
<edge_spec2> ::= [<slope>]<occurrence>
<slope> ::= {+ | -}
```

```
<occurrence> ::= integer
```

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

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

NOTE

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

:MEASure:DEFine THresholds Command Syntax

```
:MEASure:DEFine THresholds,<threshold spec>[,<source>]
<threshold spec> ::= {STANDARD
                      | {<threshold mode>,<upper>,<middle>,<lower>}}
<threshold mode> ::= {PERCENT | ABSOLUTE}
for <threshold mode> = PERCENT:
```

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

for <threshold mode> = ABSOLUTE:

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

- STANDARD threshold specification sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.
- Threshold mode PERCENT sets the measurement thresholds to any user-defined percentages between 5% and 95% of values between Vbase and Vtop.
- Threshold mode ABSOLUTE sets the measurement thresholds to absolute values. ABSOLUTE thresholds are dependent on channel scaling (:CHANnel<n>:RANGE or "[:CHANnel<n>:SCALe](#)" on page 233:CHANnel<n>:SCALE), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITS). Always set these values first before setting ABSOLUTE thresholds.

Query Syntax

```
:MEASure:DEFine? <meas_spec>[,<source>]
<meas_spec> ::= {DElay | THresholds}
```

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

Return Format

for <meas_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
for <meas_spec> = THResholds and <threshold mode> = PERCent:
```

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

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

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

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

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

for <threshold spec> = STANdard:

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

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:DELay](#)" on page 349
- "[":MEASure:SOURce](#)" on page 363
- "[":CHANnel<n>:RANGE](#)" on page 232
- "[":CHANnel<n>:SCALe](#)" on page 233
- "[":CHANnel<n>:PROBe](#)" on page 226
- "[":CHANnel<n>:UNITs](#)" on page 234

:MEASure:DElay

N (see [page 794](#))

Command Syntax

```
:MEASure:DElay [<source1> [,<source2>]
<source1>, <source2> ::= {CHANnel<n> | FUNCTion | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

The :MEASure:DElay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

$$\text{delay} = t(\text{edge spec 2}) - t(\text{edge spec 1})$$

where the <edge spec> definitions are set by the :MEASure:DEFine command

NOTE

The :MEASure:DElay command and the front-panel delay measurement differ from the :MEASure:DElay? query.

The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, you can select the edge polarity, but the instrument will select the edges that will make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DElay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

Query Syntax

```
:MEASure:DElay? [<source1> [,<source2>]
```

The :MEASure:DElay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Also in the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. *It is the middle threshold value that is used when performing the delay query.* The standard upper, middle, and lower measurement thresholds are 90%, 50%, and 10% values between Vbase and Vtop. If you want to move the delay measurement point nearer to Vtop or Vbase, you must change the threshold values with the :MEASure:DEFine THresholds command.

Return Format	<code><value><NL></code>
	<code><value></code> ::= floating-point number delay time in seconds in NR3 format
See Also	<ul style="list-style-type: none">· "Introduction to :MEASure Commands" on page 342· ":MEASure:DEFine" on page 346· ":MEASure:PHASe" on page 358

:MEASure:DUTYcycle

C (see page 794)

Command Syntax `:MEASure:DUTYcycle [<source>]`

```
<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH
              | WMEMemory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

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

NOTE

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

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

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

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

Return Format `<value><NL>`

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

See Also

- "Introduction to :MEASure Commands" on page 342
- ":MEASure:PERiod" on page 357
- ":MEASure:PWIDth" on page 360
- ":MEASure:SOURce" on page 363

Example Code `· "Example Code" on page 364`

:MEASure:FALLtime

 (see [page 794](#))

Command Syntax

```
:MEASure:FALLtime [<source>]
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

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

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

$$\text{fall time} = \text{time at lower threshold} - \text{time at upper threshold}$$

Return Format

```
<value><NL>
<value> ::= time in seconds between the lower threshold and upper
           threshold in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:RISetime](#)" on page 361
- "[":MEASure:SOURce](#)" on page 363

:MEASure:FREQuency

C (see [page 794](#))

Command Syntax :MEASure:FREQuency [<source>]

```
<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH
              | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

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

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

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

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

Return Format <source><NL>

```
<source> ::= frequency in Hertz in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:SOURce](#)" on page 363
- "[":MEASure:PERiod](#)" on page 357

Example Code

- "[":Example Code](#)" on page 364

:MEASure:NWIDth

 (see [page 794](#))

Command Syntax

```
:MEASure:NWIDth [<source>]
<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH
              | WMEMory<r>}
<digital channels> ::= DIGItal<d> for the MSO models
<n> ::= 1 to (# of analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is not specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

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

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

width = (time at trailing rising edge - time at leading falling edge)

Return Format

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

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MEASure:SOURce](#)" on page 363
- "[:MEASure:PWIDth](#)" on page 360
- "[:MEASure:PERiod](#)" on page 357

:MEASure:OVERshoot

C (see [page 794](#))

Command Syntax `:MEASure:OVERshoot [<source>]`

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

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

`<r> ::= 1-2 in NR1 format`

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax `:MEASure:OVERshoot? [<source>]`

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{overshoot} = ((\text{Vmax}-\text{Vtop}) / (\text{Vtop}-\text{Vbase})) \times 100$$

For a falling edge:

$$\text{overshoot} = ((\text{Vbase}-\text{Vmin}) / (\text{Vtop}-\text{Vbase})) \times 100$$

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right after the chosen edge, halfway to the next edge. This more restricted definition is used instead of the normal one, because it is conceivable that a signal may have more preshoot than overshoot, and the normal extremum would then be dominated by the preshoot of the following edge.

Return Format `<overshoot><NL>`

`<overshoot> ::= the percent of the overshoot of the selected waveform in NR3 format`

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MEASure:PRESHoot](#)" on page 359
- "[:MEASure:SOURce](#)" on page 363
- "[:MEASure:VMAX](#)" on page 372

- [":MEASure:VTOP" on page 377](#)
- [":MEASure:VBASe" on page 371](#)
- [":MEASure:VMIN" on page 373](#)

:MEASure:PERiod

C (see [page 794](#))

Command Syntax

```
:MEASure:PERiod [<source>]
<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH
              | WMEMemory<r>}
<digital channels> ::= DIGItal<d> for the MSO models
<n> ::= 1 to (# of analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax	<code>:MEASure:PERiod? [<source>]</code>
	The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.
	IF the edge closest to the trigger reference on screen is rising:
	THEN period = (time at trailing rising edge - time at leading rising edge)
	ELSE period = (time at trailing falling edge - time at leading falling edge)
Return Format	<code><value><NL></code>
	<code><value> ::= waveform period in seconds in NR3 format</code>
See Also	<ul style="list-style-type: none"> · "Introduction to :MEASure Commands" on page 342 · ":MEASure:SOURce" on page 363 · ":MEASure:NWIDth" on page 354 · ":MEASure:PWIDth" on page 360 · ":MEASure:FREQuency" on page 353
Example Code	<ul style="list-style-type: none"> · "Example Code" on page 364

:MEASure:PHASe

N (see [page 794](#))

Command Syntax

```
:MEASure:PHASe [<source1> [,<source2>]
    <source1>, <source2> ::= {CHANnel<n> | FUNCTion | MATH | WMEMORY<r>}
    <n> ::= 1 to (# analog channels) in NR1 format
    <r> ::= 1-2 in NR1 format
```

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

Query Syntax

```
:MEASure:PHASe? [<source1> [,<source2>]
```

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DElay for more detail on selecting the 2nd edge.

The phase is calculated as follows:

$$\text{phase} = (\text{delay} / \text{period of input 1}) \times 360$$

Return Format

```
<value><NL>
<value> ::= the phase angle value in degrees in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:DElay](#)" on page 349
- "[":MEASure:PERiod](#)" on page 357
- "[":MEASure:SOURce](#)" on page 363

:MEASure:PREShoot



(see [page 794](#))

Command Syntax :MEASure:PREShoot [<source>]

```
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

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

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

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{preshoot} = ((V_{\text{min}} - V_{\text{base}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

For a falling edge:

$$\text{preshoot} = ((V_{\text{max}} - V_{\text{top}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right before the chosen edge, halfway back to the previous edge. This more restricted definition is used instead of the normal one, because it is likely that a signal may have more overshoot than preshoot, and the normal extremum would then be dominated by the overshoot of the preceding edge.

Return Format <value><NL>

```
<value> ::= the percent of preshoot of the selected waveform
           in NR3 format
```

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 342
 - "[:MEASure:SOURce](#)" on page 363
 - "[:MEASure:VMIN](#)" on page 373
 - "[:MEASure:VMAX](#)" on page 372
 - "[:MEASure:VTOP](#)" on page 377
 - "[:MEASure:VBASE](#)" on page 371

:MEASure:PWIDth

 (see [page 794](#))

Command Syntax

```
:MEASure:PWIDth [<source>]
<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH
              | WMEMory<r>}
<digital channels> ::= DIGItal<d> for the MSO models
<n> ::= 1 to (# of analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

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

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:

THEN width = (time at trailing falling edge - time at leading rising edge)

ELSE width = (time at leading falling edge - time at leading rising edge)

Return Format

```
<value><NL>
<value> ::= width of positive pulse in seconds in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MEASure:SOURce](#)" on page 363
- "[:MEASure:NWIDth](#)" on page 354
- "[:MEASure:PERiod](#)" on page 357

:MEASure:RISetime

C (see [page 794](#))

Command Syntax `:MEASure: RISetime [<source>]`

```
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax `:MEASure: RISetime? [<source>]`

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

$$\text{rise time} = \text{time at upper threshold} - \text{time at lower threshold}$$

Return Format `<value><NL>`

```
<value> ::= rise time in seconds in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342

- "[":MEASure:SOURce](#)" on page 363

- "[":MEASure:FALLtime](#)" on page 352

:MEASure:SHOW

N (see [page 794](#))

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

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

Query Syntax `:MEASure:SHOW?`

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

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

See Also • "Introduction to :MEASure Commands" on page 342

:MEASure:SOURce

C (see [page 794](#))

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]
<source1>,<source2> ::= {<digital channels> | CHANnel<n> | FUNCtion
                           | MATH | WMEMory<r> | EXTernal}
<digital channels> ::= DIGItal<d> for the MSO models
<n> ::= 1 to (# of analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command.

If a source is specified for any measurement, the current source is changed to this new value.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

EXTernal is only a valid source for the counter measurement (and <source1>).

Query Syntax

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 only applies to :MEASure:DELay and :MEASure:PHASe measurements.

NOTE

MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Return Format

```
<source1>,<source2><NL>
<source1>,<source2> ::= {<digital channels> | CHAN<n> | FUNC | WMWM<r>
                           | EXT | NONE}
```

See Also:

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MARKer:MODE](#)" on page 321
- "[":MARKer:X1Y1source](#)" on page 323
- "[":MARKer:X2Y2source](#)" on page 325
- "[":MEASure:DELay](#)" on page 349
- "[":MEASure:PHASe](#)" on page 358

Example Code

```

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"      ' Source to measure.
myScope.WriteString ":MEASURE:FREQUENCY?"      ' Query for frequency.
varQueryResult = myScope.ReadNumber      ' Read frequency.
MsgBox "Frequency:" + vbCrLf _
+ FormatNumber(varQueryResult / 1000, 4) + " kHz"
myScope.WriteString ":MEASURE:DUTYCYCLE?"      ' Query for duty cycle.
varQueryResult = myScope.ReadNumber      ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf _
+ FormatNumber(varQueryResult, 3) + "%"
myScope.WriteString ":MEASURE:RISETIME?"      ' Query for risetime.
varQueryResult = myScope.ReadNumber      ' Read risetime.
MsgBox "Risetime:" + vbCrLf _
+ FormatNumber(varQueryResult * 1000000, 4) + " us"
myScope.WriteString ":MEASURE:VPP?"      ' Query for Pk to Pk voltage.
varQueryResult = myScope.ReadNumber      ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf _
+ FormatNumber(varQueryResult, 4) + " V"
myScope.WriteString ":MEASURE:VMAX?"      ' Query for Vmax.
varQueryResult = myScope.ReadNumber      ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf _
+ FormatNumber(varQueryResult, 4) + " V"

```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:MEASure:TEDGE

N (see [page 794](#))

Query Syntax

```
:MEASure:TEDGE? <slope><occurrence>[,<source>]

<slope> ::= direction of the waveform. A rising slope is indicated by a
           space or plus sign (+). A falling edge is indicated by a
           minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number
                  is one, the first crossing from the left screen edge is
                  reported. If the number is two, the second crossing is
                  reported, etc.

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
              | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

When the :MEASure:TEDGE query is sent, the displayed signal is searched for the specified transition. The time interval between the trigger event and this occurrence is returned as the response to the query. The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the midpoint threshold in the positive direction. Once this crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified vertical value, or if the waveform does not cross the specified vertical value for the specific number of times in the direction specified.

You can make delay and phase measurements using the MEASure:TEDGE command:

Delay = time at the nth rising or falling edge of the channel - time at the same edge of another channel

Phase = (delay between channels / period of channel) x 360

For an example of making a delay and phase measurement, see "[:MEASure:TEDGE Code](#)" on page 366.

If the optional source parameter is specified, the current source is modified.

NOTE

This query is not available if the source is FFT (Fast Fourier Transform).

Return Format <value><NL>

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

:MEASure:TEDGE
Code

```
' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.
' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

See Also

- ["Introduction to :MEASure Commands"](#) on page 342
- [":MEASure:TVALUE"](#) on page 367
- [":MEASure:VTIME"](#) on page 376

:MEASure:TVALue

C (see [page 794](#))

Query Syntax `:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]`

`<value>` ::= the vertical value that the waveform must cross. The value can be volts or a math function value such as dB, Vs, or V/s.

`<slope>` ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

`<occurrence>` ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

`<source>` ::= {CHANnel<n> | FUNCTion | MATH | WMMemory<r>}

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

`<r>` ::= 1-2 in NR1 format

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE

This query is not available if the source is FFT (Fast Fourier Transform).

Return Format `<value><NL>`

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

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 342
 - "[":MEASure:TEDGE](#)" on page 365
 - "[":MEASure:VTIMe](#)" on page 376

:MEASure:VAMPlitude

 (see [page 794](#))

Command Syntax `:MEASure:VAMPlitude [<source>]`

```
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

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

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

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

$$\text{vertical amplitude} = \text{Vtop} - \text{Vbase}$$

Return Format `<value><NL>`

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

See Also

- "[Introduction to :MEASure Commands](#)" on page 342

- "[:MEASure:SOURce](#)" on page 363
- "[:MEASure:VBASe](#)" on page 371
- "[:MEASure:VTOP](#)" on page 377
- "[:MEASure:VPP](#)" on page 374

:MEASure:VAverage

(see [page 794](#))

Command Syntax `:MEASure:VAverage [<interval>] [,] [<source>]`

```
<interval> ::= {CYCLE | DISPLAY}
<source> ::= {CHANNEL<n> | FUNCTION | MATH | WMEMORY<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

The :MEASure:VAverage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

Query Syntax `:MEASure:VAverage? [<interval>] [,] [<source>]`

The :MEASure:VAverage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

Return Format `<value><NL>`

```
<value> ::= calculated average value in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MEASure:SOURce](#)" on page 363

:MEASure:VBASE

 (see [page 794](#))

Command Syntax `:MEASure:VBASE [⟨source⟩]`

```

⟨source⟩ ::= {CHANnel⟨n⟩ | FUNCTION | MATH | WMEMORY⟨r⟩}
⟨n⟩ ::= 1 to (# analog channels) in NR1 format
⟨r⟩ ::= 1-2 in NR1 format

```

The :MEASure:VBASE command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax `:MEASure:VBASE? [⟨source⟩]`

The :MEASure:VBASE? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format `<base_voltage><NL>`

```

<base_voltage> ::= value at the base of the selected waveform in
NR3 format

```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MEASure:SOURce](#)" on page 363
- "[:MEASure:VTOP](#)" on page 377
- "[:MEASure:VAMPLitude](#)" on page 369
- "[:MEASure:VMIN](#)" on page 373

:MEASure:VMAX

 (see [page 794](#))

Command Syntax `:MEASure:VMAX [<source>]`

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

`<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format`

`<r> ::= 1-2 in NR1 format`

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

Query Syntax `:MEASure:VMAX? [<source>]`

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

Return Format `<value><NL>`

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

See Also

- ["Introduction to :MEASure Commands"](#) on page 342

- [":MEASure:SOURce"](#) on page 363
- [":MEASure:VMIN"](#) on page 373
- [":MEASure:VPP"](#) on page 374
- [":MEASure:VTOP"](#) on page 377

:MEASure:VMIN

 (see [page 794](#))

Command Syntax `:MEASure:VMIN [<source>]`

```
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
```

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

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

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

Return Format `<value><NL>`

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

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:SOURce](#)" on page 363
- "[":MEASure:VBASe](#)" on page 371
- "[":MEASure:VMAX](#)" on page 372
- "[":MEASure:VPP](#)" on page 374

:MEASure:VPP



(see [page 794](#))

Command Syntax `:MEASure:VPP [<source>]`

`<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMORY<r>}`

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

`<r> ::= 1-2 in NR1 format`

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax `:MEASure:VPP? [<source>]`

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

$$V_{pp} = V_{max} - V_{min}$$

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

Return Format `<value><NL>`

`<value> ::= vertical peak to peak value in NR3 format`

- See Also**
- "Introduction to :MEASure Commands" on page 342
 - ":MEASure:SOURce" on page 363
 - ":MEASure:VMAX" on page 372
 - ":MEASure:VMIN" on page 373
 - ":MEASure:VAMPLitude" on page 369

:MEASure:VRMS

C (see [page 794](#))

Command Syntax	<code>:MEASure:VRMS [<interval> [,] [<type>] [,] [<source>]</code> <code><interval> ::= {CYCLE DISPLAY}</code> <code><type> ::= {AC DC}</code> <code><source> ::= {CHANnel<n> FUNCTION MATH WMEMORY<r>}</code> <code><n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format</code> <code><r> ::= 1-2 in NR1 format</code>
	The :MEASure:VRMS command installs a screen measurement and starts an RMS value measurement. If the optional source parameter is specified, the current source is modified.
	The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.
	The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax	<code>:MEASure:VRMS? [<interval> [,] [<type>] [,] [<source>]</code>
	The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.
Return Format	<code><value><NL></code> <code><value> ::= calculated dc RMS value in NR3 format</code>
See Also	<ul style="list-style-type: none"> • "Introduction to :MEASure Commands" on page 342 • ":MEASure:SOURce" on page 363

:MEASure:VTIMe

N (see [page 794](#))

Query Syntax

```
:MEASure:VTIMe? <vtimetime_argument>[,<source>]
<vtimetime_argument> ::= time from trigger in seconds
<source> ::= {<digital channels> | CHANnel<n> | FUNCTion | MATH
              | WMEMory<r>}
<digital channels> ::= DIGItal<d> for the MSO models
<n> ::= 1 to (# of analog channels) in NR1 format
<r> ::= 1-2 in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:VTIMe? query returns the value at a specified time on the source specified with :MEASure:SOURce. The specified time must be on the screen and is referenced to the trigger event. If the optional source parameter is specified, the current source is modified.

NOTE

This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>
<value> ::= value at the specified time in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MEASure:SOURce](#)" on page 363
- "[:MEASure:TEDGE](#)" on page 365
- "[:MEASure:TVALUE](#)" on page 367

:MEASure:VTOP

 (see [page 794](#))

Command Syntax

```
:MEASure:VTOP [⟨source⟩]
⟨source⟩ ::= {CHANnel⟨n⟩ | FUNCTion | MATH}
⟨n⟩ ::= 1-2 or 1-4 (# of analog channels) in NR1 format
⟨r⟩ ::= 1-2 in NR1 format
```

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

NOTE

This query is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VTOP? [⟨source⟩]
```

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

Return Format

```
⟨value⟩⟨NL⟩
⟨value⟩ ::= vertical value at the top of the waveform in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:SOURce](#)" on page 363
- "[":MEASure:VMAX](#)" on page 372
- "[":MEASure:VAMPLitude](#)" on page 369
- "[":MEASure:VBASE](#)" on page 371

:MEASure:WINDOW

N (see [page 794](#))

Command Syntax `:MEASure:WINDOW <type>`

`<type> ::= {MAIN | ZOOM | AUTO}`

When the zoomed time base is displayed, the :MEASure:WINDOW command lets you specify the measurement window:

- MAIN – the measurement window is the upper, Main window.
- ZOOM – the measurement window is the lower, Zoom window.
- AUTO – the measurement is attempted in the lower, Zoom window; if it cannot be made there, the upper, Main window is used.

Query Syntax `:MEASure:WINDOW?`

The :MEASure:WINDOW? query returns the current measurement window setting.

Return Format `<type><NL>`

`<type> ::= {MAIN | ZOOM | AUTO}`

See Also

- "[Introduction to :MEASure Commands](#)" on page 342

- "[:MEASure:SOURce](#)" on page 363

21 :MTEST Commands

The MTEST subsystem commands and queries control the mask test features. See "Introduction to :MTEST Commands" on page 381.

Table 71 :MTEST Commands Summary

Command	Query	Options and Query Returns
:MTEST:ALL {{0 OFF} {1 ON}} (see page 384)	:MTEST:ALL? (see page 384)	{0 1}
:MTEST:AMASK:CREAtE (see page 385)	n/a	n/a
:MTEST:AMASK:SOURce <source> (see page 386)	:MTEST:AMASK:SOURce? (see page 386)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:MTEST:AMASK:UNITS <units> (see page 387)	:MTEST:AMASK:UNITS? (see page 387)	<units> ::= {CURRent DIVisions}
:MTEST:AMASK:XDELta <value> (see page 388)	:MTEST:AMASK:XDELta? (see page 388)	<value> ::= X delta value in NR3 format
:MTEST:AMASK:YDELta <value> (see page 389)	:MTEST:AMASK:YDELta? (see page 389)	<value> ::= Y delta value in NR3 format
n/a	:MTEST:COUNT:FWAVefor ms? [CHANnel<n>] (see page 390)	<failed> ::= number of failed waveforms in NR1 format
:MTEST:COUNT:RESet (see page 391)	n/a	n/a
n/a	:MTEST:COUNT:TIME? (see page 392)	<time> ::= elapsed seconds in NR3 format
n/a	:MTEST:COUNT:WAVeform s? (see page 393)	<count> ::= number of waveforms in NR1 format
:MTEST:DATA <mask> (see page 394)	:MTEST:DATA? (see page 394)	<mask> ::= data in IEEE 488.2 # format.

Table 71 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:DELETE (see page 395)	n/a	n/a
:MTEST:ENABLE {{0 OFF} {1 ON}} (see page 396)	:MTEST:ENABLE? (see page 396)	{0 1}
:MTEST:LOCK {{0 OFF} {1 ON}} (see page 397)	:MTEST:LOCK? (see page 397)	{0 1}
:MTEST:RMODE <rmode> (see page 398)	:MTEST:RMODE? (see page 398)	<rmode> ::= {FORever TIME SIGMa WAveforms}
:MTEST:RMODE:FACTion:MEASure {{0 OFF} {1 ON}} (see page 399)	:MTEST:RMODE:FACTion:MEASure? (see page 399)	{0 1}
:MTEST:RMODE:FACTion:PRINT {{0 OFF} {1 ON}} (see page 400)	:MTEST:RMODE:FACTion:PRINT? (see page 400)	{0 1}
:MTEST:RMODE:FACTion:SAVE {{0 OFF} {1 ON}} (see page 401)	:MTEST:RMODE:FACTion:SAVE? (see page 401)	{0 1}
:MTEST:RMODE:FACTion:STOP {{0 OFF} {1 ON}} (see page 402)	:MTEST:RMODE:FACTion:STOP? (see page 402)	{0 1}
:MTEST:RMODE:SIGMa <level> (see page 403)	:MTEST:RMODE:SIGMa? (see page 403)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTEST:RMODE:TIME <seconds> (see page 404)	:MTEST:RMODE:TIME? (see page 404)	<seconds> ::= from 1 to 86400 in NR3 format
:MTEST:RMODE:WAveforms <count> (see page 405)	:MTEST:RMODE:WAveforms? (see page 405)	<count> ::= number of waveforms in NR1 format
:MTEST:SCALe:BIND {{0 OFF} {1 ON}} (see page 406)	:MTEST:SCALe:BIND? (see page 406)	{0 1}
:MTEST:SCALe:X1 <x1_value> (see page 407)	:MTEST:SCALe:X1? (see page 407)	<x1_value> ::= X1 value in NR3 format
:MTEST:SCALe:XDELta <xdelta_value> (see page 408)	:MTEST:SCALe:XDELta? (see page 408)	<xdelta_value> ::= X delta value in NR3 format

Table 71 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:SCALe:Y1 <y1_value> (see page 409)	:MTEST:SCALe:Y1? (see page 409)	<y1_value> ::= Y1 value in NR3 format
:MTEST:SCALe:Y2 <y2_value> (see page 410)	:MTEST:SCALe:Y2? (see page 410)	<y2_value> ::= Y2 value in NR3 format
:MTEST:SOURce <source> (see page 411)	:MTEST:SOURce? (see page 411)	<source> ::= {CHANnel<n> NONE} <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
n/a	:MTEST:TITLe? (see page 412)	<title> ::= a string of up to 128 ASCII characters

Introduction to :MTEST Commands Mask testing automatically compares the current displayed waveform with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

Reporting the Setup

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

Return Format

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

```
:MTES:SOUR CHAN1;ENAB 0;LOCK 1;:MTES:AMAS:SOUR CHAN1;UNIT DIV;XDEL
+2.5000000E-001;YDEL +2.5000000E-001;:MTES:SCAL:X1 +200.000E-06;XDEL
+400.000E-06;Y1 -3.00000E+00;Y2 +3.00000E+00;BIND 0;:MTES:RMOD
FOR;RMOD:TIME +1E+00;WAV 1000;SIGM +6.0E+00;:MTES:RMOD:FACT:STOP
0;PRIN 0;SAVE 0
```

Example Code

```
' Mask testing commands example.
' -----
Option Explicit

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

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

Sub Main()
```

```

On Error GoTo VisaComError

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

' Make sure oscilloscope is running.
myScope.WriteString ":RUN"

' Set mask test termination conditions.
myScope.WriteString ":MTEST:RMODE SIGMa"
myScope.WriteString ":MTEST:RMODE?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test termination mode: " + strQueryResult

myScope.WriteString ":MTEST:RMODE:SIGMa 4.2"
myScope.WriteString ":MTEST:RMODE:SIGMa?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test termination 'test sigma': " + _
    FormatNumber(varQueryResult)

' Use auto-mask to create mask.
myScope.WriteString ":MTEST:AMASK:SOURce CHANnel1"
myScope.WriteString ":MTEST:AMASK:SOURce?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test auto-mask source: " + strQueryResult

myScope.WriteString ":MTEST:AMASK:UNITS DIVisions"
myScope.WriteString ":MTEST:AMASK:UNITS?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test auto-mask units: " + strQueryResult

myScope.WriteString ":MTEST:AMASK:XDELta 0.1"
myScope.WriteString ":MTEST:AMASK:XDELta?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test auto-mask X delta: " + _
    FormatNumber(varQueryResult)

myScope.WriteString ":MTEST:AMASK:YDELta 0.1"
myScope.WriteString ":MTEST:AMASK:YDELta?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test auto-mask Y delta: " + _
    FormatNumber(varQueryResult)

' Enable "Auto Mask Created" event (bit 10, &H400)
myScope.WriteString "*CLS"
myScope.WriteString ":MTEenable " + CStr(CInt("&H400"))

' Create mask.
myScope.WriteString ":MTEST:AMASK:CREATE"
Debug.Print "Auto-mask created, mask test automatically enabled."

' Set up timeout variables.
Dim lngTimeout As Long      ' Max millisecs to wait.

```

```

Dim lngElapsed As Long
lngTimeout = 60000    ' 60 seconds.

' Wait until mask is created.
lngElapsed = 0
Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERRegister:CONDITION?"
    varQueryResult = myScope.ReadNumber
    ' Operation Status Condition Register MTE bit (bit 9, &H200).
    If (varQueryResult And &H200) <> 0 Then
        Exit Do
    Else
        Sleep 100    ' Small wait to prevent excessive queries.
        lngElapsed = lngElapsed + 100
    End If
Loop

' Look for RUN bit = stopped (mask test termination).
lngElapsed = 0
Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERRegister:CONDITION?"
    varQueryResult = myScope.ReadNumber
    ' Operation Status Condition Register RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
        Exit Do
    Else
        Sleep 100    ' Small wait to prevent excessive queries.
        lngElapsed = lngElapsed + 100
    End If
Loop

' Get total waveforms, failed waveforms, and test time.
myScope.WriteString ":MTEST:COUNT:WAVEforms?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test total waveforms: " + strQueryResult

myScope.WriteString ":MTEST:COUNT:FWAVEforms?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test failed waveforms: " + strQueryResult

myScope.WriteString ":MTEST:COUNT:TIME?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test elapsed seconds: " + strQueryResult

Exit Sub

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

End Sub

```

:MTEST:ALL

N (see [page 794](#))

Command Syntax `:MTEST:ALL <on_off>`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:ALL command specifies the channel(s) that are included in the mask test:

- ON – All displayed analog channels are included in the mask test.
- OFF – Just the selected source channel is included in the test.

Query Syntax `:MTEST:ENABLE?`

The :MTEST:ENABLE? query returns the current setting.

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also • "Introduction to :MTEST Commands" on page 381

:MTEST:AMASK:CREate

N (see [page 794](#))

Command Syntax

`:MTEST :AMASK :CREate`

The :MTEST:AMASK:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the :MTEST:AMASK:XDELta, :MTEST:AMASK:YDELta, and :MTEST:AMASK:UNITs commands. The mask only encompasses the portion of the waveform visible on the display, so you must ensure that the waveform is acquired and displayed consistently to obtain repeatable results.

The :MTEST:SOURce command selects the channel and should be set before using this command.

See Also

- ["Introduction to :MTEST Commands"](#) on page 381
- [":MTEST:AMASK:XDELta"](#) on page 388
- [":MTEST:AMASK:YDELta"](#) on page 389
- [":MTEST:AMASK:UNITs"](#) on page 387
- [":MTEST:AMASK:SOURce"](#) on page 386
- [":MTEST:SOURce"](#) on page 411

Example Code

- ["Example Code"](#) on page 381

:MTEST:AMASK:SOURce

N (see [page 794](#))

Command Syntax `:MTEST:AMASK:SOURce <source>`

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

The :MTEST:AMASK:SOURce command selects the source for the interpretation of the :MTEST:AMASK:XDELta and :MTEST:AMASK:YDELta parameters when :MTEST:AMASK:UNITS is set to CURRent.

When UNITS are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel<n>:UNITS command, of the selected source.

Suppose that UNITS are CURRent and that you set SOURce to CHANNEL1, which is using units of volts. Then you can define AMASK:XDELta in terms of volts and AMASK:YDELta in terms of seconds.

This command is the same as the :MTEST:SOURce command.

Query Syntax `:MTEST:AMASK:SOURce?`

The :MTEST:AMASK:SOURce? query returns the currently set source.

Return Format

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

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[:MTEST:AMASK:XDELta](#)" on page 388
- "[:MTEST:AMASK:YDELta](#)" on page 389
- "[:MTEST:AMASK:UNITS](#)" on page 387
- "[:MTEST:SOURce](#)" on page 411

Example Code

- "[Example Code](#)" on page 381

:MTEST:AMASK:UNITS

N (see [page 794](#))

Command Syntax :MTEST:AMASK:UNITS <units>

<units> ::= {CURREnt | DIVisions}

The :MTEST:AMASK:UNITS command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by :MTEST:AMASK:XDELta and :MTEST:AMASK:YDELta commands.

- CURREnt – the mask test subsystem uses the units as set by the :CHANnel<n>:UNITS command, usually time for ΔX and voltage for ΔY .
- DIVisions – the mask test subsystem uses the graticule as the measurement system, so tolerance settings are specified as parts of a screen division. The mask test subsystem maintains separate XDELta and YDELta settings for CURREnt and DIVisions. Thus, XDELta and YDELta are not converted to new values when the UNITS setting is changed.

Query Syntax :MTEST:AMASK:UNITS?

The :MTEST:AMASK:UNITS query returns the current measurement units setting for the mask test automask feature.

Return Format <units><NL>

<units> ::= {CURR | DIV}

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 381
 - "[:MTEST:AMASK:XDELta](#)" on page 388
 - "[:MTEST:AMASK:YDELta](#)" on page 389
 - "[:CHANnel<n>:UNITS](#)" on page 234
 - "[:MTEST:AMASK:SOURce](#)" on page 386
 - "[:MTEST:SOURce](#)" on page 411

Example Code

- "[Example Code](#)" on page 381

:MTEST:AMASK:XDELta

N (see [page 794](#))

Command Syntax `:MTEST:AMASK:XDELta <value>`

`<value> ::= X delta value in NR3 format`

The :MTEST:AMASK:XDELta command sets the tolerance in the X direction around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to horizontal values of the waveform to determine the boundaries of the mask.

The horizontal tolerance value is interpreted based on the setting specified by the :MTEST:AMASK:UNITS command; thus, if you specify 250-E3, the setting for :MTEST:AMASK:UNITS is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ± 250 ms. If the setting for :MTEST:AMASK:UNITS is DIVisions, the same X delta value will set the tolerance to ± 250 millidivisions, or 1/4 of a division.

Query Syntax `:MTEST:AMASK:XDELta?`

The :MTEST: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 :MTEST:AMASK:UNITS query.

Return Format `<value><NL>`

`<value> ::= X delta value in NR3 format`

See Also

- "Introduction to :MTEST Commands" on page 381
- ":MTEST:AMASK:UNITS" on page 387
- ":MTEST:AMASK:YDELta" on page 389
- ":MTEST:AMASK:SOURce" on page 386
- ":MTEST:SOURce" on page 411

Example Code

- "[Example Code](#)" on page 381

:MTEST:AMASK:YDELta

N (see [page 794](#))

Command Syntax :MTEST:AMASK:YDELta <value>

<value> ::= Y delta value in NR3 format

The :MTEST:AMASK:YDELta command sets the vertical tolerance around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to vertical values of the waveform to determine the boundaries of the mask.

The vertical tolerance value is interpreted based on the setting specified by the :MTEST:AMASK:UNITS command; thus, if you specify 250-E3, the setting for :MTEST:AMASK:UNITS is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ± 250 mV. If the setting for :MTEST:AMASK:UNITS is DIVisions, the same Y delta value will set the tolerance to ± 250 millidivisions, or 1/4 of a division.

Query Syntax :MTEST:AMASK:YDELta?

The :MTEST: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 :MTEST:AMASK:UNITS query.

Return Format <value><NL>

<value> ::= Y delta value in NR3 format

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 381
 - "[:MTEST:AMASK:UNITS](#)" on page 387
 - "[:MTEST:AMASK:XDELta](#)" on page 388
 - "[:MTEST:AMASK:SOURce](#)" on page 386
 - "[:MTEST:SOURce](#)" on page 411

Example Code

- "[Example Code](#)" on page 381

:MTEST:COUNT:FWAVeforms**N** (see [page 794](#))**Query Syntax** `:MTEST:COUNT:FWAVeforms? [CHANnel<n>]``<n> ::= 1 to (# analog channels) in NR1 format`

The :MTEST:COUNT:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms collected on the channel specified by the optional parameter or collected on the currently specified source channel (:MTEST:SOURce) if there is no parameter.

Return Format `<failed><NL>``<failed> ::= number of failed waveforms in NR1 format.`**See Also**

- "Introduction to :MTEST Commands" on page 381
- ":MTEST:COUNT:WAVEforms" on page 393
- ":MTEST:COUNT:TIME" on page 392
- ":MTEST:COUNT:RESET" on page 391
- ":MTEST:SOURce" on page 411

Example Code · "[Example Code](#)" on page 381

:MTEST:COUNt:RESet

N (see [page 794](#))

Command Syntax :MTEST:COUNt:RESet

The :MTEST:COUNt:RESet command resets the mask statistics.

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:COUNt:WAVeforms](#)" on page 393
- "[":MTEST:COUNt:FWAVeforms](#)" on page 390
- "[":MTEST:COUNt:TIME](#)" on page 392

:MTEST:COUNt:TIME**N** (see [page 794](#))**Query Syntax** `:MTEST:COUNt:TIME?`

The `:MTEST:COUNt:TIME?` query returns the elapsed time in the current mask test run.

Return Format `<time><NL>`

`<time>` ::= elapsed seconds in NR3 format.

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 381
 - "[:MTEST:COUNt:WAVEforms](#)" on page 393
 - "[:MTEST:COUNt:FWAveforms](#)" on page 390
 - "[:MTEST:COUNt:RESet](#)" on page 391

Example Code

- "[Example Code](#)" on page 381

:MTEST:COUNt:WAveforms

N (see [page 794](#))

Query Syntax `:MTEST:COUNt:WAveforms?`

The `:MTEST:COUNt:WAveforms?` query returns the total number of waveforms acquired in the current mask test run.

Return Format `<count><NL>`

`<count>` ::= number of waveforms in NR1 format.

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 381
 - "[:MTEST:COUNt:FWAVeforms](#)" on page 390
 - "[:MTEST:COUNt:TIME](#)" on page 392
 - "[:MTEST:COUNt:RESet](#)" on page 391

Example Code

- "[Example Code](#)" on page 381

:MTEST:DATA**N** (see [page 794](#))**Command Syntax** `:MTEST:DATA <mask>``<mask> ::= binary block data in IEEE 488.2 # format.`

The :MTEST:DATA command loads a mask from binary block data. These are the data bytes found in a *.msk file.

Query Syntax `:MTEST:DATA?`

The :MTEST:DATA? query returns a mask in binary block data format. The format for the data transmission is the # definite-length format defined in the IEEE 488.2 specification.

Return Format `<mask><NL>``<mask> ::= binary block data in IEEE 488.2 # format`**See Also**

- [":SAVE:MASK\[:STARt\]" on page 437](#)
- [":RECall:MASK\[:STARt\]" on page 422](#)

:MTEST:DELetE

N (see [page 794](#))

Command Syntax :MTEST:DELetE

The :MTEST:DELetE command clears the currently loaded mask.

See Also

- ["Introduction to :MTEST Commands"](#) on page 381
- [":MTEST:AMASK:CREate"](#) on page 385

:MTEST:ENABLE

N (see [page 794](#))

Command Syntax `:MTEST:ENABLE <on_off>`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:ENABLE command enables or disables the mask test features.

- ON – Enables the mask test features.
- OFF – Disables the mask test features.

Query Syntax `:MTEST:ENABLE?`

The :MTEST:ENABLE? query returns the current state of mask test features.

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also · "Introduction to :MTEST Commands" on page 381

:MTEST:LOCK

N (see [page 794](#))

Command Syntax `:MTEST:LOCK <on_off>`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:LOCK command enables or disables the mask lock feature:

- ON – Locks a mask to the SOURce. As the vertical or horizontal scaling or position of the SOURce changes, the mask is redrawn accordingly.
- OFF – The mask is static and does not move.

Query Syntax `:MTEST:LOCK?`

The :MTEST:LOCK? query returns the current mask lock setting.

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also

- "[Introduction to :MTEST Commands](#)" on page 381

- "[:MTEST:SOURce](#)" on page 411

:MTEST:RMODE

N (see [page 794](#))

Command Syntax `:MTEST:RMODE <rmode>`

`<rmode> ::= {FORever | SIGMa | TIME | WAVEforms}`

The :MTEST:RMODE command specifies the termination conditions for the mask test:

- FORever – the mask test runs until it is turned off.
- SIGMa – the mask test runs until the Sigma level is reached. This level is set by the "[:MTEST:RMODE:SIGMA](#)" on page 403 command.
- TIME – the mask test runs for a fixed amount of time. The amount of time is set by the "[:MTEST:RMODE:TIME](#)" on page 404 command.
- WAVEforms – the mask test runs until a fixed number of waveforms are acquired. The number of waveforms is set by the "[:MTEST:RMODE:WAVEforms](#)" on page 405 command.

Query Syntax `:MTEST:RMODE?`

The :MTEST:RMODE? query returns the currently set termination condition.

Return Format `<rmode><NL>`

`<rmode> ::= {FOR | SIGM | TIME | WAV}`

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[:MTEST:RMODE:SIGMA](#)" on page 403
- "[:MTEST:RMODE:TIME](#)" on page 404
- "[:MTEST:RMODE:WAVEforms](#)" on page 405

Example Code

- "[Example Code](#)" on page 381

:MTEST:RMODE:FACTion:MEASure

N (see [page 794](#))

Command Syntax `:MTEST:RMODE:FACTion:MEASure <on_off>`
`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:RMODE:FACTion:MEASure command sets measuring only mask failures on or off.

When ON, measurements and measurement statistics run only on waveforms that contain a mask violation; passing waveforms do not affect measurements and measurement statistics.

This mode is not available when the acquisition mode is set to Averaging.

Query Syntax `:MTEST:RMODE:FACTion:MEASure?`

The :MTEST:RMODE:FACTion:MEASure? query returns the current mask failure measure setting.

Return Format `<on_off><NL>`
`<on_off> ::= {1 | 0}`

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:RMODE:FACTion:PRINT](#)" on page 400
- "[":MTEST:RMODE:FACTion:SAVE](#)" on page 401
- "[":MTEST:RMODE:FACTion:STOP](#)" on page 402

:MTEST:RMODE:FACTion:PRINT

N (see [page 794](#))

Command Syntax `:MTEST:RMODE:FACTion:PRINT <on_off>`
`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:RMODE:FACTion:PRINT command sets printing on mask failures on or off.

NOTE

Setting :MTEST:RMODE:FACTion:PRINT ON automatically sets :MTEST:RMODE:FACTion:SAVE OFF.

See [Chapter 17](#), “:HARDcopy Commands,” starting on page 297 for more information on setting the hardcopy device and formatting options.

Query Syntax `:MTEST:RMODE:FACTion:PRINT?`

The :MTEST:RMODE:FACTion:PRINT? query returns the current mask failure print setting.

Return Format `<on_off><NL>`
`<on_off> ::= {1 | 0}`

See Also

- ["Introduction to :MTEST Commands"](#) on page 381
- [":MTEST:RMODE:FACTion:MEASure"](#) on page 399
- [":MTEST:RMODE:FACTion:SAVE"](#) on page 401
- [":MTEST:RMODE:FACTion:STOP"](#) on page 402

:MTEST:RMODE:FACTion:SAVE

N (see [page 794](#))

Command Syntax `:MTEST:RMODE:FACTion:SAVE <on_off>`
`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:RMODE:FACTion:SAVE command sets saving on mask failures on or off.

NOTE

Setting :MTEST:RMODE:FACTion:SAVE ON automatically sets :MTEST:RMODE:FACTion:PRINT OFF.

See [Chapter 24](#), “:SAVE Commands,” starting on page 427 for more information on save options.

Query Syntax `:MTEST:RMODE:FACTion:SAVE?`

The :MTEST:RMODE:FACTion:SAVE? query returns the current mask failure save setting.

Return Format `<on_off><NL>`
`<on_off> ::= {1 | 0}`

See Also

- ["Introduction to :MTEST Commands"](#) on page 381
- [":MTEST:RMODE:FACTion:MEASure"](#) on page 399
- [":MTEST:RMODE:FACTion:PRINT"](#) on page 400
- [":MTEST:RMODE:FACTion:STOP"](#) on page 402

:MTEST:RMODE:FACTion:STOP**N** (see [page 794](#))

Command Syntax `:MTEST:RMODE:FACTion:STOP <on_off>`
`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTEST:RMODE:FACTion:STOP command sets stopping on a mask failure on or off. When this setting is ON and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Query Syntax `:MTEST:RMODE:FACTion:STOP?`

The :MTEST:RMODE:FACTion:STOP? query returns the current mask failure stop setting.

Return Format `<on_off><NL>`
`<on_off> ::= {1 | 0}`

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:RMODE:FACTion:MEASure](#)" on page 399
- "[":MTEST:RMODE:FACTion:PRINT](#)" on page 400
- "[":MTEST:RMODE:FACTion:SAVE](#)" on page 401

:MTEST:RMODE:SIGMa

N (see [page 794](#))

Command Syntax :MTEST:RMODE:SIGMa <level>

<level> ::= from 0.1 to 9.3 in NR3 format

When the :MTEST:RMODE command is set to SIGMa, the :MTEST:RMODE:SIGMa command sets the *test sigma* level to which a mask test runs. *Test sigma* is the best achievable process sigma, assuming no failures. (*Process sigma* is calculated using the number of failures per test.) The *test sigma* level indirectly specifies the number of waveforms that must be tested (in order to reach the sigma level).

Query Syntax :MTEST:RMODE:SIGMa?

The :MTEST:RMODE:SIGMa? query returns the current Sigma level setting.

Return Format <level><NL>

<level> ::= from 0.1 to 9.3 in NR3 format

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:RMODE"](#) on page 398

Example Code

- "[":Example Code"](#) on page 381

:MTESt:RMODE:TIME

N (see [page 794](#))

Command Syntax `:MTESt:RMODE:TIME <seconds>`

`<seconds> ::= from 1 to 86400 in NR3 format`

When the :MTESt:RMODE command is set to TIME, the :MTESt:RMODE:TIME command sets the number of seconds for a mask test to run.

Query Syntax `:MTESt:RMODE:TIME?`

The :MTESt:RMODE:TIME? query returns the number of seconds currently set.

Return Format `<seconds><NL>`

`<seconds> ::= from 1 to 86400 in NR3 format`

See Also

- "Introduction to :MTESt Commands" on page 381
- "[:MTESt:RMODE](#)" on page 398

:MTEST:RMODE:WAVEforms

N (see [page 794](#))

Command Syntax :MTEST:RMODE:WAVEforms <count>

<count> ::= number of waveforms in NR1 format
from 1 to 2,000,000,000

When the :MTEST:RMODE command is set to WAVEforms, the :MTEST:RMODE:WAVEforms command sets the number of waveform acquisitions that are mask tested.

Query Syntax :MTEST:RMODE:WAVEforms?

The :MTEST:RMODE:WAVEforms? query returns the number of waveforms currently set.

Return Format <count><NL>

<count> ::= number of waveforms in NR1 format
from 1 to 2,000,000,000

See Also • ["Introduction to :MTEST Commands"](#) on page 381
• [":MTEST:RMODE"](#) on page 398

:MTESt:SCALe:BIND

N (see [page 794](#))

Command Syntax `:MTESt:SCALe:BIND <on_off>`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :MTESt:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control:

- ON –

If the Bind 1 & 0 Levels control is enabled, the 1 Level and the 0 Level controls track each other. Adjusting either the 1 Level or the 0 Level control shifts the position of the mask up or down without changing its size.

If the Bind -1 & 0 Levels control is enabled, the -1 Level and the 0 Level controls track each other. Adjusting either the -1 Level or the 0 Level control shifts the position of the mask up or down without changing its size.

- OFF –

If the Bind 1 & 0 Levels control is disabled, adjusting either the 1 Level or the 0 Level control changes the vertical height of the mask.

If the Bind -1 & 0 Levels control is disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

Query Syntax `:MTESt:SCALe:BIND?`

The :MTESt:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also

- "[Introduction to :MTESt Commands](#)" on page 381
- "[":MTESt:SCALe:X1](#)" on page 407
- "[":MTESt:SCALe:XDELta](#)" on page 408
- "[":MTESt:SCALe:Y1](#)" on page 409
- "[":MTESt:SCALe:Y2](#)" on page 410

:MTEST:SCALe:X1

N (see [page 794](#))

Command Syntax :MTEST:SCALe:X1 <x1_value>

<x1_value> ::= X1 value in NR3 format

The :MTEST:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the :MTEST:SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation:

$$X = (X * \Delta X) + X1$$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

The X1 value is a time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Query Syntax :MTEST:SCALe:X1?

The :MTEST:SCALe:X1? query returns the current X1 coordinate setting.

Return Format <x1_value><NL>

<x1_value> ::= X1 value in NR3 format

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 381
 - "[":MTEST:SCALe:BIND](#)" on page 406
 - "[":MTEST:SCALe:XDELta](#)" on page 408
 - "[":MTEST:SCALe:Y1](#)" on page 409
 - "[":MTEST:SCALe:Y2](#)" on page 410

:MTEST:SCALe:XDELta

N (see [page 794](#))

Command Syntax `:MTEST:SCALe:XDELta <xdelta_value>`

`<xdelta_value> ::= X delta value in NR3 format`

The :MTEST:SCALe:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and 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:

$$X = (X * \Delta X) + X1$$

The X delta value is a time value specifying the distance of the X2 marker with respect to the X1 marker.

For example, if the period of the waveform you wish to test is 1 ms, setting ΔX to 1 ms ensures that the waveform's period is between the X1 and X2 markers.

Query Syntax `:MTEST:SCALe:XDELta?`

The :MTEST:SCALe:XDELta? query returns the current value of ΔX .

Return Format `<xdelta_value><NL>`

`<xdelta_value> ::= X delta value in NR3 format`

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[:MTEST:SCALe:BIND](#)" on page 406
- "[:MTEST:SCALe:X1](#)" on page 407
- "[:MTEST:SCALe:Y1](#)" on page 409
- "[:MTEST:SCALe:Y2](#)" on page 410

:MTEST:SCALe:Y1

N (see [page 794](#))

Command Syntax :MTEST:SCALe:Y1 <y1_value>

<y1_value> ::= Y1 value in NR3 format

The :MTEST:SCALe:Y1 command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2 according to the equation:

$$Y = (Y * (Y2 - Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

The Y1 value is a voltage value specifying the point at which Y=0.

Query Syntax :MTEST:SCALe:Y1?

The :MTEST:SCALe:Y1? query returns the current setting of the Y1 marker.

Return Format <y1_value><NL>

<y1_value> ::= Y1 value in NR3 format

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[:MTEST:SCALe:BIND](#)" on page 406
- "[:MTEST:SCALe:X1](#)" on page 407
- "[:MTEST:SCALe:XDELta](#)" on page 408
- "[:MTEST:SCALe:Y2](#)" on page 410

:MTEST:SCALe:Y2

N (see [page 794](#))

Command Syntax `:MTEST:SCALe:Y2 <y2_value>`

`<y2_value> ::= Y2 value in NR3 format`

The :MTEST:SCALe:Y2 command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2 according to the following equation:

$$Y = (Y * (Y2 - Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

The Y2 value is a voltage value specifying the location of the Y2 marker.

Query Syntax `:MTEST:SCALe:Y2?`

The :MTEST:SCALe:Y2? query returns the current setting of the Y2 marker.

Return Format `<y2_value><NL>`

`<y2_value> ::= Y2 value in NR3 format`

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 381
 - "[":MTEST:SCALe:BIND](#)" on page 406
 - "[":MTEST:SCALe:X1](#)" on page 407
 - "[":MTEST:SCALe:XDELta](#)" on page 408
 - "[":MTEST:SCALe:Y1](#)" on page 409

:MTEST:SOURce

N (see [page 794](#))

Command Syntax :MTEST:SOURce <source>

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

The :MTEST:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

Query Syntax :MTEST:SOURce?

The :MTEST:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Return Format <source><NL>

```
<source> ::= {CHAN<n> | NONE}
<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:AMASK:SOURce](#)" on page 386

:MTEST:TITLE

N (see [page 794](#))

Query Syntax `:MTEST:TITLE?`

The `:MTEST:TITLE?` query returns the mask title which is a string of up to 128 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

Return Format `<title><NL>`

`<title>` ::= a string of up to 128 ASCII characters.

See Also • ["Introduction to :MTEST Commands"](#) on page 381

22 :POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "[Introduction to :POD<n> Commands](#)" on page 413.

Table 72 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {0 OFF} {1 ON} (see page 415)	:POD<n>:DISPlay? (see page 415)	{0 1} <n> ::= 1 in NR1 format
:POD<n>:SIZE <value> (see page 416)	:POD<n>:SIZE? (see page 416)	<value> ::= {SMALL MEDIUM LARGe}
:POD<n>:THreshold <type>[suffix] (see page 417)	:POD<n>:THreshold? (see page 417)	<n> ::= 1 in NR1 format <type> ::= {CMOS ECL TTL <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V mV uV }

Introduction to :POD<n> Commands `<n> ::= 1`
The POD subsystem commands control the viewing and threshold of groups of digital channels.
`POD1 ::= D0-D7`

NOTE

These commands are only valid for the MSO models.

Reporting the Setup

Use :POD1? to query setup information for the POD subsystem.

Return Format

The following is a sample response from the :POD1? query. In this case, the query was issued following a *RST command.

```
:POD1:DISP 0;THR 1.40E+00
```

:POD<n>:DISPlay

N (see [page 794](#))

Command Syntax `:POD<n>:DISPlay <display>`

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

`<n> ::= An integer, 1, is attached as a suffix to the command and defines the group of channels that are affected by the command.`

`POD1 ::= D0-D7`

The :POD<n>:DISPlay command turns displaying of the specified group of channels on or off.

NOTE

This command is only valid for the MSO models.

Query Syntax `:POD<n>:DISPlay?`

The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

Return Format `<display><NL>`

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

See Also

- "[Introduction to :POD<n> Commands](#)" on page 413
- "[:DIGItal<d>:DISPlay](#)" on page 245
- "[:CHANnel<n>:DISPlay](#)" on page 221
- "[:VIEW](#)" on page 180
- "[:BLANK](#)" on page 156
- "[:STATus](#)" on page 177

:POD<n>:SIZE

N (see [page 794](#))

Command Syntax `:POD<n>:SIZE <value>`

`<n>` ::= An integer, 1, is attached as a suffix to the command and defines the group of channels that are affected by the command.

`POD1` ::= D0-D7

`<value>` ::= {SMALL | MEDium | LARGe}

The :POD<n>:SIZE command specifies the size of digital channels on the display.

NOTE

This command is only valid for the MSO models.

Query Syntax `:POD<n>:SIZE?`

The :POD<n>:SIZE? query returns the digital channels size setting.

Return Format `<size_value><NL>`

`<size_value>` ::= {SMAL | MED | LARG}

See Also

- "Introduction to :POD<n> Commands" on page 413
- ":DIGItal<d>:SIZE" on page 248
- ":DIGItal<d>:POStion" on page 247

:POD<n>:THreshold

N (see [page 794](#))

Command Syntax	<code>:POD<n>:THreshold <type>[<suffix>]</code>
	<code><n></code> ::= An integer, 1, is attached as a suffix to the command and defines the group of channels that are affected by the command.
	<code><type></code> ::= {CMOS ECL TTL <user defined value>}
	<code><user defined value></code> ::= -8.00 to +8.00 in NR3 format
	<code><suffix></code> ::= {V mV uV}
	POD1 ::= D0-D7
	TTL ::= 1.4V
	CMOS ::= 2.5V
	ECL ::= -1.3V

The :POD<n>:THreshold command sets the threshold for the specified group of channels. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE

This command is only valid for the MSO models.

Query Syntax

`:POD<n>:THreshold?`

The :POD<n>:THreshold? query returns the threshold value for the specified group of channels.

Return Format

`<threshold><NL>`
`<threshold>` ::= Floating point number in NR3 format

See Also

- ["Introduction to :POD<n> Commands"](#) on page 413
- [":DIGItal<d>:THReshold"](#) on page 249
- [":TRIGger\[:EDGE\]:LEVel"](#) on page 605

Example Code

```
' THRESHOLD - This command is used to set the voltage threshold for
' the waveforms. There are three preset values (TTL, CMOS, and ECL)
' and you can also set a user-defined threshold value between
' -8.0 volts and +8.0 volts.
'
' In this example, we set channels 0-7 to CMOS, and then set the
' external trigger to TTL. Of course, you only need to set the
' thresholds for the channels you will be using in your program.

' Set channels 0-7 to CMOS threshold.
myScope.WriteString ":POD1:THRESHOLD CMOS"
```

```
' Set external channel to TTL threshold (short form).  
myScope.WriteString ":TRIG:LEV TTL,EXT"
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

23 :RECall Commands

Recall previously saved oscilloscope setups, reference waveforms, and masks.

Table 73 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILEname <base_name> (see page 421)	:RECall:FILEname? (see page 421)	<base_name> ::= quoted ASCII string
:RECall:MASK[:START] [<file_spec>] (see page 422)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:PWD <path_name> (see page 423)	:RECall:PWD? (see page 423)	<path_name> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see page 424)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:WMEMory<r>[:START] [<file_name>] (see page 425)	n/a	<r> ::= 1-2 in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Introduction to :RECall Commands The :RECall subsystem provides commands to recall previously saved oscilloscope setups, reference waveforms, and masks.

Reporting the Setup

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

Return Format

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

```
:REC:FIL "scope_0"
```

:RECall:FILEname

N (see [page 794](#))

Command Syntax `:RECall:FILEname <base_name>`
`<base_name> ::= quoted ASCII string`

The :RECall:FILEname command specifies the source for any RECall operations.

NOTE

This command specifies a file's base name only, without path information or an extension.

Query Syntax `:RECall:FILEname?`

The :RECall:FILEname? query returns the current RECall filename.

Return Format `<base_name><NL>`
`<base_name> ::= quoted ASCII string`

See Also

- ["Introduction to :RECall Commands" on page 419](#)
- [":RECall:SETup\[:STARt\]" on page 424](#)
- [":SAVE:FILEname" on page 430](#)

:RECall:MASK[:STARt]**N** (see [page 794](#))**Command Syntax** `:RECall:MASK[:STARt] [<file_spec>]`

`<file_spec>` ::= {`<internal_loc>` | `<file_name>`}
`<internal_loc>` ::= 0-3; an integer in NR1 format
`<file_name>` ::= quoted ASCII string

The :RECall:MASK[:STARt] command recalls a mask.

NOTE

If a file extension is provided as part of a specified `<file_name>`, it must be ".msk".

See Also

- "[Introduction to :RECall Commands](#)" on page 419
- "[":RECall:FILEname](#)" on page 421
- "[":SAVE:MASK\[:STARt\]](#)" on page 437
- "[":MTEST:DATA](#)" on page 394

:RECall:PWD

N (see [page 794](#))

Command Syntax :RECall:PWD <path_name>

<path_name> ::= quoted ASCII string

The :RECall:PWD command sets the present working directory for recall operations.

Query Syntax :RECall:PWD?

The :RECall:PWD? query returns the currently set working directory for recall operations.

Return Format <path_name><NL>

<path_name> ::= quoted ASCII string

See Also • ["Introduction to :RECall Commands" on page 419](#)

• [":SAVE:PWD" on page 439](#)

:RECall:SETUp[:STARt]**N** (see [page 794](#))**Command Syntax** `:RECall:SETUp [:STARt] [<file_spec>]``<file_spec> ::= {<internal_loc> | <file_name>}``<internal_loc> ::= 0-9; an integer in NR1 format``<file_name> ::= quoted ASCII string`

The :RECall:SETUp[:STARt] command recalls an oscilloscope setup.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".scp".

See Also

- ["Introduction to :RECall Commands" on page 419](#)
- [":RECall:FILEname" on page 421](#)
- [":SAVE:SETUp\[:STARt\]" on page 440](#)

:RECall:WMEMory<r>[:STARt]

N (see [page 794](#))

Command Syntax :RECall:WMEMory<r>[:STARt] [<file_name>]

<r> ::= 1-2 in NRI format

<file_name> ::= quoted ASCII string

The :RECall:WMEMory<r>[:STARt] command recalls a reference waveform.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".h5".

See Also

- "[Introduction to :RECall Commands](#)" on page 419
- "[:RECall:FILENAME](#)" on page 421
- "[:SAVE:WMEMORY\[:STARt\]](#)" on page 447

24 :SAVE Commands

Save oscilloscope setups, screen images, and data. See "[Introduction to :SAVE Commands](#)" on page 429.

Table 74 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILEname <base_name> (see page 430)	:SAVE:FILEname? (see page 430)	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_name>] (see page 431)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGE:FACTors { {0 OFF} {1 ON} } (see page 432)	:SAVE:IMAGE:FACTors? (see page 432)	{0 1}
:SAVE:IMAGE:FORMAT <format> (see page 433)	:SAVE:IMAGE:FORMAT? (see page 433)	<format> ::= {TIFF {BMP BMP24bit} BMP8bit PNG NONE}
:SAVE:IMAGE:INKSaver { {0 OFF} {1 ON} } (see page 434)	:SAVE:IMAGE:INKSaver? (see page 434)	{0 1}
:SAVE:IMAGE:PALETTE <palette> (see page 435)	:SAVE:IMAGE:PALETTE? (see page 435)	<palette> ::= {COLOR GRAYscale MONochrome}
:SAVE:LISTER[:START] [<file_name>] (see page 436)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:START] [<file_spec>] (see page 437)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string

Table 74 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:MULTi[:START] [<file_name>] (see page 438)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 439)	:SAVE:PWD? (see page 439)	<path_name> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see page 440)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:STARt] [<file_name>] (see page 441)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMAT <format> (see page 442)	:SAVE:WAVeform:FORMAT? (see page 442)	<format> ::= {ASCIixy CSV BINary NONE}
:SAVE:WAVeform:LENGTH <length> (see page 443)	:SAVE:WAVeform:LENGTH? (see page 443)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGTH: :MAX {{0 OFF} {1 ON}} (see page 444)	:SAVE:WAVeform:LENGTH: :MAX? (see page 444)	{0 1}
:SAVE:WAVeform:SEGMen ted <option> (see page 445)	:SAVE:WAVeform:SEGMen ted? (see page 445)	<option> ::= {ALL CURRent}
:SAVE:WMEMory:SOURce <source> (see page 446)	:SAVE:WMEMory:SOURce? (see page 446)	<source> ::= {CHANnel<n> FUNCtion MATH WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:START] [<file_name>] (see page 447)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Introduction to :SAVE Commands The :SAVE subsystem provides commands to save oscilloscope setups, screen images, and data.

:SAV is an acceptable short form for :SAVE.

Reporting the Setup

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

Return Format

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

```
:SAVE:FIL ""; :SAVE:IMAG:AREA GRAT;FACT 0;FORM TIFF;INKS 0;PAL  
MON; :SAVE:PWD "C:/setups/"; :SAVE:WAV:FORM NONE;LENG 1000;SEGM CURR
```

:SAVE:FILEname**N** (see [page 794](#))**Command Syntax** `:SAVE:FILEname <base_name>` `<base_name> ::= quoted ASCII string`

The :SAVE:FILEname command specifies the source for any SAVE operations.

NOTE

This command specifies a file's base name only, without path information or an extension.

Query Syntax `:SAVE:FILEname?`

The :SAVE:FILEname? query returns the current SAVE filename.

Return Format `<base_name><NL>` `<base_name> ::= quoted ASCII string`**See Also**

- "[Introduction to :SAVE Commands](#)" on page 429

- "[":SAVE:IMAGe\[:STARt\]](#)" on page 431

- "[":SAVE:SETup\[:STARt\]](#)" on page 440

- "[":SAVE:WAVEform\[:STARt\]](#)" on page 441

- "[":SAVE:PWD](#)" on page 439

- "[":RECall:FILEname](#)" on page 421

:SAVE:IMAGe[:STARt]

N (see [page 794](#))

Command Syntax `:SAVE:IMAGe [:STARt] [<file_name>]`
`<file_name> ::= quoted ASCII string`

The :SAVE:IMAGe[:STARt] command saves an image.

NOTE

Be sure to set the :SAVE:IMAGe:FORMat before saving an image. If the format is NONE, the save image command will not succeed.

NOTE

If a file extension is provided as part of a specified <file_name>, and it does not match the extension expected by the format specified in :SAVE:IMAGe:FORMat, the format will be changed if the extension is a valid image file extension.

NOTE

If the extension ".bmp" is used and the current :SAVE:IMAGe:FORMat is not BMP or BMP8, the format will be changed to BMP.

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:IMAGe:FACTors](#)" on page 432
- "[":SAVE:IMAGe:FORMat](#)" on page 433
- "[":SAVE:IMAGe:INKSaver](#)" on page 434
- "[":SAVE:IMAGe:PALETTE](#)" on page 435
- "[":SAVE:FILEname](#)" on page 430

:SAVE:IMAGe:FACTors

N (see [page 794](#))

Command Syntax `:SAVE:IMAGe:FACTors <factors>`
`<factors> ::= {{OFF | 0} | {ON | 1}}`

The :SAVE:IMAGe:FACTors command controls whether the oscilloscope factors are output along with the image.

NOTE

Factors are written to a separate file with the same path and base name but with the ".txt" extension.

Query Syntax `:SAVE:IMAGe:FACTors?`

The :SAVE:IMAGe:FACTors? query returns a flag indicating whether oscilloscope factors are output along with the image.

Return Format `<factors><NL>`
`<factors> ::= {0 | 1}`

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:IMAGe\[:STARt\]](#)" on page 431
- "[":SAVE:IMAGe:FORMAT](#)" on page 433
- "[":SAVE:IMAGe:INKSaver](#)" on page 434
- "[":SAVE:IMAGe:PALETTE](#)" on page 435

:SAVE:IMAGe:FORMAT

N (see [page 794](#))

Command Syntax `:SAVE:IMAGe:FORMAT <format>`

`<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG}`

The :SAVE:IMAGe:FORMAT command sets the image format type.

Query Syntax `:SAVE:IMAGe:FORMAT?`

The :SAVE:IMAGe:FORMAT? query returns the selected image format type.

Return Format `<format><NL>`

`<format> ::= {BMP | BMP8 | PNG | NONE}`

When NONE is returned, it indicates that a waveform data file format is currently selected.

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:IMAGe\[:STARt\]](#)" on page 431
- "[":SAVE:IMAGe:FACTors](#)" on page 432
- "[":SAVE:IMAGe:INKSaver](#)" on page 434
- "[":SAVE:IMAGe:PALETTE](#)" on page 435
- "[":SAVE:WAVEform:FORMAT](#)" on page 442

:SAVE:IMAGe:INKSaver**N** (see [page 794](#))

Command Syntax `:SAVE:IMAGe:INKSaver <value>`
`<value> ::= {{OFF | 0} | {ON | 1}}`

The :SAVE:IMAGe:INKSaver command controls whether the graticule colors are inverted or not.

Query Syntax `:SAVE:IMAGe:INKSaver?`

The :SAVE:IMAGe:INKSaver? query returns a flag indicating whether graticule colors are inverted or not.

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

See Also

- ["Introduction to :SAVE Commands"](#) on page 429
- [":SAVE:IMAGe\[:STARt\]"](#) on page 431
- [":SAVE:IMAGe:FACTors"](#) on page 432
- [":SAVE:IMAGe:FORMAT"](#) on page 433
- [":SAVE:IMAGe:PAlette"](#) on page 435

:SAVE:IMAGe:PALETTE

N (see [page 794](#))

Command Syntax `:SAVE:IMAGe:PALETTE <palette>`
`<palette> ::= {COLOR | GRAYscale}`

The :SAVE:IMAGe:PALETTE command sets the image palette color.

Query Syntax `:SAVE:IMAGe:PALETTE?`

The :SAVE:IMAGe:PALETTE? query returns the selected image palette color.

Return Format `<palette><NL>`
`<palette> ::= {COL | GRAY}`

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:IMAGe\[:STARt\]](#)" on page 431
- "[":SAVE:IMAGe:FACTors](#)" on page 432
- "[":SAVE:IMAGe:FORMAT](#)" on page 433
- "[":SAVE:IMAGe:INKSaver](#)" on page 434

:SAVE:LISTER[:STARt]

N (see [page 794](#))

Command Syntax `:SAVE:LISTER[:STARt] [<file_name>]`
`<file_name> ::= quoted ASCII string`

The :SAVE:LISTER[:STARt] command saves the Lister display data to a file.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".csv".

-
- See Also**
- ["Introduction to :SAVE Commands" on page 429](#)
 - [":SAVE:FILEname" on page 430](#)
 - [Chapter 18, “:LISTER Commands,” starting on page 315](#)

:SAVE:MASK[:STARt]**N** (see [page 794](#))**Command Syntax** `:SAVE:MASK[:STARt] [<file_spec>]``<file_spec> ::= {<internal_loc> | <file_name>}``<internal_loc> ::= 0-3; an integer in NR1 format``<file_name> ::= quoted ASCII string`

The :SAVE:MASK[:STARt] command saves a mask.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".msk".

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[:SAVE:FILEname](#)" on page 430
- "[:RECall:MASK\[:STARt\]](#)" on page 422
- "[:MTEST:DATA](#)" on page 394

:SAVE:MULTi[:STARt]**N** (see [page 794](#))

Command Syntax `:SAVE:MULTi [:STARt] [<file_name>]`
`<file_name> ::= quoted ASCII string`

The :SAVE:MULTi[:STARt] command saves multi-channel waveform data to a file. This file can be opened by the N8900A InfiniiView oscilloscope analysis software.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".h5".

-
- See Also**
- "[Introduction to :SAVE Commands](#)" on page 429
 - "[":SAVE:FILENAME](#)" on page 430
 - "[":SAVE:PWD](#)" on page 439

:SAVE:PWD

N (see [page 794](#))

Command Syntax `:SAVE:PWD <path_name>`

`<path_name>` ::= quoted ASCII string

The :SAVE:PWD command sets the present working directory for save operations.

Query Syntax `:SAVE:PWD?`

The :SAVE:PWD? query returns the currently set working directory for save operations.

Return Format `<path_name><NL>`

`<path_name>` ::= quoted ASCII string

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:FILENAME](#)" on page 430
- "[":RECALL:PWD](#)" on page 423

:SAVE:SETup[:STARt]**N** (see [page 794](#))**Command Syntax** `:SAVE:SETup [:STARt] [<file_spec>]` `<file_spec> ::= {<internal_loc> | <file_name>}` `<internal_loc> ::= 0-9; an integer in NR1 format` `<file_name> ::= quoted ASCII string`

The :SAVE:SETup[:STARt] command saves an oscilloscope setup.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".scp".

See Also

- ["Introduction to :SAVE Commands"](#) on page 429
- [":SAVE:FILEname"](#) on page 430
- [":RECall:SETup\[:STARt\]"](#) on page 424

:SAVE:WAVEform[:STARt]**N** (see [page 794](#))

Command Syntax `:SAVE:WAVEform[:STARt] [<file_name>]`
`<file_name> ::= quoted ASCII string`

The :SAVE:WAVEform[:STARt] command saves oscilloscope waveform data to a file.

NOTE

Be sure to set the :SAVE:WAVEform:FORMAT before saving waveform data. If the format is NONE, the save waveform command will not succeed.

NOTE

If a file extension is provided as part of a specified <file_name>, and it does not match the extension expected by the format specified in :SAVE:WAVEform:FORMAT, the format will be changed if the extension is a valid waveform file extension.

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:WAVEform:FORMAT](#)" on page 442
- "[":SAVE:WAVEform:LENGTH](#)" on page 443
- "[":SAVE:FILENAME](#)" on page 430
- "[":RECALL:SETUP\[:STARt\]](#)" on page 424

:SAVE:WAVEform:FORMAT

N (see [page 794](#))

Command Syntax `:SAVE:WAVEform:FORMAT <format>`

`<format> ::= {ASCIixy | CSV | BINARY}`

The :SAVE:WAVEform:FORMAT command sets the waveform data format type:

- ASCIixy – creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is ".csv".
- CSV – creates one comma-separated value file that contains information for all analog channels that are displayed (turned on). The proper file extension for this format is ".csv".
- BINARY – creates an oscilloscope binary data format file. See the *User's Guide* for a description of this format. The proper file extension for this format is ".bin".

Query Syntax `:SAVE:WAVEform:FORMAT?`

The :SAVE:WAVEform:FORMAT? query returns the selected waveform data format type.

Return Format `<format><NL>`

`<format> ::= {ASC | CSV | BIN | NONE}`

When NONE is returned, it indicates that an image file format is currently selected.

See Also ["Introduction to :SAVE Commands"](#) on page 429

[":SAVE:WAVEform\[:START\]"](#) on page 441

[":SAVE:WAVEform:LENGTH"](#) on page 443

[":SAVE:IMAGE:FORMAT"](#) on page 433

:SAVE:WAVEform:LENGth

N (see [page 794](#))

Command Syntax `:SAVE:WAVEform:LENGth <length>`

`<length> ::= 100 to max. length; an integer in NR1 format`

When the :SAVE:WAVEform:LENGth:MAX setting is OFF, the :SAVE:WAVEform:LENGth command sets the waveform data length (that is, the number of points saved).

When the :SAVE:WAVEform:LENGth:MAX setting is ON, the :SAVE:WAVEform:LENGth setting has no effect.

Query Syntax `:SAVE:WAVEform:LENGth?`

The :SAVE:WAVEform:LENGth? query returns the current waveform data length setting.

Return Format `<length><NL>`

`<length> ::= 100 to max. length; an integer in NR1 format`

See Also

- "Introduction to :SAVE Commands" on page 429
- ":SAVE:WAVEform:LENGth:MAX" on page 444
- ":SAVE:WAVEform[:STARt]" on page 441
- ":WAVEform:POINTs" on page 642
- ":SAVE:WAVEform:FORMAT" on page 442

:SAVE:WAVEform:LENGth:MAX**N** (see [page 794](#))

Command Syntax `:SAVE:WAVEform:LENGth:MAX <setting>`
`<setting> ::= {{OFF | 0} | {ON | 1}}`

The :SAVE:WAVEform:LENGth:MAX command specifies whether maximum number of waveform data points is saved.

When OFF, the :SAVE:WAVEform:LENGth command specifies the number of waveform data points saved.

Query Syntax `:SAVE:WAVEform:LENGth:MAX?`

The :SAVE:WAVEform:LENGth:MAX? query returns the current setting.

Return Format `<setting><NL>`
`<setting> ::= {0 | 1}`

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:WAVEform\[:START\]](#)" on page 441
- "[":SAVE:WAVEform:LENGth](#)" on page 443

:SAVE:WAVeform:SEGmented

N (see [page 794](#))

Command Syntax `:SAVE:WAVeform:SEGmented <option>`
`<option> ::= {ALL | CURR}`

When segmented memory is used for acquisitions, the :SAVE:WAVeform:SEGmented command specifies which segments are included when the waveform is saved:

- ALL – all acquired segments are saved.
- CURR – only the currently selected segment is saved.

Query Syntax `:SAVE:WAVeform:SEGmented?`

The :SAVE:WAVeform:SEGmented? query returns the current segmented waveform save option setting.

Return Format `<option><NL>`
`<option> ::= {ALL | CURR}`

See Also ["Introduction to :SAVE Commands"](#) on page 429
[":SAVE:WAVeform\[:START\]"](#) on page 441
[":SAVE:WAVeform:FORMAT"](#) on page 442
[":SAVE:WAVeform:LENGTH"](#) on page 443

:SAVE:WMEMORY:SOURce

N (see [page 794](#))

Command Syntax `:SAVE:WMEMORY:SOURce <source>`

```
<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= {1 | 2}
```

The :SAVE:WMEMORY:SOURce command selects the source to be saved as a reference waveform file.

NOTE

Only ADD or SUBtract math operations can be saved as reference waveforms.

NOTE

MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

`:SAVE:WMEMORY:SOURce?`

The :SAVE:WMEMORY:SOURce? query returns the source to be saved as a reference waveform file.

Return Format `<source><NL>`

```
<source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}
```

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[":SAVE:WMEMORY\[:STARt\]" on page 447](#)
- "[":RECall:WMEMORY<r>\[:STARt\]" on page 425](#)

:SAVE:WMEMory[:STARt]

N (see [page 794](#))

Command Syntax `:SAVE:WMEMory [:STARt] [<file_name>]
<file_name> ::= quoted ASCII string`

The :SAVE:WMEMory[:STARt] command saves oscilloscope waveform data to a reference waveform file.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".h5".

See Also

- "[Introduction to :SAVE Commands](#)" on page 429
- "[:SAVE:WMEMory:SOURce](#)" on page 446
- "[:RECall:WMEMory<r>\[:STARt\]](#)" on page 425

25 :SBUS<n> Commands

Control the modes and parameters for each serial bus decode/trigger type. See:

- "[Introduction to :SBUS<n> Commands](#)" on page 449
- "[General :SBUS<n> Commands](#)" on page 451
- "[:SBUS<n>:CAN Commands](#)" on page 454
- "[:SBUS<n>:IIC Commands](#)" on page 471
- "[:SBUS<n>:LIN Commands](#)" on page 481
- "[:SBUS<n>:SPI Commands](#)" on page 495
- "[:SBUS<n>:UART Commands](#)" on page 511

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

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE

These commands are only valid on oscilloscope models when a serial decode option has been licensed.

The following serial bus decode/trigger types are available (see "[:TRIGger:MODE](#)" on page 600).

- **CAN (Controller Area Network) triggering**— will trigger on CAN version 2.0A and 2.0B signals. Setup consists of connecting the oscilloscope to a CAN signal. Baud rate, signal source, and signal polarity, and type of data to trigger on can be specified. You can trigger on CAN data and identifier patterns and you can set the bit sample point.
- **IIC (Inter-IC bus) triggering**— consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.
- **LIN (Local Interconnect Network) triggering**— will trigger on LIN sync break at the beginning of a message frame. You can trigger on Sync Break, Frame IDs, or Frame IDs and Data.

- **SPI (Serial Peripheral Interface) triggering**— consists of connecting the oscilloscope to a clock, data (MOSI or MISO), and framing signal. You can then trigger on a data pattern during a specific framing period. The serial data string can be specified to be from 4 to 64 bits long.
- **UART/RS-232 triggering** (with Option 232) – lets you trigger on RS-232 serial data.

NOTE

Two I₂S buses or two SPI buses cannot be decoded on both SBUS1 and SBUS2 at the same time.

Reporting the Setup

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

Return Format

The following is a sample response from the :SBUS1? query. In this case, the query was issued following a *RST command.

```
:SBUS1:DISP 0;MODE IIC;:SBUS1:IIC:ASIZ BIT7;:SBUS1:IIC:TRIG:TYPE  
STAR;QUAL EQU;:SBUS1:IIC:SOUR:CLOC CHAN1;DATA  
CHAN2;:SBUS1:IIC:TRIG:PATT:ADDR -1;DATA -1;DATA2 -1
```

General :SBUS<n> Commands

Table 75 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 452)	:SBUS<n>:DISPlay? (see page 452)	{0 1}
:SBUS<n>:MODE <mode> (see page 453)	:SBUS<n>:MODE? (see page 453)	<mode> ::= {CAN IIC LIN SPI UART}

:SBUS<n>:DISPlay

N (see [page 794](#))

Command Syntax `: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.

NOTE

Two I2S buses or two SPI buses cannot be decoded on both SBUS1 and SBUS2 at the same time.

Query Syntax

`:SBUS<n>:DISPlay?`

The :SBUS<n>:DISPlay? query returns the current display setting of the serial decode bus.

Return Format `<display><NL>`

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

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :SBUS<n> Commands" on page 449](#)

 • [":CHANnel<n>:DISPlay" on page 221](#)

 • [":DIGItal<d>:DISPlay" on page 245](#)

 • [":POD<n>:DISPlay" on page 415](#)

 • [":VIEW" on page 180](#)

 • [":BLANK" on page 156](#)

 • [":STATus" on page 177](#)

:SBUS<n>:MODE

N (see [page 794](#))

Command Syntax :SBUS<n>:MODE <mode>

<mode> ::= {CAN | IIC | LIN | SPI | UART}

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 Syntax :SBUS<n>:MODE?

The :SBUS<n>:MODE? query returns the current serial bus decode mode setting.

Return Format <mode><NL>

<mode> ::= {CAN | IIC | LIN | SPI | UART | NONE}

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :SBUS<n> Commands" on page 449](#)

• [":SBUS<n>:CAN Commands" on page 454](#)

• [":SBUS<n>:IIC Commands" on page 471](#)

• [":SBUS<n>:LIN Commands" on page 481](#)

• [":SBUS<n>:SPI Commands" on page 495](#)

• [":SBUS<n>:UART Commands" on page 511](#)

:SBUS<n>:CAN Commands

NOTE

These commands are valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Table 76 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNT:ER Ror? (see page 456)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:OV ERload? (see page 457)	<frame_count> ::= integer in NR1 format
:SBUS<n>:CAN:COUNT:RE Set (see page 458)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNT:TO Tal? (see page 459)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:UT ILization? (see page 460)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:SAMPLEpo int <value> (see page 461)	:SBUS<n>:CAN:SAMPLEpo int? (see page 461)	<value> ::= { 60 62.5 68 70 75 80 87.5 } in NR3 format
:SBUS<n>:CAN:SIGNAl:B AUDrate <baudrate> (see page 462)	:SBUS<n>:CAN:SIGNAl:B AUDrate? (see page 462)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNAl:D EFinition <value> (see page 463)	:SBUS<n>:CAN:SIGNAl:D EFinition? (see page 463)	<value> ::= { CANH CANL RX TX DIFFerential DIFL DIFH }
:SBUS<n>:CAN:SOURce <source> (see page 464)	:SBUS<n>:CAN:SOURce? (see page 464)	<source> ::= { CHANnel<n> EXTernal } <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 465)	:SBUS<n>:CAN:TRIGger? (see page 466)	<condition> ::= { SOF DATA ERRor IDData IDEither IDRemote ALLerrors OVERload ACKerror }
:SBUS<n>:CAN:TRIGger: PATTern:DATA <string> (see page 467)	:SBUS<n>:CAN:TRIGger: PATTern:DATA? (see page 467)	<string> ::= "nn...n" where n ::= { 0 1 X \$ } <string> ::= "0xnn...n" where n ::= { 0,...,9 A,...,F X \$ }

Table 76 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger: PATtern:DATA:LENGTH <length> (see page 468)	:SBUS<n>:CAN:TRIGger: PATtern:DATA:LENGTH? (see page 468)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger: PATtern:ID <string> (see page 469)	:SBUS<n>:CAN:TRIGger: PATtern:ID? (see page 469)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:CAN:TRIGger: PATtern:ID:MODE <value> (see page 470)	:SBUS<n>:CAN:TRIGger: PATtern:ID:MODE? (see page 470)	<value> ::= {STANDARD EXTENDED}

:SBUS<n>:CAN:COUNT:ERRor

N (see [page 794](#))

Query Syntax :SBUS<n>:CAN:COUNT:ERRor?

Returns the error frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing" on page 755](#)

See Also • [":SBUS<n>:CAN:COUNT:RESet" on page 458](#)
• ["Introduction to :SBUS<n> Commands" on page 449](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:CAN Commands" on page 454](#)

:SBUS<n>:CAN:COUNT:OVERload**N** (see [page 794](#))**Query Syntax** :SBUS<n>:CAN:COUNT:OVERload?

Returns the overload frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing" on page 755](#)**See Also** • [":SBUS<n>:CAN:COUNT:RESet" on page 458](#)
• ["Introduction to :SBUS<n> Commands" on page 449](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:CAN Commands" on page 454](#)

:SBUS<n>:CAN:COUNT:RESet**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:CAN:COUNT:RESet`

Resets the frame counters.

Errors - ["-241, Hardware missing" on page 755](#)**See Also** - [":SBUS<n>:CAN:COUNT:ERRor" on page 456](#)- [":SBUS<n>:CAN:COUNT:OVERload" on page 457](#)- [":SBUS<n>:CAN:COUNT:TOTal" on page 459](#)- [":SBUS<n>:CAN:COUNt:UTILization" on page 460](#)- ["Introduction to :SBUS<n> Commands" on page 449](#)- [":SBUS<n>:MODE" on page 453](#)- [":SBUS<n>:CAN Commands" on page 454](#)

:SBUS<n>:CAN:COUNT:TOTal**N** (see [page 794](#))**Query Syntax** :SBUS<n>:CAN:COUNT:TOTal?

Returns the total frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing" on page 755](#)**See Also** • [":SBUS<n>:CAN:COUNT:RESet" on page 458](#)
• ["Introduction to :SBUS<n> Commands" on page 449](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:CAN Commands" on page 454](#)

:SBUS<n>:CAN:COUNT:UTILization

N (see [page 794](#))

Query Syntax :SBUS<n>:CAN:COUNT:UTILization?

Returns the percent utilization.

Return Format <percent><NL>

<percent> ::= floating-point in NR3 format

Errors • ["-241, Hardware missing" on page 755](#)

See Also • [":SBUS<n>:CAN:COUNT:RESet" on page 458](#)
• ["Introduction to :SBUS<n> Commands" on page 449](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:CAN Commands" on page 454](#)

:SBUS<n>:CAN:SAMPLEpoint**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:CAN:SAMPLEpoint <value>` `<value><NL>` `<value> ::= { 60 | 62.5 | 68 | 70 | 75 | 80 | 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.

Query Syntax `:SBUS<n>:CAN:SAMPLEpoint?`

The :SBUS<n>:CAN:SAMPLEpoint? query returns the current CAN sample point setting.

Return Format `<value><NL>` `<value> ::= { 60 | 62.5 | 68 | 70 | 75 | 80 | 87.5 } in NR3 format`**See Also**

- "Introduction to :TRIGger Commands" on page 591

- ":SBUS<n>:MODE" on page 453

- ":SBUS<n>:CAN:TRIGger" on page 465

:SBUS<n>:CAN:SIGNAl:BAUDrate

N (see [page 794](#))

Command Syntax

```
:SBUS<n>:CAN:SIGNAl:BAUDrate <baudrate>
<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments,
               or 5000000
```

The :SBUS<n>:CAN:SIGNAl:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 4 Mb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

You can also set the baud rate of the CAN signal to 5 Mb/s. Fractional baud rates between 4 Mb/s and 5 Mb/s are not allowed.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:SBUS<n>:CAN:SIGNAl:BAUDrate?
```

The :SBUS<n>:CAN:SIGNAl:BAUDrate? query returns the current CAN baud rate setting.

Return Format

```
<baudrate><NL>
<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments,
               or 5000000
```

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:SBUS<n>:MODE](#)" on page 453
- "[:SBUS<n>:CAN:TRIGger](#)" on page 465
- "[:SBUS<n>:CAN:SIGNAl:DEFinition](#)" on page 463
- "[:SBUS<n>:CAN:SOURce](#)" on page 464

:SBUS<n>:CAN:SIGNAl:DEFinition

N (see [page 794](#))

Command Syntax `:SBUS<n>:CAN:SIGNAl:DEFinition <value>`
`<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}`

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.
- DIFH – the CAN differential (H-L) bus signal connected to an analog source channel using a differential probe.

Dominant low signals:

- CANL – the actual CAN_L differential bus signal.
- RX – the Receive signal from the CAN bus transceiver.
- TX – the Transmit signal to the CAN bus transceiver.
- 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.

Query Syntax `:SBUS<n>:CAN:SIGNAl:DEFinition?`

The :SBUS<n>:CAN:SIGNAl:DEFinition? query returns the current CAN signal type.

Return Format `<value><NL>`
`<value> ::= {CANH | CANL | RX | TX | DIFL | DIFH}`

See Also ["Introduction to :TRIGger Commands" on page 591](#)
[":SBUS<n>:MODE" on page 453](#)
[":SBUS<n>:CAN:SIGNAl:BAUDrate" on page 462](#)
[":SBUS<n>:CAN:SOURce" on page 464](#)
[":SBUS<n>:CAN:TRIGger" on page 465](#)

:SBUS<n>:CAN:SOURce**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:CAN:SOURce <source>` `<source> ::= {CHANnel<n> | EXTernal}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :SBUS<n>:CAN:SOURce command sets the source for the CAN signal.

Query Syntax `:SBUS<n>:CAN:SOURce?`

The :SBUS<n>:CAN:SOURce? query returns the current source for the CAN signal.

Return Format `<source><NL>`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:SBUS<n>:MODE](#)" on page 453
 - "[:SBUS<n>:CAN:TRIGger](#)" on page 465
 - "[:SBUS<n>:CAN:SIGNal:DEFinition](#)" on page 463

:SBUS<n>:CAN:TRIGger

N (see [page 794](#))

Command Syntax :SBUS<n>:CAN:TRIGger <condition>

```
<condition> ::= {SOF | DATA | ERROR | IDData | IDEither | IDRremote |
    ALLerrors | OVERload | ACKerror}
```

The :SBUS<n>:CAN:TRIGger command sets the CAN trigger on condition:

- SOF - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- DATA - will trigger on CAN Data frames matching the specified Id, Data, and the DLC (Data length code).
- ERROR - will trigger on CAN Error frame.
- IDData - will trigger on CAN frames matching the specified Id of a Data frame.
- IDEither - will trigger on the specified Id, regardless if it is a Remote frame or a Data frame.
- IDRremote - will trigger on CAN frames matching the specified Id of a Remote frame.
- ALLerrors - will trigger on CAN active error frames and unknown bus conditions.
- OVERload - will trigger on CAN overload frames.
- ACKerror - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

Remote <condition> parameter	Front-panel Trigger on: softkey selection (softkey text - softkey popup text)
SOF	SOF - Start of Frame
DATA	ID & Data - Data Frame ID and Data
ERROR	Error - Error frame
IDData	ID & ~RTR - Data Frame ID (~RTR)
IDEither	ID - Remote or Data Frame ID
IDRremote	ID & RTR - Remote Frame ID (RTR)
ALLerrors	All Errors - All Errors
OVERload	Overload - Overload Frame
ACKerror	Ack Error - Acknowledge Error

CAN Id specification is set by the :SBUS<n>:CAN:TRIGger:PATTern:ID and :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE commands.

CAN Data specification is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA command.

CAN Data Length Code is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command.

Query Syntax :SBUS<n>:CAN:TRIGger?

The :SBUS<n>:CAN:TRIGger? query returns the current CAN trigger on condition.

Return Format <condition><NL>

<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}

- Errors** · ["-241, Hardware missing"](#) on page 755

See Also · ["Introduction to :SBUS<n> Commands"](#) on page 449

- [":SBUS<n>:MODE"](#) on page 453
- [":SBUS<n>:CAN:TRIGger:PATTern:DATA"](#) on page 467
- [":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth"](#) on page 468
- [":SBUS<n>:CAN:TRIGger:PATTern:ID"](#) on page 469
- [":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE"](#) on page 470
- [":SBUS<n>:CAN:SIGNAL:DEFinition"](#) on page 463
- [":SBUS<n>:CAN:SOURce"](#) on page 464

:SBUS<n>:CAN:TRIGger:PATTERn:DATA

N (see [page 794](#))

Command Syntax :SBUS<n>:CAN:TRIGger:PATTERn:DATA <string>

```
<string> ::= "nn...n" where n ::= {0 | 1 | x | $}
<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | x | $}
```

The :SBUS<n>:CAN:TRIGger:PATTERn:DATA command defines the CAN data pattern resource according to the string parameter. This pattern, along with the data length (set by the :SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth command), control the data pattern searched for in each CAN message.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

If more bits are sent for <string> than specified by the :SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth command, the most significant bits will be truncated. If the data length is changed after the <string> is programmed, the added or deleted bits will be added to or deleted from the least significant bits.

Query Syntax :SBUS<n>:CAN:TRIGger:PATTERn:DATA?

The :SBUS<n>:CAN:TRIGger:PATTERn:DATA? query returns the current settings of the specified CAN data pattern resource in the binary string format.

Return Format <string><NL> in nondecimal format

Errors

- ["-241, Hardware missing" on page 755](#)

See Also

- ["Introduction to :TRIGger Commands" on page 591](#)
- [":SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth" on page 468](#)
- [":SBUS<n>:CAN:TRIGger:PATTERn:ID" on page 469](#)

:SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth <length>` `<length> ::= integer from 1 to 8 in NR1 format`

The :SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string. The number of bytes in the string can be anywhere from 1 bytes to 8 bytes (64 bits). The value for these bytes is set by the :SBUS<n>:CAN:TRIGger:PATTERn:DATA command.

Query Syntax `:SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth?`

The :SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth? query returns the current CAN data pattern length setting.

Return Format `<count><NL>` `<count> ::= integer from 1 to 8 in NR1 format`**Errors** • ["-241, Hardware missing" on page 755](#)**See Also** • ["Introduction to :TRIGger Commands" on page 591](#)
 • [":SBUS<n>:CAN:TRIGger:PATTERn:DATA" on page 467](#)
 • [":SBUS<n>:CAN:SOURce" on page 464](#)

:SBUS<n>:CAN:TRIGger:PATTern:ID

N (see [page 794](#))

Command Syntax :SBUS<n>:CAN:TRIGger:PATTern:ID <string>

```
<string> ::= "nn...n" where n ::= {0 | 1 | x | $}
<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | x | $}
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID command defines the CAN identifier pattern resource according to the string parameter. This pattern, along with the identifier mode (set by the :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE command), control the identifier pattern searched for in each CAN message.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

The ID pattern resource string is always 29 bits. Only 11 of these bits are used when the :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE is STANDARD.

A string longer than 29 bits is truncated to 29 bits when setting the ID pattern resource.

Query Syntax :SBUS<n>:CAN:TRIGger:PATTern:ID?

The :SBUS<n>:CAN:TRIGger:PATTern:ID? query returns the current settings of the specified CAN identifier pattern resource in the 29-bit binary string format.

Return Format <string><NL> in 29-bit binary string format

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :TRIGger Commands" on page 591](#)
 • [":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 470](#)
 • [":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 467](#)

:SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE

N (see [page 794](#))

Command Syntax `:SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE <value>`
`<value> ::= {STANDARD | EXTENDED}`

The :SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE command sets the CAN identifier mode. STANDARD selects the standard 11-bit identifier. EXTENDED selects the extended 29-bit identifier. The CAN identifier is set by the :SBUS<n>:CAN:TRIGger:PATTERn:ID command.

Query Syntax `:SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE?`

The :SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE? query returns the current setting of the CAN identifier mode.

Return Format `<value><NL>`

`<value> ::= {STAN | EXT}`

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :TRIGger Commands" on page 591](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:CAN:TRIGger:PATTERn:DATA" on page 467](#)
• [":SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGTH" on page 468](#)
• [":SBUS<n>:CAN:TRIGger:PATTERn:ID" on page 469](#)

:SBUS<n>:IIC Commands

NOTE

These commands are only valid when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Table 77 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZE <size> (see page 472)	:SBUS<n>:IIC:ASIZE? (see page 472)	<size> ::= {BIT7 BIT8}
:SBUS<n>:IIC[:SOURce] :CLOCk <source> (see page 473)	:SBUS<n>:IIC[:SOURce] :CLOCk? (see page 473)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC[:SOURce] :DATA <source> (see page 474)	:SBUS<n>:IIC[:SOURce] :DATA? (see page 474)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC:TRIGger: PATtern:ADDRess <value> (see page 475)	:SBUS<n>:IIC:TRIGger: PATtern:ADDRess? (see page 475)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGGER: PATtern:DATA <value> (see page 476)	:SBUS<n>:IIC:TRIGger: PATtern:DATA? (see page 476)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATtern:DATA2 <value> (see page 477)	:SBUS<n>:IIC:TRIGger: PATtern:DATA2? (see page 477)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: QUALifier <value> (see page 478)	:SBUS<n>:IIC:TRIGger: QUALifier? (see page 478)	<value> ::= {EQUAL NOTEqual LESSthan GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 479)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 479)	<type> ::= {START STOP READ7 READEprom WRITE7 WRITE10 NACKnowledge ANACK R7Data2 W7Data2 REStart}

:SBUS<n>:IIC:ASIZE**N** (see [page 794](#))

Command Syntax `: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 Syntax `:SBUS<n>:IIC:ASIZE?`

The :SBUS<n>:IIC:ASIZE? query returns the current IIC address width setting.

Return Format `<mode><NL>`
 `<mode> ::= {BIT7 | BIT8}`

Errors - ["-241, Hardware missing" on page 755](#)

See Also - ["Introduction to :SBUS<n> Commands" on page 449](#)
 - [":SBUS<n>:IIC Commands" on page 471](#)

:SBUS<n>:IIC[:SOURce]:CLOCK

N (see [page 794](#))

Command Syntax :SBUS<n>:IIC[:SOURce]:CLOCK <source>

<source> ::= {CHANnel<n> | EXTERNAL}

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

The :SBUS<n>:IIC[:SOURce]:CLOCK command sets the source for the IIC serial clock (SCL).

Query Syntax :SBUS<n>:IIC[:SOURce]:CLOCK?

The :SBUS<n>:IIC[:SOURce]:CLOCK? query returns the current source for the IIC serial clock.

Return Format <source><NL>

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:SBUS<n>:IIC\[:SOURce\]:DATA](#)" on page 474

:SBUS<n>:IIC[:SOURce]:DATA**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:IIC[:SOURce]:DATA <source>` `<source> ::= {CHANnel<n> | EXTERNAL}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :SBUS<n>:IIC[:SOURce]:DATA command sets the source for IIC serial data (SDA).

Query Syntax `:SBUS<n>:IIC[:SOURce]:DATA?`

The :SBUS<n>:IIC[:SOURce]:DATA? query returns the current source for IIC serial data.

Return Format `<source><NL>`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:SBUS<n>:IIC\[:SOURce\]:CLOCK](#)" on page 473

:SBUS<n>:IIC:TRIGger:PATTERn:ADDResS

N (see [page 794](#))

Command Syntax :SBUS<n>:IIC:TRIGger:PATTERn:ADDResS <value>

<value> ::= integer or <string>

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

The :SBUS<n>:IIC:TRIGger:PATTERn:ADDResS command sets the address for IIC data. The address can range from 0x00 to 0x7F (7-bit) or 0x3FF (10-bit) hexadecimal. Use the don't care address (-1 or 0xFFFFFFFF) to ignore the address value.

Query Syntax :SBUS<n>:IIC:TRIGger:PATTERn:ADDResS?

The :SBUS<n>:IIC:TRIGger:PATTERn:ADDResS? query returns the current address for IIC data.

Return Format <value><NL>

<value> ::= integer

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[":SBUS<n>:IIC:TRIGger:PATTERn:DATA](#)" on page 476
- "[":SBUS<n>:IIC:TRIGger:PATTERn:DATa2](#)" on page 477
- "[":SBUS<n>:IIC:TRIGger\[:TYPE\]](#)" on page 479

:SBUS<n>:IIC:TRIGger:PATTERn:DATA**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:IIC:TRIGger:PATTERn:DATA <value>` `<value> ::= integer or <string>` `<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F}`

The :SBUS<n>:IIC:TRIGger:PATTERn:DATA command sets IIC data. The data value can range from 0x00 to 0xFF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax `:SBUS<n>:IIC:TRIGger:PATTERn:DATA?`

The :SBUS<n>:IIC:TRIGger:PATTERn:DATA? query returns the current pattern for IIC data.

Return Format `<value><NL>`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:SBUS<n>:IIC:TRIGger:PATTERn:ADDResS](#)" on page 475
- "[:SBUS<n>:IIC:TRIGger:PATTERn:DATa2](#)" on page 477
- "[:SBUS<n>:IIC:TRIGger\[:TYPE\]](#)" on page 479

:SBUS<n>:IIC:TRIGger:PATTERn:DATA2

N (see [page 794](#))

Command Syntax :SBUS<n>:IIC:TRIGger:PATTERn:DATA2 <value>

<value> ::= integer or <string>

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

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

Query Syntax :SBUS<n>:IIC:TRIGger:PATTERn:DATA2?

The :SBUS<n>:IIC:TRIGger:PATTERn:DATA2? query returns the current pattern for IIC data 2.

Return Format <value><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:SBUS<n>:IIC:TRIGger:PATTERn:ADDResS](#)" on page 475
 - "[:SBUS<n>:IIC:TRIGger:PATTERn:DATA](#)" on page 476
 - "[:SBUS<n>:IIC:TRIGger\[:TYPE\]](#)" on page 479

:SBUS<n>:IIC:TRIGger:QUALifier**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:IIC:TRIGger:QUALifier <value>``<value> ::= {EQUAL | NOTEQUAL | LESSthan | GREATERthan}`

The :SBUS<n>:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEprom.

Query Syntax `:SBUS<n>:IIC:TRIGger:QUALifier?`

The :SBUS<n>:IIC:TRIGger:QUALifier? query returns the current IIC data qualifier value.

Return Format `<value><NL>``<value> ::= {EQUAL | NOTEQUAL | LESSthan | GREATERthan}`**See Also**

- "Introduction to :TRIGger Commands" on page 591
- ":TRIGger:MODE" on page 600
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 479

:SBUS<n>:IIC:TRIGger[:TYPE]

N (see [page 794](#))

Command Syntax `:SBUS<n>:IIC:TRIGger[:TYPE] <value>`

```
<value> ::= {START | STOP | READ7 | READEprom | WRITe7 | WRITe10
| NACKnowledge | ANACK | R7Data2 | W7Data2 | RESTart}
```

The :SBUS<n>:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- START – Start condition.
- STOP – Stop condition.
- READ7 – 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- R7Data2 – 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- READEprom – EEPROM data read.
- WRITe7 – 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- W7Data2 – 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).
- WRITe10 – 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).
- NACKnowledge – Missing acknowledge.
- ANACK – Address with no acknowledge.
- RESTart – Another start condition occurs before a stop condition.

NOTE

The short form of READ7 (READ7), READEprom (READE), WRITe7 (WRIT7), and WRITe10 (WRIT10) do not follow the defined Long Form to Short Form Truncation Rules (see [page 796](#)).

Query Syntax `:SBUS<n>:IIC:TRIGger[:TYPE] ?`

The :SBUS<n>:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

Return Format `<value><NL>`

```
<value> ::= {STAR | STOP | READ7 | READE | WRIT7 | WRIT10 | NACK | ANAC
| R7D2 | W7D2 | REST}
```

See Also ["Introduction to :TRIGger Commands"](#) on page 591

- [":TRIGger:MODE"](#) on page 600
- [":SBUS<n>:IIC:TRIGger:PATTERn:ADDResS"](#) on page 475
- [":SBUS<n>:IIC:TRIGger:PATTERn:DATA"](#) on page 476

- "[:SBUS<n>:IIC:TRIGger:PATTERn:DATa2](#)" on page 477
- "[:SBUS<n>:IIC:TRIGger:QUALifier](#)" on page 478
- "["Long Form to Short Form Truncation Rules"](#) on page 796

:SBUS<n>:LIN Commands

NOTE

These commands are valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Table 78 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:PARity { {0 OFF} {1 ON} } (see page 483)	:SBUS<n>:LIN:PARity? (see page 483)	{0 1}
:SBUS<n>:LIN:SAMPLEpo int <value> (see page 484)	:SBUS<n>:LIN:SAMPLEpo int? (see page 484)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:LIN:SIGNAL:B AUDrate <baudrate> (see page 485)	:SBUS<n>:LIN:SIGNAL:B AUDrate? (see page 485)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 486)	:SBUS<n>:LIN:SOURce? (see page 486)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:LIN:STANDARD <std> (see page 487)	:SBUS<n>:LIN:STANDARD ? (see page 487)	<std> ::= {LIN13 LIN20}
:SBUS<n>:LIN:SYNCbrea k <value> (see page 488)	:SBUS<n>:LIN:SYNCbrea k? (see page 488)	<value> ::= integer = {11 12 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 489)	:SBUS<n>:LIN:TRIGger? (see page 489)	<condition> ::= {SYNCbreak ID DATA}
:SBUS<n>:LIN:TRIGger: ID <value> (see page 490)	:SBUS<n>:LIN:TRIGger: ID? (see page 490)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal

Table 78 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger: PATtern:DATA <string> (see page 491)	:SBUS<n>:LIN:TRIGger: PATtern:DATA? (see page 491)	<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when <base> = DECimal <string> ::= "nn...n" where n ::= {0 1 X \$} when <base> = BINary <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$} when <base> = HEX
:SBUS<n>:LIN:TRIGger: PATtern:DATA:LENGTH <length> (see page 493)	:SBUS<n>:LIN:TRIGger: PATtern:DATA:LENGTH? (see page 493)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger: PATtern:FORMAT <base> (see page 494)	:SBUS<n>:LIN:TRIGger: PATtern:FORMAT? (see page 494)	<base> ::= {BINary HEX DECimal}

:SBUS<n>:LIN:PARity**N** (see [page 794](#))

Command Syntax `:SBUS<n>:LIN:PARity <display>`
`<display> ::= {{1 | ON} | {0 | OFF}}`

The :SBUS<n>:LIN:PARity command determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.

Query Syntax `:SBUS<n>:LIN:PARity?`

The :SBUS<n>:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

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

Errors • ["-241, Hardware missing"](#) on page 755

See Also • ["Introduction to :SBUS<n> Commands"](#) on page 449
• [":SBUS<n>:LIN Commands"](#) on page 481

:SBUS<n>:LIN:SAMPLEpoint**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:LIN:SAMPLEpoint <value>` `<value><NL>` `<value> ::= { 60 | 62.5 | 68 | 70 | 75 | 80 | 87.5 } in NR3 format`

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

NOTE

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

Query Syntax `:SBUS<n>:LIN:SAMPLEpoint?`

The :SBUS<n>:LIN:SAMPLEpoint? query returns the current LIN sample point setting.

Return Format `<value><NL>` `<value> ::= { 60 | 62.5 | 68 | 70 | 75 | 80 | 87.5 } in NR3 format`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:LIN:TRIGger](#)" on page 489

:SBUS<n>:LIN:SIGNAl:BAUDrate

N (see [page 794](#))

Command Syntax `:SBUS<n>:LIN:SIGNAl:BAUDrate <baudrate>`
`<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments`

The :SBUS<n>:LIN:SIGNAl:BAUDrate command sets the standard baud rate of the LIN signal from 2400 b/s to 625 kb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

Query Syntax `:SBUS<n>:LIN:SIGNAl:BAUDrate?`

The :SBUS<n>:LIN:SIGNAl:BAUDrate? query returns the current LIN baud rate setting.

Return Format `<baudrate><NL>`
`<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments`

See Also

- ["Introduction to :TRIGger Commands"](#) on page 591
- [":TRIGger:MODE"](#) on page 600
- [":SBUS<n>:LIN:TRIGger"](#) on page 489
- [":SBUS<n>:LIN:SOURce"](#) on page 486

:SBUS<n>:LIN:SOURce**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:LIN:SOURCE <source>` `<source> ::= {CHANnel<n> | EXTERNAL}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :SBUS<n>:LIN:SOURce command sets the source for the LIN signal.

Query Syntax `:SBUS<n>:LIN:SOURCE?`

The :SBUS<n>:LIN:SOURce? query returns the current source for the LIN signal.

Return Format `<source><NL>`

- See Also**
- "Introduction to :TRIGger Commands" on page 591
 - ":TRIGger:MODE" on page 600
 - ":SBUS<n>:LIN:TRIGger" on page 489

:SBUS<n>:LIN:STANDARD

N (see [page 794](#))

Command Syntax `:SBUS<n>:LIN:STANDARD <std>`
 `<std> ::= {LIN13 | LIN20}`

The :SBUS<n>:LIN:STANDARD command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

Query Syntax `:SBUS<n>:LIN:STANDARD?`

The :SBUS<n>:LIN:STANDARD? query returns the current LIN standard setting.

Return Format `<std><NL>`
 `<std> ::= {LIN13 | LIN20}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:LIN:SOURce](#)" on page 486

:SBUS<n>:LIN:SYNCbreak**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:LIN:SYNCbreak <value>``<value> ::= integer = {11 | 12 | 13}`

The :SBUS<n>:LIN:SYNCbreak command sets the length of the LIN sync break to be greater than or equal to 11, 12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.

Query Syntax `:SBUS<n>:LIN:SYNCbreak?`

The :SBUS<n>:LIN:SYNCbreak? query returns the current LIN sync break setting.

Return Format `<value><NL>``<value> ::= {11 | 12 | 13}`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:LIN:SOURce](#)" on page 486

:SBUS<n>:LIN:TRIGger

N (see [page 794](#))

Command Syntax `:SBUS<n>:LIN:TRIGger <condition>`
`<condition> ::= {SYNCbreak | ID | DATA}`

The :SBUS<n>:LIN:TRIGger command sets the LIN trigger condition to be:

- SYNCbreak – Sync Break.
- ID – Frame ID.
 Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.
- DATA – Frame ID and Data.
 Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.
 Use the :SBUS<n>:LIN:TRIGger:PATtern:DATA:LENGTH and
 :SBUS<n>:LIN:TRIGger:PATtern:DATA commands to specify the data string
 length and value.

Query Syntax `:SBUS<n>:LIN:TRIGger?`

The :SBUS<n>:LIN:TRIGger? query returns the current LIN trigger value.

Return Format `<condition><NL>`
`<condition> ::= {SYNC | ID | DATA}`

- Errors** • "[-241, Hardware missing](#)" on page 755

See Also • "[Introduction to :TRIGger Commands](#)" on page 591
 • "[:TRIGger:MODE](#)" on page 600
 • "[:SBUS<n>:LIN:TRIGger:ID](#)" on page 490
 • "[:SBUS<n>:LIN:TRIGger:PATtern:DATA:LENGth](#)" on page 493
 • "[:SBUS<n>:LIN:TRIGger:PATtern:DATA](#)" on page 491
 • "[:SBUS<n>:LIN:SOURce](#)" on page 486

:SBUS<n>:LIN:TRIGger:ID

N (see [page 794](#))

Command Syntax `:SBUS<n>:LIN:TRIGger:ID <value>`

```
<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>
           from 0-63 or 0x00-0x3f

<nondecimal> ::= #Hnn where n ::= {0,...,9 | A,...,F} for hexadecimal
<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F} for hexadecimal
```

The :SBUS<n>:LIN:TRIGger:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

Setting the ID to a value of "-1" results in "0xXX" which is equivalent to all IDs.

Query Syntax `:SBUS<n>:LIN:TRIGger:ID?`

The :SBUS<n>:LIN:TRIGger:ID? query returns the current LIN identifier setting.

Return Format `<value><NL>`

```
<value> ::= integer in decimal
```

Errors • ["-241, Hardware missing"](#) on page 755

See Also • ["Introduction to :TRIGger Commands"](#) on page 591
 • [":TRIGger:MODE"](#) on page 600
 • [":SBUS<n>:LIN:TRIGger"](#) on page 489
 • [":SBUS<n>:LIN:SOURce"](#) on page 486

:SBUS<n>:LIN:TRIGger:PATTERn:DATA

N (see [page 794](#))

Command Syntax :SBUS<n>:LIN:TRIGger:PATTERn:DATA <string>

```
<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when
           <base> = DECimal

<string> ::= "nn...n" where n ::= {0 | 1 | x | $} when
           <base> = BINary

<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | x | $} when
           <base> = HEX
```

NOTE

<base> is specified with the :SBUS<n>:LIN:TRIGger:PATTERn:FORMat command. The default <base> is BINary.

The :SBUS<n>:LIN:TRIGger:PATTERn:DATA command specifies the LIN trigger data pattern searched for in each LIN data field.

Set a <string> bit to "0" or "1" to set the corresponding bit in the data pattern to low or high, respectively.

Set a <string> bit to "X" to ignore (mask off) that bit in the data pattern.

Use the "\$" character to indicate that the value of the corresponding bit will not be changed (the existing bit value is used).

When <base> = DECimal, the "X" and "\$" characters cannot be entered. When queried, the "\$" character is returned when any bits in the pattern have the value of "X" and <base> = DECimal. When any bits in a given nibble have the value of "X" and <base> = HEX, the "\$" character is returned for the corresponding nibble.

NOTE

The length of the trigger data value is determined by the :SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGTH command.

NOTE

If more bits are sent for <string> than the specified trigger pattern data length, the most significant bits will be truncated. If the data length size is changed after the <string> is programmed, the added or deleted bits will be added to or deleted from the least significant bits.

Query Syntax

:SBUS<n>:LIN:TRIGger:PATTERn:DATA?

The :SBUS<n>:LIN:TRIGger:PATTERn:DATA? query returns the currently specified LIN trigger data pattern.

Return Format

<string><NL>

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:SBUS<n>:LIN:TRIGger:FORMAT](#)" on page 494
- "[:SBUS<n>:LIN:TRIGger](#)" on page 489
- "[:SBUS<n>:LIN:TRIGger:PATtern:DATA:LENGTH](#)" on page 493

:SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth

N (see [page 794](#))

Command Syntax :SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format

The :SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth command sets the number of 8-bit bytes in the LIN data string. The number of bytes in the string can be anywhere from 1 bytes to 8 bytes (64 bits). The value for these bytes is set by the :SBUS<n>:LIN:TRIGger:PATTERn:DATA command.

Query Syntax :SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth?

The :SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth? query returns the current LIN data pattern length setting.

Return Format <count><NL>

<count> ::= integer from 1 to 8 in NR1 format

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :TRIGger Commands" on page 591](#)
 • [":SBUS<n>:LIN:TRIGger:PATTERn:DATA" on page 491](#)
 • [":SBUS<n>:LIN:SOURce" on page 486](#)

:SBUS<n>:LIN:TRIGger:PATTERn:FORMAT**N** (see [page 794](#))

Command Syntax `:SBUS<n>:LIN:TRIGger:PATTERn:FORMAT <base>`
 `<base> ::= {BINary | HEX | DECimal}`

The :SBUS<n>:LIN:TRIGger:PATTERn:FORMAT command sets the entry (and query) number base used by the :SBUS<n>:LIN:TRIGger:PATTERn:DATA command. The default <base> is BINary.

Query Syntax `:SBUS<n>:LIN:TRIGger:PATTERn:FORMAT?`

The :SBUS<n>:LIN:TRIGger:PATTERn:FORMAT? query returns the currently set number base for LIN pattern data.

Return Format `<base><NL>`
 `<base> ::= {BIN | HEX | DEC}`

See Also

- "Introduction to :TRIGger Commands" on page 591
- ":SBUS<n>:LIN:TRIGger:PATTERn:DATA" on page 491
- ":SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth" on page 493

:SBUS<n>:SPI Commands

NOTE

These commands are only valid when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Table 79 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 497)	:SBUS<n>:SPI:BITorder? (see page 497)	<order> ::= {LSBFFirst MSBFFirst}
:SBUS<n>:SPI:CLOCK:SLOPe <slope> (see page 498)	:SBUS<n>:SPI:CLOCK:SLOPe? (see page 498)	<slope> ::= {NEGative POSitive}
:SBUS<n>:SPI:CLOCK:TIMEout <time_value> (see page 499)	:SBUS<n>:SPI:CLOCK:TIMEout? (see page 499)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:FRAMing <value> (see page 500)	:SBUS<n>:SPI:FRAMing? (see page 500)	<value> ::= {CHIPselect {NCHipselect NOTC} TIMEout}
:SBUS<n>:SPI:SOURce:LOCK <source> (see page 501)	:SBUS<n>:SPI:SOURce:LOCK? (see page 501)	<value> ::= {CHANnel<n> EXTERNAL} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:SOURCE:FRAME <source> (see page 502)	:SBUS<n>:SPI:SOURCE:FRAME? (see page 502)	<value> ::= {CHANnel<n> EXTERNAL} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:SOURCE:MISO <source> (see page 503)	:SBUS<n>:SPI:SOURCE:MISO? (see page 503)	<value> ::= {CHANnel<n> EXTERNAL} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:SOURCE:MOXI <source> (see page 504)	:SBUS<n>:SPI:SOURCE:MOXI? (see page 504)	<value> ::= {CHANnel<n> EXTERNAL} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SPI:TRIGGER:PATTERn:MISO:DATA <string> (see page 505)	:SBUS<n>:SPI:TRIGGER:PATTERn:MISO:DATA? (see page 505)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:SPI:TRIGGER:PATTERn:MISO:WIDTH <width> (see page 506)	:SBUS<n>:SPI:TRIGGER:PATTERn:MISO:WIDTH? (see page 506)	<width> ::= integer from 4 to 64 in NR1 format

Table 79 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:TRIGger: PATtern:MOSI:DATA <string> (see page 507)	:SBUS<n>:SPI:TRIGger: PATtern:MOSI:DATA? (see page 507)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:SPI:TRIGger: PATtern:MOSI:WIDTH <width> (see page 508)	:SBUS<n>:SPI:TRIGger: PATtern:MOSI:WIDTH? (see page 508)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGGER: TYPE <value> (see page 509)	:SBUS<n>:SPI:TRIGger: TYPE? (see page 509)	<value> ::= {MOSI MISO}
:SBUS<n>:SPI:WIDTH <word_width> (see page 510)	:SBUS<n>:SPI:WIDTH? (see page 510)	<word_width> ::= integer 4-16 in NR1 format

:SBUS<n>:SPI:BITorder

N (see [page 794](#))

Command Syntax `:SBUS<n>:SPI:BITorder <order>`

`<order> ::= {LSBFFirst | MSBFFirst}`

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.

Query Syntax `:SBUS<n>:SPI:BITorder?`

The :SBUS<n>:SPI:BITorder? query returns the current SPI decode bit order.

Return Format `<order><NL>`

`<order> ::= {LSBF | MSBF}`

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :SBUS<n> Commands" on page 449](#)

 • [":SBUS<n>:MODE" on page 453](#)

 • [":SBUS<n>:SPI Commands" on page 495](#)

:SBUS<n>:SPI:CLOCK:SLOPe**N** (see [page 794](#))

Command Syntax `:SBUS<n>:SPI:CLOCK:SLOPe <slope>`
 `<slope> ::= {NEGative | POSitive}`

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.

Query Syntax `:SBUS<n>:SPI:CLOCK:SLOPe?`

The :SBUS<n>:SPI:CLOCK:SLOPe? query returns the current SPI clock source slope.

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

See Also

- "Introduction to :TRIGger Commands" on page 591
- ":SBUS<n>:SPI:CLOCK:TIMEout" on page 499
- ":SBUS<n>:SPI:SOURce:CLOCK" on page 501

:SBUS<n>:SPI:CLOCK:TIMEout

N (see [page 794](#))

Command Syntax	<code>:SBUS<n>:SPI:CLOCK:TIMEout <time_value></code> <code><time_value> ::= time in seconds in NR3 format</code>
	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.
Query Syntax	<code>:SBUS<n>:SPI:CLOCK:TIMEout?</code>
	The :SBUS<n>:SPI:CLOCK:TIMEout? query returns current SPI clock timeout setting.
Return Format	<code><time value><NL></code> <code><time_value> ::= time in seconds in NR3 format</code>
See Also	<ul style="list-style-type: none"> • "Introduction to :TRIGger Commands" on page 591 • ":SBUS<n>:SPI:CLOCK:SLOPe" on page 498 • ":SBUS<n>:SPI:SOURce:CLOCK" on page 501 • ":SBUS<n>:SPI:FRAMing" on page 500

:SBUS<n>:SPI:FRAMing

N (see [page 794](#))

Command Syntax `:SBUS<n>:SPI:FRAMing <value>`

`<value> ::= {CHIPselect | {NCHipselect | NOTC} | TMeout}`

The :SBUS<n>:SPI:FRAMing command sets the SPI trigger framing value. If TMeout is selected, the timeout value is set by the :SBUS<n>:SPI:CLOCK:TMeout command.

NOTE

The NOTC value is deprecated. It is the same as NCHipselect.

Query Syntax `:SBUS<n>:SPI:FRAMing?`

The :SBUS<n>:SPI:FRAMing? query returns the current SPI framing value.

Return Format `<value><NL>`

`<value> ::= {CHIP | NCH | TIM}`

See Also

- "Introduction to :TRIGger Commands" on page 591

- "":TRIGger:MODE" on page 600

- "":SBUS<n>:SPI:CLOCK:TMeout" on page 499

- "":SBUS<n>:SPI:SOURce:FRAMe" on page 502

:SBUS<n>:SPI:SOURce:CLOCK

N (see [page 794](#))

Command Syntax :SBUS<n>:SPI:SOURce:CLOCK <source>

<source> ::= {CHANnel<n> | EXTERNAL}

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

The :SBUS<n>:SPI:SOURce:CLOCK command sets the source for the SPI serial clock.

Query Syntax :SBUS<n>:SPI:SOURce:CLOCK?

The :SBUS<n>:SPI:SOURce:CLOCK? query returns the current source for the SPI serial clock.

Return Format <source><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:SBUS<n>:SPI:CLOCK:SLOPe](#)" on page 498
 - "[:SBUS<n>:SPI:CLOCK:TIMEout](#)" on page 499
 - "[:SBUS<n>:SPI:SOURce:FRAMe](#)" on page 502
 - "[:SBUS<n>:SPI:SOURce:MOSI](#)" on page 504
 - "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 503

:SBUS<n>:SPI:SOURce:FRAMe**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:SPI:SOURce:FRAMe <source>` `<source> ::= {CHANnel<n> | EXTERNAL}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :SBUS<n>:SPI:SOURce:FRAMe command sets the frame source when :SBUS<n>:SPI:FRAMing is set to CHIPselect or NOTChipselect.

Query Syntax `:SBUS<n>:SPI:SOURce:FRAMe?`

The :SBUS<n>:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Return Format `<source><NL>`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:SBUS<n>:SPI:SOURce:CLOCK](#)" on page 501
 - "[:SBUS<n>:SPI:SOURce:MOSI](#)" on page 504
 - "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 503
 - "[:SBUS<n>:SPI:FRAMing](#)" on page 500

:SBUS<n>:SPI:SOURce:MISO

N (see [page 794](#))

Command Syntax :SBUS<n>:SPI:SOURce:MISO <source>

<source> ::= {CHANnel<n> | EXTERNAL}

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

The :SBUS<n>:SPI:SOURce:MISO command sets the source for the SPI serial MISO data.

Query Syntax :SBUS<n>:SPI:SOURce:MISO?

The :SBUS<n>:SPI:SOURce:MISO? query returns the current source for the SPI serial MISO data.

Return Format <source><NL>

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:SBUS<n>:SPI:SOURce:MOSI](#)" on page 504
- "[:SBUS<n>:SPI:SOURce:CLOCK](#)" on page 501
- "[:SBUS<n>:SPI:SOURce:FRAMe](#)" on page 502
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA](#)" on page 505
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA](#)" on page 507
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh](#)" on page 506
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh](#)" on page 508

:SBUS<n>:SPI:SOURce:MOSI

N (see [page 794](#))

Command Syntax `:SBUS<n>:SPI:SOURce:MOSI <source>`

`<source> ::= {CHANnel<n> | EXTernal}`

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

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.

Query Syntax `:SBUS<n>:SPI:SOURce:MOSI?`

The :SBUS<n>:SPI:SOURce:MOSI? query returns the current source for the SPI serial MOSI data.

Return Format `<source><NL>`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[":SBUS<n>:SPI:SOURce:MISO](#)" on page 503
- "[":SBUS<n>:SPI:SOURce:CLOCK](#)" on page 501
- "[":SBUS<n>:SPI:SOURce:FRAMe](#)" on page 502
- "[":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA](#)" on page 505
- "[":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA](#)" on page 507
- "[":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh](#)" on page 506
- "[":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh](#)" on page 508

:SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA

N (see [page 794](#))

Command Syntax :SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA <string>

```
<string> ::= "nn...n" where n ::= {0 | 1 | x | $}
<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | x | $}
```

The :SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA command defines the SPI data pattern resource according to the string parameter. This pattern, along with the data width, control the data pattern searched for in the data stream.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

The :SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA.

Query Syntax :SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA?

The :SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Return Format <string><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh](#)" on page 506
 - "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 503

:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh <width>` `<width> ::= integer from 4 to 64 in NR1 format`

The :SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 64 bits.

NOTE

The :SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA.

Query Syntax `:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh?`

The :SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh? query returns the current SPI data pattern width setting.

Return Format `<width><NL>` `<width> ::= integer from 4 to 64 in NR1 format`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591

- "[":SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA](#)" on page 505

- "[":SBUS<n>:SPI:SOURce:MISO](#)" on page 503

:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA

N (see [page 794](#))

Command Syntax :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA <string>

```
<string> ::= "nn...n" where n ::= {0 | 1 | x | $}
<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | x | $}
```

The :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA command defines the SPI data pattern resource according to the string parameter. This pattern, along with the data width, control the data pattern searched for in the data stream.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

The :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA.

Query Syntax :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA?

The :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Return Format <string><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 591
 - [":SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh"](#) on page 508
 - [":SBUS<n>:SPI:SOURce:MOSI"](#) on page 504

:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh

N (see [page 794](#))

Command Syntax `:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh <width>`
`<width> ::= integer from 4 to 64 in NR1 format`

The :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 64 bits.

NOTE

The :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA.

Query Syntax `:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh?`

The :SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh? query returns the current SPI data pattern width setting.

Return Format `<width><NL>`
`<width> ::= integer from 4 to 64 in NR1 format`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[":SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA](#)" on page 507
- "[":SBUS<n>:SPI:SOURce:MOSI](#)" on page 504

:SBUS<n>:SPI:TRIGger:TYPE

N (see [page 794](#))

Command Syntax `:SBUS<n>:SPI:TRIGger:TYPE <value>`
`<value> ::= {MOSI | MISO}`

The :SBUS<n>:SPI:TRIGger:TYPE command specifies whether the SPI trigger will be on the MOSI data or the MISO data.

When triggering on MOSI data, the data value is specified by the :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA and :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh commands.

When triggering on MISO data, the data value is specified by the :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA and :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh commands.

Query Syntax `:SBUS<n>:SPI:TRIGger:TYPE?`

The :SBUS<n>:SPI:TRIGger:TYPE? query returns the current SPI trigger type setting.

Return Format `<value><NL>`
`<value> ::= {MOSI | MISO}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[":SBUS<n>:SPI:SOURce:MOSI"](#) on page 504
- "[":SBUS<n>:SPI:SOURce:MISO"](#) on page 503
- "[":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA"](#) on page 505
- "[":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA"](#) on page 507
- "[":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTH"](#) on page 506
- "[":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTH"](#) on page 508
- "[":TRIGger:MODE"](#) on page 600

:SBUS<n>:SPI:WIDTH**N** (see [page 794](#))**Command Syntax** `: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.

Query Syntax `:SBUS<n>:SPI:WIDTH?`

The :SBUS<n>:SPI:WIDTH? query returns the current SPI decode word width.

Return Format `<word_width><NL>` `<word_width> ::= integer 4-16 in NR1 format`**Errors** • ["-241, Hardware missing" on page 755](#)**See Also** • ["Introduction to :SBUS<n> Commands" on page 449](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:SPI Commands" on page 495](#)

:SBUS<n>:UART Commands

NOTE

These commands are only valid when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Table 80 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 513)	:SBUS<n>:UART:BASE? (see page 513)	<base> ::= {ASCII BINARY HEX}
:SBUS<n>:UART:BAUDrate e <baudrate> (see page 514)	:SBUS<n>:UART:BAUDrate e? (see page 514)	<baudrate> ::= integer from 100 to 8000000
:SBUS<n>:UART:BITorde r <bitorder> (see page 515)	:SBUS<n>:UART:BITorde r? (see page 515)	<bitorder> ::= {LSBFIRST MSBFIRST}
n/a	:SBUS<n>:UART:COUNT:E RRor? (see page 516)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNT:R ESet (see page 517)	n/a	n/a
n/a	:SBUS<n>:UART:COUNT:R XFRAMES? (see page 518)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNT:T XFRAMES? (see page 519)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 520)	:SBUS<n>:UART:FRAMing ? (see page 520)	<value> ::= {OFF <decimal> <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xFF) <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary
:SBUS<n>:UART:PARity <parity> (see page 521)	:SBUS<n>:UART:PARity? (see page 521)	<parity> ::= {EVEN ODD NONE}
:SBUS<n>:UART:POLarit y <polarity> (see page 522)	:SBUS<n>:UART:POLarit y? (see page 522)	<polarity> ::= {HIGH LOW}

Table 80 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce: RX <source> (see page 523)	:SBUS<n>:UART:SOURce: RX? (see page 523)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:UART:SOURce: TX <source> (see page 524)	:SBUS<n>:UART:SOURce: TX? (see page 524)	<source> ::= {CHANnel<n> EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:UART:TRIGger :BASE <base> (see page 525)	:SBUS<n>:UART:TRIGger :BASE? (see page 525)	<base> ::= {ASCII HEX}
:SBUS<n>:UART:TRIGger :BURSt <value> (see page 526)	:SBUS<n>:UART:TRIGger :BURSt? (see page 526)	<value> ::= {OFF 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger :DATA <value> (see page 527)	:SBUS<n>:UART:TRIGger :DATA? (see page 527)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SBUS<n>:UART:TRIGger :IDLE <time_value> (see page 528)	:SBUS<n>:UART:TRIGger :IDLE? (see page 528)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger :QUALifier <value> (see page 529)	:SBUS<n>:UART:TRIGger :QUALifier? (see page 529)	<value> ::= {EQUAL NOTEqual GREaterthan LESSthan}
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 530)	:SBUS<n>:UART:TRIGger :TYPE? (see page 530)	<value> ::= {RSTARt RSTOP RDATa RD1 RD0 RDX PARityerror TSTARt TSTOP TDATa TD1 TD0 TDX}
:SBUS<n>:UART:WIDTh <width> (see page 531)	:SBUS<n>:UART:WIDTh? (see page 531)	<width> ::= {5 6 7 8 9}

:SBUS<n>:UART:BASE**N** (see [page 794](#))

Command Syntax `:SBUS<n>:UART:BASE <base>`
 `<base> ::= {ASCII | BINary | HEX}`

The :SBUS<n>:UART:BASE command determines the base to use for the UART decode and Lister display.

Query Syntax `:SBUS<n>:UART:BASE?`

The :SBUS<n>:UART:BASE? query returns the current UART decode and Lister base setting.

Return Format `<base><NL>`
 `<base> ::= {ASCII | BINary | HEX}`

Errors • ["-241, Hardware missing" on page 755](#)

See Also • ["Introduction to :SBUS<n> Commands" on page 449](#)
 • [":SBUS<n>:UART Commands" on page 511](#)

:SBUS<n>:UART:BAUDrate**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:UART:BAUDrate <baudrate>``<baudrate> ::= integer from 100 to 8000000`

The :SBUS<n>:UART:BAUDrate command selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode. The baud rate can be set from 100 b/s to 8 Mb/s.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax `:SBUS<n>:UART:BAUDrate?`

The :SBUS<n>:UART:BAUDrate? query returns the current UART baud rate setting.

Return Format `<baudrate><NL>``<baudrate> ::= integer from 100 to 8000000`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530

:SBUS<n>:UART:BITorder

N (see [page 794](#))

Command Syntax `:SBUS<n>:UART:BITorder <bitorder>`
`<bitorder> ::= {LSBFFirst | MSBFFirst}`

The :SBUS<n>:UART:BITorder command specifies the order of transmission used by the physical Tx and Rx input signals for the serial decoder and/or trigger when in UART mode. LSBFirst sets the least significant bit of each message "byte" as transmitted first. MSBFirst sets the most significant bit as transmitted first.

Query Syntax `:SBUS<n>:UART:BITorder?`

The :SBUS<n>:UART:BITorder? query returns the current UART bit order setting.

Return Format `<bitorder><NL>`
`<bitorder> ::= {LSBF | MSBF}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530
- "[:SBUS<n>:UART:SOURce:RX](#)" on page 523
- "[:SBUS<n>:UART:SOURce:TX](#)" on page 524

:SBUS<n>:UART:COUNt:ERROr

N (see [page 794](#))

Query Syntax :SBUS<n>:UART:COUNt:ERROr?

Returns the UART error frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing" on page 755](#)

See Also • [":SBUS<n>:UART:COUNt:RESet" on page 517](#)
• ["Introduction to :SBUS<n> Commands" on page 449](#)
• [":SBUS<n>:MODE" on page 453](#)
• [":SBUS<n>:UART Commands" on page 511](#)

:SBUS< n >:UART:COUNT:RESet**N** (see [page 794](#))**Command Syntax** :SBUS< n >:UART:COUNT:RESet

Resets the UART frame counters.

Errors • ["-241, Hardware missing" on page 755](#)**See Also** • [":SBUS< n >:UART:COUNT:ERRor" on page 516](#)
• [":SBUS< n >:UART:COUNT:RXFRAMES" on page 518](#)
• [":SBUS< n >:UART:COUNT:TXFRAMES" on page 519](#)
• ["Introduction to :SBUS< n > Commands" on page 449](#)
• [":SBUS< n >:MODE" on page 453](#)
• [":SBUS< n >:UART Commands" on page 511](#)

:SBUS<n>:UART:COUNT:RXFRAMES

N (see [page 794](#))

Query Syntax :SBUS<n>:UART:COUNT:RXFRAMES?

Returns the UART Rx frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing"](#) on page 755

See Also • [":SBUS<n>:UART:COUNT:RESET"](#) on page 517
• ["Introduction to :SBUS<n> Commands"](#) on page 449
• [":SBUS<n>:MODE"](#) on page 453
• [":SBUS<n>:UART Commands"](#) on page 511

:SBUS<n>:UART:COUNt:TXFRAMES**N** (see [page 794](#))**Query Syntax** :SBUS<n>:UART:COUNt:TXFRAMES?

Returns the UART Tx frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing"](#) on page 755**See Also** • [":SBUS<n>:UART:COUNt:RESet"](#) on page 517
• ["Introduction to :SBUS<n> Commands"](#) on page 449
• [":SBUS<n>:MODE"](#) on page 453
• [":SBUS<n>:UART Commands"](#) on page 511

:SBUS<n>:UART:FRAMing

N (see [page 794](#))

Command Syntax	<code>:SBUS<n>:UART:FRAMing <value></code>
	<code><value> ::= {OFF <decimal> <nondecimal>}</code>
	<code><decimal> ::= 8-bit integer in decimal from 0-255 (0x00-0xff)</code>
	<code><nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal</code>
	<code><nondecimal> ::= #Bnn...n where n ::= {0 1} for binary</code>
	The :SBUS<n>:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.
Query Syntax	<code>:SBUS<n>:UART:FRAMing?</code>
	The :SBUS<n>:UART:FRAMing? query returns the current UART decode base setting.
Return Format	<code><value><NL></code>
	<code><value> ::= {OFF <decimal>}</code>
	<code><decimal> ::= 8-bit integer in decimal from 0-255</code>
Errors	<ul style="list-style-type: none"> • "-241, Hardware missing" on page 755
See Also	<ul style="list-style-type: none"> • "Introduction to :SBUS<n> Commands" on page 449 • ":SBUS<n>:UART Commands" on page 511

:SBUS<n>:UART:PARity**N** (see [page 794](#))

Command Syntax `:SBUS<n>:UART:PARity <parity>`
`<parity> ::= {EVEN | ODD | NONE}`

The :SBUS<n>:UART:PARity command selects the parity to be used with each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax `:SBUS<n>:UART:PARity?`

The :SBUS<n>:UART:PARity? query returns the current UART parity setting.

Return Format `<parity><NL>`
`<parity> ::= {EVEN | ODD | NONE}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530

:SBUS<n>:UART:POLarity**N** (see [page 794](#))

Command Syntax `:SBUS<n>:UART:POLarity <polarity>`
`<polarity> ::= {HIGH | LOW}`

The :SBUS<n>:UART:POLarity command selects the polarity as idle low or idle high for the serial decoder and/or trigger when in UART mode.

Query Syntax `:SBUS<n>:UART:POLarity?`

The :SBUS<n>:UART:POLarity? query returns the current UART polarity setting.

Return Format `<polarity><NL>`
`<polarity> ::= {HIGH | LOW}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530

:SBUS<n>:UART:SOURce:RX

N (see [page 794](#))

Command Syntax :SBUS<n>:UART:SOURce:RX <source>

<source> ::= {CHANnel<n> | EXTERNAL}

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

The :SBUS<n>:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

Query Syntax :SBUS<n>:UART:SOURce:RX?

The :SBUS<n>:UART:SOURce:RX? query returns the current source for the UART Rx signal.

Return Format <source><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:MODE](#)" on page 600
 - "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530
 - "[:SBUS<n>:UART:BITorder](#)" on page 515

:SBUS<n>:UART:SOURce:TX**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:UART:SOURce:TX <source>` `<source> ::= {CHANnel<n> | EXTernal}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :SBUS<n>:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

Query Syntax `:SBUS<n>:UART:SOURce:TX?`

The :SBUS<n>:UART:SOURce:TX? query returns the current source for the UART Tx signal.

Return Format `<source><NL>`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:MODE](#)" on page 600
 - "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530
 - "[:SBUS<n>:UART:BITorder](#)" on page 515

:SBUS<n>:UART:TRIGger:BASE

N (see [page 794](#))

Command Syntax `:SBUS<n>:UART:TRIGger:BASE <base>`
`<base> ::= {ASCII | HEX}`

The :SBUS<n>:UART:TRIGger:BASE command sets the front panel UART/RS232 trigger setup data selection option:

- ASCII – front panel data selection is from ASCII values.
- HEX – front panel data selection is from hexadecimal values.

The :SBUS<n>:UART:TRIGger:BASE setting does not affect the :SBUS<n>:UART:TRIGger:DATA command which can always set data values using ASCII or hexadecimal values.

NOTE

The :SBUS<n>:UART:TRIGger:BASE command is independent of the :SBUS<n>:UART:BASE command which affects decode and Lister only.

Query Syntax `:SBUS<n>:UART:TRIGger:BASE?`

The :SBUS<n>:UART:TRIGger:BASE? query returns the current UART base setting.

Return Format `<base><NL>`

`<base> ::= {ASC | HEX}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:DATA](#)" on page 527

:SBUS<n>:UART:TRIGger:BURSt**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:UART:TRIGger:BURSt <value>``<value> ::= {OFF | 1 to 4096 in NR1 format}`

The :SBUS<n>:UART:TRIGger:BURSt command selects the burst value (Nth frame after idle period) in the range 1 to 4096 or OFF, for the trigger when in UART mode.

Query Syntax `:SBUS<n>:UART:TRIGger:BURSt?`

The :SBUS<n>:UART:TRIGger:BURSt? query returns the current UART trigger burst value.

Return Format `<value><NL>``<value> ::= {OFF | 1 to 4096 in NR1 format}`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:IDLE](#)" on page 528
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530

:SBUS<n>:UART:TRIGger:DATA

N (see [page 794](#))

Command Syntax :SBUS<n>:UART:TRIGger:DATA <value>

```
<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal,
           <hexadecimal>, <binary>, or <quoted_string> format

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

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

<quoted_string> ::= any of the 128 valid 7-bit ASCII characters
                   (or standard abbreviations)
```

The :SBUS<n>:UART:TRIGger:DATA command selects the data byte value (0x00 to 0xFF) for the trigger QUALifier when in UART mode. The data value is used when one of the RD or TD trigger types is selected.

When entering an ASCII character via the quoted string, it must be one of the 128 valid characters (case-sensitive): "NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK", "BEL", "BS", "HT", "LF", "VT", "FF", "CR", "SO", "SI", "DLE", "DC1", "DC2", "DC3", "DC4", "NAK", "SYN", "ETB", "CAN", "EM", "SUB", "ESC", "FS", "GS", "RS", "US", "SP", "!", "\\", "#", "\$", "%", "&", "\\", "(", ")", "*", "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "[", "\\", "]", "^", "_", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~", or "DEL".

Query Syntax :SBUS<n>:UART:TRIGger:DATA?

The :SBUS<n>:UART:TRIGger:DATA? query returns the current UART trigger data value.

Return Format <value><NL>

```
<value> ::= 8-bit integer in decimal from 0-255
```

See Also

- ["Introduction to :TRIGger Commands"](#) on page 591
- [":TRIGger:MODE"](#) on page 600
- [":SBUS<n>:UART:TRIGger:BASE"](#) on page 525
- [":SBUS<n>:UART:TRIGger:TYPE"](#) on page 530

:SBUS<n>:UART:TRIGger:IDLE**N** (see [page 794](#))**Command Syntax** `:SBUS<n>:UART:TRIGger:IDLE <time_value>``<time_value> ::= time from 1 us to 10 s in NR3 format`

The :SBUS<n>:UART:TRIGger:IDLE command selects the value of the idle period for burst trigger in the range from 1 us to 10 s when in UART mode.

Query Syntax `:SBUS<n>:UART:TRIGger:IDLE?`

The :SBUS<n>:UART:TRIGger:IDLE? query returns the current UART trigger idle period time.

Return Format `<time_value><NL>``<time_value> ::= time from 1 us to 10 s in NR3 format`**See Also**

- "Introduction to :TRIGger Commands" on page 591
- ":TRIGger:MODE" on page 600
- ":SBUS<n>:UART:TRIGger:BURSt" on page 526
- ":SBUS<n>:UART:TRIGger:TYPE" on page 530

:SBUS<n>:UART:TRIGger:QUALifier

N (see [page 794](#))

Command Syntax	<code>:SBUS<n>:UART:TRIGger:QUALifier <value></code> <code><value> ::= {EQUAL NOTEqual GREaterthan LESSthan}</code>
	The :SBUS<n>:UART:TRIGger:QUALifier command selects the data qualifier when :TYPE is set to RDATa, RD1, RD0, RDX, TDATa, TD1, TD0, or TDX for the trigger when in UART mode.
Query Syntax	<code>:SBUS<n>:UART:TRIGger:QUALifier?</code>
	The :SBUS<n>:UART:TRIGger:QUALifier? query returns the current UART trigger qualifier.
Return Format	<code><value><NL></code> <code><value> ::= {EQU NOT GRE LESS}</code>
See Also	<ul style="list-style-type: none"> · "Introduction to :TRIGger Commands" on page 591 · ":TRIGger:MODE" on page 600 · ":SBUS<n>:UART:TRIGger:TYPE" on page 530

:SBUS<n>:UART:TRIGger:TYPE

N (see [page 794](#))

Command Syntax `:SBUS<n>:UART:TRIGger:TYPE <value>`

```
<value> ::= {RSTA | RSTO | RDAT | RD1 | RD0 | RDX | PARityerror
             | TSTA | TSTO | TDAT | TD1 | TD0 | TDX}
```

The :SBUS<n>:UART:TRIGger:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :SBUS<n>:UART:TRIGger:DATA and :SBUS<n>:UART:TRIGger:QUALifier commands are used to specify the data value and comparison operator.

The RD1, RD0, RDX, TD1, TD0, and TDX types (for triggering on data and alert bit values) are only valid when a 9-bit width has been selected.

Query Syntax `:SBUS<n>:UART:TRIGger:TYPE?`

The :SBUS<n>:UART:TRIGger:TYPE? query returns the current UART trigger data value.

Return Format `<value><NL>`

```
<value> ::= {RSTA | RSTO | RDAT | RD1 | RD0 | RDX | PAR | TSTA |
             TSTO | TDAT | TD1 | TD0 | TDX}
```

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:DATA](#)" on page 527
- "[:SBUS<n>:UART:TRIGger:QUALifier](#)" on page 529
- "[:SBUS<n>:UART:WIDTh](#)" on page 531

:SBUS<n>:UART:WIDTh**N** (see [page 794](#))

Command Syntax `:SBUS<n>:UART:WIDTh <width>`
`<width> ::= {5 | 6 | 7 | 8 | 9}`

The :SBUS<n>:UART:WIDTh command determines the number of bits (5–9) for each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax `:SBUS<n>:UART:WIDTh?`

The :SBUS<n>:UART:WIDTh? query returns the current UART width setting.

Return Format `<width><NL>`
`<width> ::= {5 | 6 | 7 | 8 | 9}`

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 530

26 :SEARch Commands

Control the event search modes and parameters for each search type. See:

- "[General :SEARch Commands](#)" on page 534
- "[:SEARch:SERial:CAN Commands](#)" on page 538
- "[:SEARch:SERial:IIC Commands](#)" on page 544
- "[:SEARch:SERial:LIN Commands](#)" on page 551
- "[:SEARch:SERial:SPI Commands](#)" on page 557
- "[:SEARch:SERial:UART Commands](#)" on page 561

General :SEARch Commands

Table 81 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNT? (see page 535)	<count> ::= an integer count value
:SEARch:MODE <value> (see page 536)	:SEARch:MODE? (see page 536)	<value> ::= {SERial1}
:SEARch:STATE <value> (see page 537)	:SEARch:STATE? (see page 537)	<value> ::= {{0 OFF} {1 ON}}

:SEARch:COUNT**N** (see [page 794](#))**Query Syntax** `:SEARch:COUNT?`

The :SEARch:COUNT? query returns the number of search events found.

Return Format `<count><NL>`

`<count>` ::= an integer count value

See Also • [Chapter 26](#), “:SEARch Commands,” starting on page 533

:SEARch:MODE**N** (see [page 794](#))

Command Syntax `:SEARch:MODE <value>`
 `<value> ::= {SERial1}`

The :SEARch:MODE command selects the search mode.

The command is only valid when the :SEARch:STATe is ON.

Query Syntax `:SEARch:MODE?`

The :SEARch:MODE? query returns the currently selected mode or OFF if the :SEARch:STATe is OFF.

Return Format `<value><NL>`
 `<value> ::= {SER1 | OFF}`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:STATe"](#) on page 537

:SEARch:STATe

N (see [page 794](#))

Command Syntax `:SEARch:STATe <value>`

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

The :SEARch:STATe command enables or disables the search feature.

Query Syntax `:SEARch:STATE?`

The :SEARch:STATE? query returns the current setting.

Return Format `<value><NL>`

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

See Also • [Chapter 26](#), “:SEARch Commands,” starting on page 533

• [":SEARch:MODE"](#) on page 536

:SEARch:SERial:CAN Commands

Table 82 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 539)	:SEARch:SERial:CAN:MODE? (see page 539)	<value> ::= {DATA IDData IDEither IDRmote ALLerrors OVERload ERRor}
:SEARch:SERial:CAN:ATTern:DATA <string> (see page 540)	:SEARch:SERial:CAN:ATTern:DATA? (see page 540)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARch:SERial:CAN:ATTern:DATA:LENGTH <length> (see page 541)	:SEARch:SERial:CAN:ATTern:DATA:LENGTH? (see page 541)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:ATTern:ID <string> (see page 542)	:SEARch:SERial:CAN:ATTern:ID? (see page 542)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARch:SERial:CAN:ATTern:ID:MODE <value> (see page 543)	:SEARch:SERial:CAN:ATTern:ID:MODE? (see page 543)	<value> ::= {STANDARD EXTENDED}

:SEARch:SERial:CAN:MODE

N (see [page 794](#))

Command Syntax `:SEARch:SERial:CAN:MODE <value>`

```
<value> ::= {DATA | IDDData | IDEither | IDRremote | ALLerrors
              | OVERload | ERRor}
```

The :SEARch:SERial:CAN:MODE command selects the type of CAN information to find in the Lister display:

- DATA - searches for CAN Data frames matching the specified ID, Data, and the DLC (Data length code).
- IDDData - searches for CAN frames matching the specified ID of a Data frame.
- IDEither - searches for the specified ID, regardless if it is a Remote frame or a Data frame.
- IDRremote - searches for CAN frames matching the specified ID of a Remote frame.
- ALLerrors - searches for CAN active error frames and unknown bus conditions.
- OVERload - searches for CAN overload frames.
- ERRor - searches for CAN Error frame.

Query Syntax `:SEARch:SERial:CAN:MODE?`

The :SEARch:SERial:CAN:MODE? query returns the currently selected mode.

Return Format `<value><NL>`

```
<value> ::= {DATA | IDD | IDE | IDR | ALL | OVER | ERR}
```

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:CAN:PATTERn:DATA"](#) on page 540
- [":SEARch:SERial:CAN:PATTERn:ID"](#) on page 542

:SEARCh:SERial:CAN:PATTERn:DATA**N** (see [page 794](#))**Command Syntax** `:SEARCh:SERial:CAN:PATTERn:DATA <string>``<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X}
for hexadecimal`

The :SEARCh:SERial:CAN:PATTERn:DATA command specifies the data value when searching for Data Frame ID and Data.

The length of the data value is specified using the :SEARCh:SERial:CAN:PATTERn:DATA:LENGth command.

Query Syntax `:SEARCh:SERial:CAN:PATTERn:DATA?`

The :SEARCh:SERial:CAN:PATTERn:DATA? query returns the current data value setting.

Return Format `<string><NL>``<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X}
for hexadecimal`**See Also**

- [Chapter 26](#), “:SEARCh Commands,” starting on page 533
- [":SEARCh:SERial:CAN:MODE"](#) on page 539
- [":SEARCh:SERial:CAN:PATTERn:DATA:LENGth"](#) on page 541

:SEARch:SERial:CAN:PATTERn:DATA:LENGth

N (see [page 794](#))

Command Syntax `:SEARch:SERial:CAN:PATTERn:DATA:LENGth <length>`
`<length> ::= integer from 1 to 8 in NR1 format`

The :SEARch:SERial:CAN:PATTERn:DATA:LENGth command specifies the length of the data value when searching for Data Frame ID and Data.

The data value is specified using the :SEARch:SERial:CAN:PATTERn:DATA command.

Query Syntax `:SEARch:SERial:CAN:PATTERn:DATA:LENGth?`

The :SEARch:SERial:CAN:PATTERn:DATA:LENGth? query returns the current data length setting.

Return Format `<length><NL>`
`<length> ::= integer from 1 to 8 in NR1 format`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:CAN:MODE"](#) on page 539
- [":SEARch:SERial:CAN:PATTERn:DATA"](#) on page 540

:SEARch:SERial:CAN:PATTERn:ID**N** (see [page 794](#))**Command Syntax** `:SEARch:SERial:CAN:PATTERn:ID <string>``<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X}
for hexadecimal`

The :SEARch:SERial:CAN:PATTERn:ID command specifies the ID value when searching for a CAN event.

The value can be a standard ID or an extended ID, depending on the :SEARch:SERial:CAN:PATTERn:ID:MODE command's setting.

Query Syntax `:SEARch:SERial:CAN:PATTERn:ID?`

The :SEARch:SERial:CAN:PATTERn:ID? query returns the current ID value setting.

Return Format `<string><NL>``<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X}
for hexadecimal`**See Also**

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:CAN:MODE"](#) on page 539
- [":SEARch:SERial:CAN:PATTERn:ID:MODE"](#) on page 543

:SEARch:SERial:CAN:PATTERn:ID:MODE

N (see [page 794](#))

Command Syntax `:SEARch:SERial:CAN:PATTERn:ID:MODE <value>`
`<value> ::= {STANDARD | EXTended}`

The :SEARch:SERial:CAN:PATTERn:ID:MODE command specifies whether a standard ID value or an extended ID value is used when searching for a CAN event.

The ID value is specified using the :SEARch:SERial:CAN:PATTERn:ID command.

Query Syntax `:SEARch:SERial:CAN:PATTERn:ID:MODE?`

The :SEARch:SERial:CAN:PATTERn:ID:MODE? query returns the current setting.

Return Format `<value><NL>`
`<value> ::= {STAN | EXT}`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:CAN:MODE"](#) on page 539
- [":SEARch:SERial:CAN:PATTERn:ID"](#) on page 542

:SEARch:SERial:IIC Commands

Table 83 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MO DE <value> (see page 545)	:SEARch:SERial:IIC:MO DE? (see page 545)	<value> ::= { READ7 WRITE7 NACKnowledge ANACK R7Data2 W7Data2 RESTart READEprom}
:SEARch:SERial:IIC:PA TTern:ADDReSS <value> (see page 547)	:SEARch:SERial:IIC:PA TTern:ADDReSS? (see page 547)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:PA TTern:DATA <value> (see page 548)	:SEARch:SERial:IIC:PA TTern:DATA? (see page 548)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:PA TTern:DATA2 <value> (see page 549)	:SEARch:SERial:IIC:PA TTern:DATA2? (see page 549)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:QU ALifier <value> (see page 550)	:SEARch:SERial:IIC:QU ALifier? (see page 550)	<value> ::= { EQUAL NOTequal LESSthan GREaterthan}

:SEARch:SERial:IIC:MODE

N (see [page 794](#))

Command Syntax :SEARch:SERial:IIC:MODE <value>

```
<value> ::= {READ7 | WRITE7 | NACKnowledge | ANACK | R7Data2  
| W7Data2 | RESTart | READEprom}
```

The :SEARch:SERial:IIC:MODE command selects the type of IIC information to find in the Lister display:

- READ7 – searches for 7-bit address frames containing Start:Address7:Read:Ack:Data. The value READ is also accepted for READ7.
- WRITE7 – searches for 7-bit address frames containing Start:Address7:Write:Ack:Data. The value WRITE is also accepted for WRITE7.
- NACKnowledge – searches for missing acknowledge events.
- ANACK – searches for address with no acknowledge events.
- R7Data2 – searches for 7-bit address frames containing Start:Address7:Read:Ack:Data:Ack:Data2.
- W7Data2 – searches for 7-bit address frames containing Start:Address7:Write:Ack:Data:Ack:Data2.
- RESTart – searches for another start condition occurring before a stop condition.
- READEprom – searches for EEPROM data reads.

NOTE

The short form of READ7 (READ7), READEprom (READE), and WRITE7 (WRIT7) do not follow the defined Long Form to Short Form Truncation Rules (see [page 796](#)).

When searching for events containing addresses, address values are specified using the :SEARch:SERial:IIC:PATTERn:ADDReSS command.

When searching for events containing data, data values are specified using the :SEARch:SERial:IIC:PATTERn:DATA and :SEARch:SERial:IIC:PATTERn:DATA2 commands.

Query Syntax :SEARch:SERial:IIC:MODE?

The :SEARch:SERial:IIC:MODE? query returns the currently selected mode.

Return Format <value><NL>

```
<value> ::= {READ7 | WRITE7 | NACK | ANAC | R7D2 | W7D2 | REST  
| READE}
```

- See Also**
- [Chapter 26](#), “:SEARch Commands,” starting on page 533
 - [":SEARch:SERial:IIC:PATTERn:DATA"](#) on page 547
 - [":SEARch:SERial:IIC:PATTERn:DATA2"](#) on page 548

- [":SEARch:SERial:IIC:PATTERn:DATA2" on page 549](#)
- [":SEARch:SERial:IIC:QUALifier" on page 550](#)

:SEARch:SERial:IIC:PATTERn:ADDRess

N (see [page 794](#))

Command Syntax

```
:SEARch:SERial:IIC:PATTERn:ADDRess <value>
<value> ::= integer or <string>
<string> ::= "0xnn" n ::= {0,...,9 | A,...,F}
```

The :SEARch:SERial:IIC:PATTERn:ADDRess command specifies address values when searching for IIC events.

To set don't care values, use the integer -1.

Query Syntax

```
:SEARch:SERial:IIC:PATTERn:ADDRess?
```

The :SEARch:SERial:IIC:PATTERn:ADDRess? query returns the current address value setting.

Return Format

```
<value><NL>
<value> ::= integer
```

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:IIC:MODE"](#) on page 545

:SEARch:SERial:IIC:PATTERn:DATA**N** (see [page 794](#))

Command Syntax `:SEARch:SERial:IIC:PATTERn:DATA <value>`
`<value> ::= integer or <string>`
`<string> ::= "0xnn" n ::= {0,...,9 | A,...,F}`

The :SEARch:SERial:IIC:PATTERn:DATA command specifies data values when searching for IIC events.

To set don't care values, use the integer -1.

When searching for IIC EEPROM data read events, you specify the data value qualifier using the :SEARch:SERial:IIC:QUALifier command.

Query Syntax `:SEARch:SERial:IIC:PATTERn:DATA?`

The :SEARch:SERial:IIC:PATTERn:DATA? query returns the current data value setting.

Return Format `<value><NL>`

`<value> ::= integer`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:IIC:MODE"](#) on page 545
- [":SEARch:SERial:IIC:QUALifier"](#) on page 550
- [":SEARch:SERial:IIC:PATTERn:DATA2"](#) on page 549

:SEARch:SERial:IIC:PATTERn:DATA2**N** (see [page 794](#))

Command Syntax `:SEARch:SERial:IIC:PATTERn:DATA2 <value>`
`<value> ::= integer or <string>`
`<string> ::= "0xnn" n ::= {0,...,9 | A,...,F}`

The :SEARch:SERial:IIC:PATTERn:DATA2 command specifies the second data value when searching for IIC events with two data values.

To set don't care values, use the integer -1.

Query Syntax `:SEARch:SERial:IIC:PATTERn:DATA2?`

The :SEARch:SERial:IIC:PATTERn:DATA2? query returns the current second data value setting.

Return Format `<value><NL>`
`<value> ::= integer`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:IIC:MODE"](#) on page 545
- [":SEARch:SERial:IIC:PATTERn:DATA"](#) on page 548

:SEARch:SERial:IIC:QUALifier**N** (see [page 794](#))**Command Syntax** `:SEARch:SERial:IIC:QUALifier <value>``<value> ::= {EQUAL | NOTEqual | LESSthan | GREaterthan}`

The :SEARch:SERial:IIC:QUALifier command specifies the data value qualifier used when searching for IIC EEPROM data read events.

Query Syntax `:SEARch:SERial:IIC:QUALifier?`

The :SEARch:SERial:IIC:QUALifier? query returns the current data value qualifier setting.

Return Format `<value><NL>``<value> ::= {EQU | NOT | LESS | GRE}`**See Also**

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:IIC:MODE"](#) on page 545
- [":SEARch:SERial:IIC:PATTERn:DATA"](#) on page 548

:SEARch:SERial:LIN Commands

Table 84 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 552)	:SEARch:SERial:LIN:ID ? (see page 552)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal
:SEARch:SERial:LIN:MO DE <value> (see page 553)	:SEARch:SERial:LIN:MO DE? (see page 553)	<value> ::= {ID DATA ERRor}
:SEARch:SERial:LIN:PA TTern:DATA <string> (see page 554)	:SEARch:SERial:LIN:PA TTern:DATA? (see page 554)	When :SEARch:SERial:LIN:PATTern:FORMAT DECimal, <string> ::= "n" where n ::= 32-bit integer in unsigned decimal, returns "\$" if data has any don't cares When :SEARch:SERial:LIN:PATTern:FORMAT HEX, <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X }
:SEARch:SERial:LIN:PA TTern:DATA:LENGTH <length> (see page 555)	:SEARch:SERial:LIN:PA TTern:DATA:LENGTH? (see page 555)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PA TTern:FORMAT <base> (see page 556)	:SEARch:SERial:LIN:PA TTern:FORMAT? (see page 556)	<base> ::= {HEX DECimal}

:SEARch:SERial:LIN:ID**N** (see [page 794](#))

- Command Syntax** `:SEARch:SERial:LIN:ID <value>`
- `<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>`
`from 0-63 or 0x00-0x3f (with Option AMS)`
- `<nondecimal> ::= #Hnn where n ::= {0,...,9 | A,...,F} for hexadecimal`
- `<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary`
- `<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F} for hexadecimal`
- The :SEARch:SERial:LIN:ID command specifies the frame ID value when searching for LIN events.
- Query Syntax** `:SEARch:SERial:LIN:ID?`
- The :SEARch:SERial:LIN:ID? query returns the current frame ID setting.
- Return Format** `<value><NL>`
- `<value> ::= 7-bit integer in decimal (with Option AMS)`
- See Also**
- [Chapter 26, “:SEARch Commands,” starting on page 533](#)
 - [“:SEARch:SERial:LIN:MODE” on page 553](#)

:SEARch:SERial:LIN:MODE

N (see [page 794](#))

Command Syntax `:SEARch:SERial:LIN:MODE <value>`
`<value> ::= {ID | DATA | ERRor}`

The :SEARch:SERial:LIN:MODE command selects the type of LIN information to find in the Lister display:

- ID – searches for a frame ID.
- DATA – searches for a frame ID and data.
- ERRor – searches for errors.

Frame IDs are specified using the :SEARch:SERial:LIN:ID command.

Data values are specified using the :SEARch:SERial:LIN:PATTERn:DATA command.

Query Syntax `:SEARch:SERial:LIN:MODE?`

The :SEARch:SERial:LIN:MODE? query returns the currently selected mode.

Return Format `<value><NL>`
`<value> ::= {ID | DATA | ERR}`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- “[:SEARch:SERial:LIN:ID](#)” on page 552
- “[:SEARch:SERial:LIN:PATTERn:DATA](#)” on page 554

:SEARCh:SERial:LIN:PATTern:DATA

N (see [page 794](#))

Command Syntax	<code>:SEARCh:SERial:LIN:PATTern:DATA <string></code>
	When :SEARCh:SERial:LIN:PATTern:FORMAT DECimal, <code><string> ::= "n"</code> where n ::= 32-bit integer in unsigned decimal
	When :SEARCh:SERial:LIN:PATTern:FORMAT HEX, <code><string> ::= "0xnn...n"</code> where n ::= {0,...,9 A,...,F X}

The :SEARCh:SERial:LIN:PATTern:DATA command specifies the data value when searching for LIN events.

The number base of the value entered with this command is specified using the :SEARCh:SERial:LIN:PATTern:FORMAT command. To set don't care values with the DATA command, the FORMAT must be HEX.

The length of the data value entered is specified using the :SEARCh:SERial:LIN:PATTern:DATA:LENGth command.

Query Syntax	<code>:SEARCh:SERial:LIN:PATTern:DATA?</code>
---------------------	---

The :SEARCh:SERial:LIN:PATTern:DATA? query returns the current data value setting.

Return Format	<code><string><NL></code>
	When :SEARCh:SERial:LIN:PATTern:FORMAT DECimal, <code><string> ::= "n"</code> where n ::= 32-bit integer in unsigned decimal or <code>"\$"</code> if data has any don't cares
	When :SEARCh:SERial:LIN:PATTern:FORMAT HEX, <code><string> ::= "0xnn...n"</code> where n ::= {0,...,9 A,...,F X}

See Also	<ul style="list-style-type: none"> • Chapter 26, “:SEARCh Commands,” starting on page 533 • ":SEARCh:SERial:LIN:MODE" on page 553 • ":SEARCh:SERial:LIN:PATTern:FORMAT" on page 556 • ":SEARCh:SERial:LIN:PATTern:DATA:LENGth" on page 555
-----------------	--

:SEARch:SERial:LIN:PATTern:DATA:LENGth

N (see [page 794](#))

Command Syntax `:SEARch:SERial:LIN:PATTern:DATA:LENGth <length>`
`<length> ::= integer from 1 to 8 in NR1 format`

The :SEARch:SERial:LIN:PATTern:DATA:LENGth command specifies the the length of the data value when searching for LIN events.

The data value is specified using the :SEARch:SERial:LIN:PATTern:DATA command.

Query Syntax `:SEARch:SERial:LIN:PATTern:DATA:LENGth?`

The :SEARch:SERial:LIN:PATTern:DATA:LENGth? query returns the current data value length setting.

Return Format `<length><NL>`
`<length> ::= integer from 1 to 8 in NR1 format`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:LIN:PATTern:DATA"](#) on page 554

:SEARch:SERial:LIN:PATTern:FORMAT**N** (see [page 794](#))

Command Syntax `:SEARch:SERial:LIN:PATTern:FORMAT <base>`
 `<base> ::= {HEX | DECimal}`

The :SEARch:SERial:LIN:PATTern:FORMAT command specifies the number base used with the :SEARch:SERial:LIN:PATTern:DATA command.

Query Syntax `:SEARch:SERial:LIN:PATTern:FORMAT?`

The :SEARch:SERial:LIN:PATTern:FORMAT? query returns the current number base setting.

Return Format `<base><NL>`
 `<base> ::= {HEX | DEC}`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:LIN:PATTern:DATA"](#) on page 554

:SEARch:SERial:SPI Commands

Table 85 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MO DE <value> (see page 558)	:SEARch:SERial:SPI:MO DE? (see page 558)	<value> ::= {MOSI MISO}
:SEARch:SERial:SPI:PA TTern:DATA <string> (see page 559)	:SEARch:SERial:SPI:PA TTern:DATA? (see page 559)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARch:SERial:SPI:PA TTern:WIDTH <width> (see page 560)	:SEARch:SERial:SPI:PA TTern:WIDTH? (see page 560)	<width> ::= integer from 1 to 10

:SEARch:SERial:SPI:MODE**N** (see [page 794](#))

Command Syntax `:SEARch:SERial:SPI:MODE <value>`
 `<value> ::= {MOSI | MISO}`

The :SEARch:SERial:SPI:MODE command specifies whether the SPI search will be on the MOSI data or the MISO data.

Data values are specified using the :SEARch:SERial:SPI:PATTERn:DATA command.

Query Syntax `:SEARch:SERial:SPI:MODE?`

The :SEARch:SERial:SPI:MODE? query returns the current SPI search mode setting.

Return Format `<value><NL>`
 `<value> ::= {MOSI | MISO}`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:SPI:PATTERn:DATA"](#) on page 559

:SEARch:SERial:SPI:PATTern:DATA**N** (see [page 794](#))**Command Syntax** `:SEARch:SERial:SPI:PATTern:DATA <string>``<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X}`

The :SEARch:SERial:SPI:PATTern:DATA command specifies the data value when searching for SPI events.

The width of the data value is specified using the :SEARch:SERial:SPI:PATTern:WIDTh command.

Query Syntax `:SEARch:SERial:SPI:PATTern:DATA?`

The :SEARch:SERial:SPI:PATTern:DATA? query returns the current data value setting.

Return Format `<string><NL>``<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X}`**See Also**

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:SPI:PATTern:WIDTh"](#) on page 560

:SEARch:SERial:SPI:PATTern:WIDTh

N (see [page 794](#))

Command Syntax `:SEARch:SERial:SPI:PATTern:WIDTh <width>`
 `<width> ::= integer from 1 to 10`

The :SEARch:SERial:SPI:PATTern:WIDTh command specifies the width of the data value (in bytes) when searching for SPI events.

The data value is specified using the :SEARch:SERial:SPI:PATTern:DATA command.

Query Syntax `:SEARch:SERial:SPI:PATTern:WIDTh?`

The :SEARch:SERial:SPI:PATTern:WIDTh? query returns the current data width setting.

Return Format `<width><NL>`
 `<width> ::= integer from 1 to 10`

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:SPI:PATTern:DATA"](#) on page 559

:SEARch:SERial:UART Commands

Table 86 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:D ATA <value> (see page 562)	:SEARch:SERial:UART:D ATA? (see page 562)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SEARch:SERial:UART:M ODE <value> (see page 563)	:SEARch:SERial:UART:M ODE? (see page 563)	<value> ::= {RDATa RD1 RD0 RDX TDATA TD1 TD0 TDX PARityerror AERRor}
:SEARch:SERial:UART:Q UALifier <value> (see page 564)	:SEARch:SERial:UART:Q UALifier? (see page 564)	<value> ::= {EQUAL NOTEqual GREaterthan LESSthan}

:SEARch:SERial:UART:DATA

N (see [page 794](#))

Command Syntax `:SEARch:SERial:UART:DATA <value>`

```
<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal,
           <hexadecimal>, <binary>, or <quoted_string> format

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

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

<quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or
                     standard abbreviations)
```

The :SEARch:SERial:UART:DATA command specifies a data value when searching for UART/RS232 events.

The data value qualifier is specified using the :SEARch:SERial:UART:QUALifier command.

Query Syntax `:SEARch:SERial:UART:DATA?`

The :SEARch:SERial:UART:DATA? query returns the current data value setting.

Return Format `<value><NL>`

```
<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal format
```

See Also

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:UART:MODE"](#) on page 563
- [":SEARch:SERial:UART:QUALifier"](#) on page 564

:SEARch:SERial:UART:MODE

N (see [page 794](#))

Command Syntax :SEARch:SERial:UART:MODE <value>

```
<value> ::= {RDATA | RD1 | RD0 | RDX | TDATA | TD1 | TD0 | TDX
              | PARityerror | AERRor}
```

The :SEARch:SERial:UART:MODE command selects the type of UART/RS232 information to find in the Lister display:

- RDATA – searches for a receive data value when data words are from 5 to 8 bits long.
- RD1 – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is 1.
- RD0 – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is 0.
- RDX – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is a don't care (X).
- TDATA – searches for a transmit data value when data words are from 5 to 8 bits long.
- TD1 – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is 1.
- TD0 – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is 0.
- TDX – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is a don't care (X).
- PARityerror – searches for parity errors.
- AERRor – searches for any error.

Data values are specified using the :SEARch:SERial:UART:DATA command.

Data value qualifiers are specified using the :SEARch:SERial:UART:QUALifier command.

Query Syntax :SEARch:SERial:UART:MODE?

The :SEARch:SERial:UART:MODE? query returns ...

Return Format <value><NL>

```
<value> ::= {RDAT | RD1 | RD0 | RDX | TDAT | TD1 | TD0 | TDX | PAR
              | AERR}
```

- See Also**
- [Chapter 26](#), “:SEARch Commands,” starting on page 533
 - [":SEARch:SERial:UART:DATA"](#) on page 562
 - [":SEARch:SERial:UART:QUALifier"](#) on page 564

:SEARch:SERial:UART:QUALifier**N** (see [page 794](#))**Command Syntax** `:SEARch:SERial:UART:QUALifier <value>``<value> ::= {EQUAL | NOTEqual | GREaterthan | LESSthan}`

The :SEARch:SERial:UART:QUALifier command specifies the data value qualifier when searching for UART/RS232 events.

Query Syntax `:SEARch:SERial:UART:QUALifier?`

The :SEARch:SERial:UART:QUALifier? query returns the current data value qualifier setting.

Return Format `<value><NL>``<value> ::= {EQU | NOT | GRE | LESS}`**See Also**

- [Chapter 26](#), “:SEARch Commands,” starting on page 533
- [":SEARch:SERial:UART:DATA"](#) on page 562

27 :SYSTem Commands

Control basic system functions of the oscilloscope. See "[Introduction to :SYSTem Commands](#)" on page 566.

Table 87 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 567)	:SYSTem:DATE? (see page 567)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,...,12 JANuary FEBruary MARch APRil MAY JUNe JULy AUGust SEPtember OCTober NOVember DECember} <day> ::= {1,...31}
:SYSTem:DSP <string> (see page 568)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 569)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 753).
:SYSTem:LOCK <value> (see page 570)	:SYSTem:LOCK? (see page 570)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:MENU <menu> (see page 571)	n/a	<menu> ::= {MASK MEASure SEGmented LISTer}
:SYSTem:PRESet (see page 572)	n/a	See :SYSTem:PRESet (see page 572)
:SYSTem:PROTection:LOCK <value> (see page 575)	:SYSTem:PROTection:LOCK? (see page 575)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:SETup <setup_data> (see page 576)	:SYSTem:SETup? (see page 576)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 578)	:SYSTem:TIME? (see page 578)	<time> ::= hours,minutes,seconds in NR1 format

Introduction to :SYSTem Commands SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

:SYSTem:DATE

N (see [page 794](#))

Command Syntax :SYSTem:DATE <date>

```
<date> ::= <year>,<month>,<day>
<year> ::= 4-digit year in NR1 format
<month> ::= {1,...,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNE
              | JULY | AUGust | SEPtember | OCTober | NOVember | DECember}
<day> ::= {1,...,31}
```

The :SYSTem:DATE command sets the date. Validity checking is performed to ensure that the date is valid.

Query Syntax :SYSTem:DATE?

The SYSTem:DATE? query returns the date.

Return Format <year>,<month>,<day><NL>

See Also

- "Introduction to :SYSTem Commands" on page 566
- ":SYSTem:TIME" on page 578

:SYSTem:DSP

N (see [page 794](#))

Command Syntax `:SYSTem:DSP <string>`

`<string>` ::= quoted ASCII string (up to 75 characters)

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYSTem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

See Also • "Introduction to :SYSTem Commands" on page 566

:SYSTem:ERRor



(see page 794)

Query Syntax :SYSTem:ERRor?

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

Return Format	<pre><error number>,<error string><NL> <error number> ::= an integer error code in NR1 format <error string> ::= quoted ASCII string containing the error message</pre> <p>Error messages are listed in Chapter 34, “Error Messages,” starting on page 753.</p>
See Also	<ul style="list-style-type: none">“Introduction to :SYSTem Commands” on page 566“*ESR (Standard Event Status Register)” on page 126“*CLS (Clear Status)” on page 123

:SYSTem:LOCK

N (see [page 794](#))

Command Syntax `:SYSTem:LOCK <value>`

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

The :SYSTem:LOCK command disables the front panel. LOCK ON is the equivalent of sending a local lockout message over the programming interface.

Query Syntax `:SYSTem:LOCK?`

The :SYSTem:LOCK? query returns the lock status of the front panel.

Return Format `<value><NL>`

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

See Also • "Introduction to :SYSTem Commands" on page 566

:SYSTem:MENU**N** (see [page 794](#))**Command Syntax** `:SYSTem:MENU <menu>``<menu> ::= {MASK | MEASure | SEGmented | LISTer}`

The :SYSTem:MENU command changes the front panel softkey menu.

:SYSTem:PRESet

C (see [page 794](#))

Command Syntax :SYSTem:PRESet

The :SYSTem:PRESet command places the instrument in a known state. This is the same as pressing the **[Default Setup]** key or **[Save/Recall] > Default/Erase > Default Setup** on the front panel.

When you perform a default setup, some user settings (like preferences) remain unchanged. To reset all user settings to their factory defaults, use the *RST command.

Reset conditions are:

Acquire Menu	
Mode	Normal
Averaging	Off
# Averages	8

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

Cursor Menu	
Source	Channel 1

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

Display Menu	
Persistence	Off
Grid	20%

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

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

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

- See Also
- ["Introduction to Common \(*\) Commands"](#) on page 121
 - ["*RST \(Reset\)"](#) on page 134

:SYSTem:PROTection:LOCK

N (see [page 794](#))

Command Syntax :SYSTem:PROTection:LOCK <value>

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

The :SYSTem:PROTection:LOCK command disables the fifty ohm impedance setting for all analog channels.

Query Syntax :SYSTem:PROTection:LOCK?

The :SYSTem:PROTection:LOCK? query returns the analog channel protection lock status.

Return Format <value><NL>

<value> ::= {1 | 0}

See Also · "Introduction to :SYSTem Commands" on page 566

:SYSTem:SEtup

 (see [page 794](#))

Command Syntax	<pre>:SYSTem:SEtup <setup_data> <setup_data> ::= binary block data in IEEE 488.2 # format.</pre>
	The :SYSTem:SEtup command sets the oscilloscope as defined by the data in the setup (learn) string sent from the controller. The setup string does not change the interface mode or interface address.
Query Syntax	<pre>:SYSTem:SEtup?</pre>
	The :SYSTem:SEtup? query operates the same as the *LRN? query. It outputs the current oscilloscope setup in the form of a learn string to the controller. The setup (learn) string is sent and received as a binary block of data. The format for the data transmission is the # format defined in the IEEE 488.2 specification.
Return Format	<pre><setup_data><NL> <setup_data> ::= binary block data in IEEE 488.2 # format</pre>
See Also	<ul style="list-style-type: none"> · "Introduction to :SYSTem Commands" on page 566 · "*LRN (Learn Device Setup)" on page 129
Example Code	<pre>' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program ' message that contains the current state of the instrument. Its ' format is a definite-length binary block, for example, ' #800075595<setup string><NL> ' where the setup string is 75595 bytes in length. myScope.WriteString ":SYSTEM:SETUP?" varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1) CheckForInstrumentErrors ' After reading query results. ' Output setup string to a file: Dim strPath As String strPath = "c:\scope\config\setup.dat" ' Open file for output. Close #1 ' If #1 is open, close it. Open strPath For Binary Access Write Lock Write As #1 Put #1, , varQueryResult ' Write data. Close #1 ' Close file. ' RESTORE_SYSTEM_SETUP - Read the setup string from a file and ' write it back to the oscilloscope. Dim varSetupString As Variant strPath = "c:\scope\config\setup.dat" ' Open file for input. Open strPath For Binary Access Read As #1 Get #1, , varSetupString ' Read data. Close #1 ' Close file. ' Write setup string back to oscilloscope using ":SYSTEM:SETUP"</pre>

```
' command:  
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString  
CheckForInstrumentErrors
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:SYSTem:TIME**N** (see [page 794](#))**Command Syntax** `:SYSTem:TIME <time>``<time> ::= hours,minutes,seconds in NR1 format`

The :SYSTem:TIME command sets the system time, using a 24-hour format. Commas are used as separators. Validity checking is performed to ensure that the time is valid.

Query Syntax `:SYSTem:TIME? <time>`

The :SYSTem:TIME? query returns the current system time.

Return Format `<time><NL>``<time> ::= hours,minutes,seconds in NR1 format`**See Also**

- "Introduction to :SYSTem Commands" on page 566
- "[:SYSTem:DATE](#)" on page 567

28 :TIMEbase Commands

Control all horizontal sweep functions. See "[Introduction to :TIMEbase Commands](#)" on page 580.

Table 88 :TIMEbase Commands Summary

Command	Query	Options and Query Returns
:TIMEbase:MODE <value> (see page 581)	:TIMEbase:MODE? (see page 581)	<value> ::= {MAIN WINDOW XY ROLL}
:TIMEbase:POSITION <pos> (see page 582)	:TIMEbase:POSITION? (see page 582)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMEbase:RANGE <range_value> (see page 583)	:TIMEbase:RANGE? (see page 583)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMEbase:REFERENCE {LEFT CENTER RIGHT} (see page 584)	:TIMEbase:REFERENCE? (see page 584)	<return_value> ::= {LEFT CENTER RIGHT}
:TIMEbase:SCALE <scale_value> (see page 585)	:TIMEbase:SCALE? (see page 585)	<scale_value> ::= time/div in seconds in NR3 format
:TIMEbase:VERNier {{0 OFF} {1 ON}} (see page 586)	:TIMEbase:VERNier? (see page 586)	{0 1}
:TIMEbase:WINDOW:POSITION <pos> (see page 587)	:TIMEbase:WINDOW:POSITION? (see page 587)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMEbase:WINDOW:RANGE <range_value> (see page 588)	:TIMEbase:WINDOW:RANGE? (see page 588)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMEbase:WINDOW:SCALE <scale_value> (see page 589)	:TIMEbase:WINDOW:SCALE? (see page 589)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Introduction to :T1Mebase Commands The T1Mebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (zoomed) time bases.

Reporting the Setup

Use :T1Mebase? to query setup information for the T1Mebase subsystem.

Return Format

The following is a sample response from the :T1Mebase? query. In this case, the query was issued following a *RST command.

```
:T1M:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
```

:TIMEbase:MODE

C (see [page 794](#))

Command Syntax `:TIMEbase:MODE <value>`

`<value> ::= {MAIN | WINDow | XY | ROLL}`

The :TIMEbase:MODE command sets the current time base. There are four time base modes:

- MAIN – The normal time base mode is the main time base. It is the default time base mode after the *RST (Reset) command.
- WINDow – In the WINDow (zoomed or delayed) time base mode, measurements are made in the zoomed time base if possible; otherwise, the measurements are made in the main time base.
- XY – In the XY mode, the :TIMEbase:RANGE, :TIMEbase:POSITION, and :TIMEbase:REFERENCE commands are not available. No measurements are available in this mode.
- ROLL – In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMEbase:REFERENCE selection changes to RIGHT.

Query Syntax `:TIMEbase:MODE?`

The :TIMEbase:MODE query returns the current time base mode.

Return Format `<value><NL>`

`<value> ::= {MAIN | WIND | XY | ROLL}`

See Also

- "[Introduction to :TIMEbase Commands](#)" on page 580
- "[*RST \(Reset\)](#)" on page 134
- "[:TIMEbase:RANGE](#)" on page 583
- "[:TIMEbase:POSITION](#)" on page 582
- "[:TIMEbase:REFERENCE](#)" on page 584

Example Code

```
' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
myScope.WriteString ":TIMEBASE:MODE MAIN"
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:TIMEbase:POSITION

 (see [page 794](#))

Command Syntax `:TIMEbase:POSITION <pos>`

`<pos>` ::= time in seconds from the trigger to the display reference
in NR3 format

The :TIMEbase:POSITION command sets the time interval between the trigger event and the display reference point on the screen. The display reference point is either left, right, or center and is set with the :TIMEbase:REFERENCE command. The maximum position value depends on the time/division settings.

NOTE

This command is an alias for the :TIMEbase:DELAY command.

Query Syntax `:TIMEbase:POSITION?`

The :TIMEbase:POSITION? query returns the current time from the trigger to the display reference in seconds.

Return Format `<pos><NL>`

`<pos>` ::= time in seconds from the trigger to the display reference
in NR3 format

See Also

- "[Introduction to :TIMEbase Commands](#)" on page 580
- "[:TIMEbase:REFERENCE](#)" on page 584
- "[:TIMEbase:RANGE](#)" on page 583
- "[:TIMEbase:SCALE](#)" on page 585
- "[:TIMEbase:WINDOW:POSITION](#)" on page 587
- "[:TIMEbase:DELAY](#)" on page 749

:TIMEbase:RANGE

C (see [page 794](#))

Command Syntax `:TIMEbase:RANGE <range_value>`

`<range_value> ::= time for 10 div in seconds in NR3 format`

The :TIMEbase:RANGE command sets the full-scale horizontal time in seconds for the main window. The range is 10 times the current time-per-division setting.

Query Syntax `:TIMEbase:RANGE?`

The :TIMEbase:RANGE query returns the current full-scale range value for the main window.

Return Format `<range_value><NL>`

`<range_value> ::= time for 10 div in seconds in NR3 format`

- See Also**
- "[Introduction to :TIMEbase Commands](#)" on page 580
 - "[:TIMEbase:MODE](#)" on page 581
 - "[:TIMEbase:SCALe](#)" on page 585
 - "[:TIMEbase:WINDOW:RANGE](#)" on page 588

Example Code

```
' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
' range value is 10 times the time per division.
myScope.WriteString ":TIM:RANG 2e-3"    ' Set the time range to 0.002
seconds.
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:TImebase:REFerence

C (see [page 794](#))

Command Syntax `:TImebase:REFerence <reference>`

`<reference> ::= {LEFT | CENTER | RIGHT}`

The :TImebase:REFerence command sets the time reference to one division from the left side of the screen, to the center of the screen, or to one division from the right side of the screen. Time reference is the point on the display where the trigger point is referenced.

Query Syntax `:TImebase:REFerence?`

The :TImebase:REFerence? query returns the current display reference for the main window.

Return Format `<reference><NL>`

`<reference> ::= {LEFT | CENT | RIGH}`

- See Also**
- "Introduction to :TImebase Commands" on page 580
 - "[:TImebase:MODE](#)" on page 581

Example Code

```
' TIME_REFERENCE - Possible values are LEFT, CENTER, or RIGHT.
'   - LEFT sets the display reference one time division from the left.
'   - CENTER sets the display reference to the center of the screen.
'   - RIGHT sets the display reference one time division from the right.
myScope.WriteString ":TImebase:REFerence CENTER"    ' Set reference to center.
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:TIMEbase:SCALe**N** (see [page 794](#))**Command Syntax** `:TIMEbase:SCALe <scale_value>``<scale_value> ::= time/div in seconds in NR3 format`

The :TIMEbase:SCALe command sets the horizontal scale or units per division for the main window.

Query Syntax `:TIMEbase:SCALe?`

The :TIMEbase:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

Return Format `<scale_value><NL>``<scale_value> ::= time/div in seconds in NR3 format`**See Also**

- "Introduction to :TIMEbase Commands" on page 580
- ":TIMEbase:RANGE" on page 583
- ":TIMEbase:WINDOW:SCALe" on page 589
- ":TIMEbase:WINDOW:RANGE" on page 588

:TIMEbase:VERNier**N** (see [page 794](#))**Command Syntax** `:TIMEbase:VERNier <vernier value>``<vernier value> ::= {{1 | ON} | {0 | OFF}}`

The :TIMEbase:VERNier command specifies whether the time base control's vernier (fine horizontal adjustment) setting is ON (1) or OFF (0).

Query Syntax `:TIMEbase:VERNier?`

The :TIMEbase:VERNier? query returns the current state of the time base control's vernier setting.

Return Format `<vernier value><NL>``<vernier value> ::= {0 | 1}`**See Also** · "Introduction to :TIMEbase Commands" on page 580

:TIMEbase:WINDOW:POSITION

C (see [page 794](#))

Command Syntax :TIMEbase:WINDOW:POSITION <pos value>

<pos value> ::= time from the trigger event to the zoomed (delayed) view reference point in NR3 format

The :TIMEbase:WINDOW:POSITION command sets the horizontal position in the zoomed (delayed) view of the main sweep. The main sweep range and the main sweep horizontal position determine the range for this command. The value for this command must keep the zoomed view window within the main sweep range.

Query Syntax :TIMEbase:WINDOW:POSITION?

The :TIMEbase:WINDOW:POSITION? query returns the current horizontal window position setting in the zoomed view.

Return Format <value><NL>

<value> ::= position value in seconds

- See Also**
- "[Introduction to :TIMEbase Commands](#)" on page 580
 - "[:TIMEbase:MODE](#)" on page 581
 - "[:TIMEbase:POStion](#)" on page 582
 - "[:TIMEbase:RANGE](#)" on page 583
 - "[:TIMEbase:SCALe](#)" on page 585
 - "[:TIMEbase:WINDOW:RANGE](#)" on page 588
 - "[:TIMEbase:WINDOW:SCALe](#)" on page 589

:TIMEbase:WINDOW:RANGE(see [page 794](#))**Command Syntax** `:TIMEbase:WINDOW:RANGE <range value>``<range value> ::= range value in seconds in NR3 format`

The :TIMEbase:WINDOW:RANGE command sets the full-scale horizontal time in seconds for the zoomed (delayed) window. The range is 10 times the current zoomed view window seconds per division setting. The main sweep range determines the range for this command. The maximum value is one half of the :TIMEbase:RANGE value.

Query Syntax `:TIMEbase:WINDOW:RANGE?`

The :TIMEbase:WINDOW:RANGE? query returns the current window timebase range setting.

Return Format `<value><NL>``<value> ::= range value in seconds`**See Also**

- ["Introduction to :TIMEbase Commands"](#) on page 580
- [":TIMEbase:RANGE"](#) on page 583
- [":TIMEbase:POSIon"](#) on page 582
- [":TIMEbase:SCALe"](#) on page 585

:TIMEbase:WINDOW:SCALe

N (see [page 794](#))

Command Syntax	<code>:TIMEbase:WINDOW:SCALe <scale_value></code> <code><scale_value> ::= scale value in seconds in NR3 format</code>
	The :TIMEbase:WINDOW:SCALe command sets the zoomed (delayed) window horizontal scale (seconds/division). The main sweep scale determines the range for this command. The maximum value is one half of the :TIMEbase:SCALe value.
Query Syntax	<code>:TIMEbase:WINDOW:SCALe?</code>
	The :TIMEbase:WINDOW:SCALe? query returns the current zoomed window scale setting.
Return Format	<code><scale_value><NL></code> <code><scale_value> ::= current seconds per division for the zoomed window</code>
See Also	<ul style="list-style-type: none"> • "Introduction to :TIMEbase Commands" on page 580 • ":TIMEbase:RANGE" on page 583 • ":TIMEbase:POStion" on page 582 • ":TIMEbase:SCALe" on page 585 • ":TIMEbase:WINDOW:RANGE" on page 588

29 :TRIGger Commands

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

- ["Introduction to :TRIGger Commands" on page 591](#)
- ["General :TRIGger Commands" on page 593](#)
- [":TRIGger\[:EDGE\] Commands" on page 603](#)
- [":TRIGger:GLITch Commands" on page 609 \(Pulse Width trigger\)](#)
- [":TRIGger:PATTERn Commands" on page 618](#)
- [":TRIGger:TV Commands" on page 623](#)

Introduction to :TRIGger Commands

The commands in the TRIGger subsystem define the conditions for an internal trigger. Many of these commands are valid in multiple trigger modes.

The default trigger mode is :EDGE.

The trigger subsystem controls the trigger sweep mode and the trigger specification. The trigger sweep (see [":TRIGger:SWEep" on page 602](#)) can be AUTO or NORMal.

- **NORMal** mode – displays a waveform only if a trigger signal is present and the trigger conditions are met. Otherwise the oscilloscope does not trigger and the display is not updated. This mode is useful for low-repetitive-rate signals.
- **AUTO** trigger mode – generates an artificial trigger event if the trigger specification is not satisfied within a preset time, acquires unsynchronized data and displays it.

AUTO mode is useful for signals other than low-repetitive-rate signals. You must use this mode to display a DC signal because there are no edges on which to trigger.

The following trigger types are available (see [":TRIGger:MODE" on page 600](#)).

- **Edge triggering**— identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Pulse width triggering**— (:TRIGger:GLITch commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.
- **Pattern triggering**— identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels. You can also trigger on a specified time duration of a pattern.

- **TV triggering**— is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than $\frac{1}{4}$ division of sync amplitude with any analog channel as the trigger source.

Reporting the Setup

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

Return Format

The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a *RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.000000000000E-09;  
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```

General :TRIGger Commands

Table 89 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 594)	n/a	n/a
:TRIGger:HFReject {{0 OFF} {1 ON}} (see page 595)	:TRIGger:HFReject? (see page 595)	{0 1}
:TRIGger:HOLDoff <holdoff_time> (see page 596)	:TRIGger:HOLDoff? (see page 596)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:ASETup (see page 597)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 598)	:TRIGger:LEVel:HIGH? <source> (see page 598)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see page 599)	:TRIGger:LEVel:LOW? <source> (see page 599)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:MODE <mode> (see page 600)	:TRIGger:MODE? (see page 600)	<mode> ::= {EDGE GLITCH PATTerm TV} <return_value> ::= {<mode> <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREJect {{0 OFF} {1 ON}} (see page 601)	:TRIGger:NREJect? (see page 601)	{0 1}
:TRIGger:SWEep <sweep> (see page 602)	:TRIGger:SWEep? (see page 602)	<sweep> ::= {AUTO NORMAL}

:TRIGger:FORCe

N (see [page 794](#))

Command Syntax :TRIGger:FORCe

The :TRIGger:FORCe command causes an acquisition to be captured even though the trigger condition has not been met. This command is equivalent to the front panel **[Force Trigger]** key.

See Also • ["Introduction to :TRIGger Commands"](#) on page 591

:TRIGger:HFReject**C** (see [page 794](#))**Command Syntax** `:TRIGger:HFReject <value>``<value> ::= {{0 | OFF} | {1 | ON}}`

The :TRIGger:HFReject command turns the high frequency reject filter off and on. The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use this filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.

Query Syntax `:TRIGger:HFReject?`

The :TRIGger:HFReject? query returns the current high frequency reject filter mode.

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

- ["Introduction to :TRIGger Commands"](#) on page 591
- [":TRIGger\[:EDGE\]:REject"](#) on page 606

:TRIGger:HOLDoff**C** (see [page 794](#))**Command Syntax** `:TRIGger:HOLDoff <holdoff_time>``<holdoff_time> ::= 60 ns to 10 s in NR3 format`

The :TRIGger:HOLDoff command defines the holdoff time value in seconds. Holdoff keeps a trigger from occurring until after a certain amount of time has passed since the last trigger. This feature is valuable when a waveform crosses the trigger level multiple times during one period of the waveform. Without holdoff, the oscilloscope could trigger on each of the crossings, producing a confusing waveform. With holdoff set correctly, the oscilloscope always triggers on the same crossing. The correct holdoff setting is typically slightly less than one period.

Query Syntax `:TRIGger:HOLDoff?`

The :TRIGger:HOLDoff? query returns the holdoff time value for the current trigger mode.

Return Format `<holdoff_time><NL>``<holdoff_time> ::= the holdoff time value in seconds in NR3 format.`**See Also**

- ["Introduction to :TRIGger Commands"](#) on page 591

:TRIGger:LEVel:ASETup

N (see [page 794](#))

Command Syntax :TRIGger:LEVel:ASETup

The :TRIGger:LEVel:ASETup command automatically sets the trigger levels of all displayed analog channels to their waveforms' 50% values.

If AC coupling is used, the trigger levels are set to 0 V.

When High and Low (dual) trigger levels are used (as with Rise/Fall Time and Runt triggers, for example), this command has no effect.

See Also • [":TRIGger\[:EDGE\]:LEVel"](#) on page 605

:TRIGger:LEVel:HIGH**N** (see [page 794](#))

- Command Syntax** `:TRIGger:LEVel:HIGH <level>, <source>`
- `<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
for internal triggers`
- `<source> ::= CHANnel<n>`
- `<n> ::= 1 to (# analog channels) in NR1 format`
- The :TRIGger:LEVel:HIGH command sets the high trigger voltage level voltage for the specified source.
- Query Syntax** `:TRIGger:LEVel:HIGH? <source>`
- The :TRIGger:LEVel:HIGH? query returns the high trigger voltage level for the specified source.
- Return Format** `<level><NL>`
- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[":TRIGger:LEVel:LOW](#)" on page 599
 - "[":TRIGger\[:EDGE\]:SOURce](#)" on page 608

:TRIGger:LEVel:LOW

N (see [page 794](#))

Command Syntax	<code>:TRIGger:LEVel:LOW <level>, <source></code>
	<code><level> ::= 0.75 x full-scale voltage from center screen in NR3 format for internal triggers</code>
	<code><source> ::= CHANnel<n></code>
	<code><n> ::= 1 to (# analog channels) in NR1 format</code>
	The :TRIGger:LEVel:LOW command sets the low trigger voltage level voltage for the specified source.
Query Syntax	<code>:TRIGger:LEVel:LOW? <source></code>
	The :TRIGger:LEVel:LOW? query returns the low trigger voltage level for the specified source.
Return Format	<code><level><NL></code>
See Also	<ul style="list-style-type: none"> • "Introduction to :TRIGger Commands" on page 591 • "":TRIGger:LEVel:HIGH" on page 598 • "":TRIGger[:EDGE]:SOURce" on page 608

:TRIGger:MODE (see [page 794](#))**Command Syntax** `:TRIGger:MODE <mode>``<mode> ::= {EDGE | GLITch | PATTern | TV}`

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax `:TRIGger:MODE?`

The :TRIGger:MODE? query returns the current trigger mode. If the :TIMEbase:MODE is ROLL or XY, the query returns "NONE".

Return Format `<mode><NL>``<mode> ::= {EDGE | GLIT | PATT | TV}`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:SWEep](#)" on page 602
 - "[:TIMEbase:MODE](#)" on page 581

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE.  
myScope.WriteString ":TRIGger:MODE EDGE"
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:TRIGger:NREject(see [page 794](#))**Command Syntax** `:TRIGger:NREject <value>``<value> ::= {{0 | OFF} | {1 | ON}}`

The :TRIGger:NREject command turns the noise reject filter off and on. When the noise reject filter is on, the trigger circuitry is less sensitive to noise but may require a greater amplitude waveform to trigger the oscilloscope. This command is not valid in TV trigger mode.

Query Syntax `:TRIGger:NREject?`

The :TRIGger:NREject? query returns the current noise reject filter mode.

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

- ["Introduction to :TRIGger Commands"](#) on page 591

:TRIGger:SWEep(see [page 794](#))**Command Syntax** `:TRIGger:SWEep <sweep>``<sweep> ::= {AUTO | NORMAl}`

The :TRIGger:SWEep command selects the trigger sweep mode.

When AUTO sweep mode is selected, a baseline is displayed in the absence of a signal. If a signal is present but the oscilloscope is not triggered, the unsynchronized signal is displayed instead of a baseline.

When NORMAl sweep mode is selected and no trigger is present, the instrument does not sweep, and the data acquired on the previous trigger remains on the screen.

NOTE

This feature is called "Mode" on the instrument's front panel.

Query Syntax `:TRIGger:SWEep?`

The :TRIGger:SWEep? query returns the current trigger sweep mode.

Return Format `<sweep><NL>``<sweep> ::= current trigger sweep mode`**See Also**

- ["Introduction to :TRIGger Commands"](#) on page 591

:TRIGger[:EDGE] Commands

Table 90 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE] :COUPLing {AC DC LFReject} (see page 604)	:TRIGger[:EDGE] :COUPLing? (see page 604)	{AC DC LFReject}
:TRIGger[:EDGE] :LEVel <level> [,<source>] (see page 605)	:TRIGger[:EDGE] :LEVel? [<source>] (see page 605)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> EXTERNAL} for DSO models <source> ::= {CHANnel<n> DIGital<d> EXTERNAL } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE] :REJect {OFF LFReject HFReject} (see page 606)	:TRIGger[:EDGE] :REJect? (see page 606)	{OFF LFReject HFReject}
:TRIGger[:EDGE] :SLOPe <polarity> (see page 607)	:TRIGger[:EDGE] :SLOPe? (see page 607)	<polarity> ::= {POSitive NEGative EITHer ALTernate}
:TRIGger[:EDGE] :SOURce <source> (see page 608)	:TRIGger[:EDGE] :SOURce? (see page 608)	<source> ::= {CHANnel<n> EXTERNAL LINE WGEN} for the DSO models <source> ::= {CHANnel<n> DIGital<d> EXTERNAL LINE WGEN} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

:TRIGger[:EDGE]:COUpling



(see [page 794](#))

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

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

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

NOTE

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

Query Syntax `:TRIGger[:EDGE]:COUpling?`

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

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

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:MODE](#)" on page 600
- "[:TRIGger\[:EDGE\]:REJect](#)" on page 606

:TRIGger[:EDGE]:LEVel

C (see [page 794](#))

Command Syntax

```
:TRIGger[:EDGE]:LEVel <level>
<level> ::= <level>[,<source>]
<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
           for internal triggers
<level> ::= ±(external range setting) in NR3 format
           for external triggers
<level> ::= ±8 V for digital channels (MSO models)
<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models
<source> ::= {CHANnel<n> | DIGital<d> | EXTERNAL}
           for the MSO models
<n> ::= 1 to (# analog channels) in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

NOTE

If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

Query Syntax

:TRIGger[:EDGE]:LEVel? [<source>]

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

Return Format

<level><NL>

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger\[:EDGE\]:SOURce](#)" on page 608
- "[:EXTERNAL:RANGE](#)" on page 276
- "[:POD<n>:THRESHOLD](#)" on page 417
- "[:DIGITAL<d>:THRESHOLD](#)" on page 249

:TRIGger[:EDGE]:REJect

 (see [page 794](#))

Command Syntax `:TRIGger[:EDGE]:REJect <reject>`

`<reject> ::= {OFF | LFRReject | HFReject}`

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

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

NOTE

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

Query Syntax `:TRIGger[:EDGE]:REJect?`

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

Return Format `<reject><NL>`

`<reject> ::= {OFF | LFR | HFR}`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:HFReject](#)" on page 595
 - "[:TRIGger\[:EDGE\]:COUPLing](#)" on page 604

:TRIGger[:EDGE]:SLOPe

C (see [page 794](#))

Command Syntax :TRIGger[:EDGE]:SLOPe <slope>

<slope> ::= {NEGative | POSitive | EITHer | ALTernate}

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

Query Syntax :TRIGger[:EDGE]:SLOPe?

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

Return Format <slope><NL>

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

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:MODE](#)" on page 600
 - "[:TRIGger:TV:POLarity](#)" on page 626

Example Code

```
' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.  
  
' Set the slope to positive.  
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:TRIGger[:EDGE]:SOURce

C (see [page 794](#))

Command Syntax `:TRIGger[:EDGE]:SOURce <source>`

```
<source> ::= {CHANnel<n> | EXTernal | LINE | WGEN} for the DSO models
<source> ::= {CHANnel<n> | DIGItal<d> | EXTernal | LINE | WGEN}
            for the MSO models
```

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

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

The :TRIGger[:EDGE]:SOURce command selects the input that produces the trigger.

- EXTernal – triggers on the rear panel EXT TRIG IN signal.
- LINE – triggers at the 50% level of the rising or falling edge of the AC power source signal.
- WGEN – triggers at the 50% level of the rising edge of the waveform generator output signal. This option is not available when the DC or NOISe waveforms are selected.

Query Syntax `:TRIGger[:EDGE]:SOURce?`

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

Return Format `<source><NL>`

```
<source> ::= {CHAN<n> | EXT | LINE | WGEN | NONE} for the DSO models
<source> ::= {CHAN<n> | DIG<d> | EXTernal | LINE | WGEN | NONE}
            for the MSO models
```

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:MODE](#)" on page 600

Example Code

```
' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces the
' edge trigger. Any channel can be selected.
myScope.WriteString ":TRIGger:EDGE:SOURce CHANnel1"
```

See complete example programs at: [Chapter 38, “Programming Examples,”](#) starting on page 803

:TRIGger:GLITch Commands

Table 91 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 611)	:TRIGger:GLITch:GREaterthan? (see page 611)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 612)	:TRIGger:GLITch:LESSthan? (see page 612)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 613)	:TRIGger:GLITch:LEVel? (see page 613)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> EXTERNAL} for DSO models <source> ::= {CHANnel<n> DIGITAL<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 614)	:TRIGger:GLITch:POLarity? (see page 614)	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 615)	:TRIGger:GLITch:QUALifier? (see page 615)	<qualifier> ::= {GREaterthan LESSthan RANGE}

Table 91 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 616)	:TRIGger:GLITch:RANGE? ? (see page 616)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:SOURce <source> (see page 617)	:TRIGger:GLITch:SOURce? ? (see page 617)	<source> ::= {CHANnel<n> DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

:TRIGger:GLITch:GREaterthan

N (see [page 794](#))

Command Syntax :TRIGger:GLITch:GREaterthan <greater_than_time>[<suffix>]
 <greater_than_time> ::= floating-point number in NR3 format
 <suffix> ::= {s | ms | us | ns | ps}

The :TRIGger:GLITch:GREaterthan command sets the minimum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax :TRIGger:GLITch:GREaterthan?

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

Return Format <greater_than_time><NL>
 <greater_than_time> ::= floating-point number in NR3 format.

See Also

- "[Introduction to :TRIGger Commands](#)" on page 591
- "[:TRIGger:GLITch:SOURce](#)" on page 617
- "[:TRIGger:GLITch:QUALifier](#)" on page 615
- "[:TRIGger:MODE](#)" on page 600

:TRIGger:GLITch:LESSthan

N (see [page 794](#))

Command Syntax `:TRIGger:GLITch:LESSthan <less_than_time>[<suffix>]`

`<less_than_time>` ::= floating-point number in NR3 format

`<suffix>` ::= {s | ms | us | ns | ps}

The :TRIGger:GLITch:LESSthan command sets the maximum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax `:TRIGger:GLITch:LESSthan?`

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

Return Format `<less_than_time><NL>`

`<less_than_time>` ::= floating-point number in NR3 format.

See Also

- "Introduction to :TRIGger Commands" on page 591
- ":TRIGger:GLITch:SOURce" on page 617
- ":TRIGger:GLITch:QUALifier" on page 615
- ":TRIGger:MODE" on page 600

:TRIGger:GLITch:LEVel

N (see [page 794](#))

Command Syntax `:TRIGger:GLITch:LEVel <level_argument>`

`<level_argument> ::= <level>[, <source>]`

`<level> ::= .75 x full-scale voltage from center screen in NR3 format
for internal triggers`

`<level> ::= ±(external range setting) in NR3 format
for external triggers (DSO models)`

`<level> ::= ±8 V for digital channels (MSO models)`

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

`<source> ::= {CHANnel<n> | DIGital<d>} for MSO models`

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

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

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the active pulse width trigger.

Query Syntax `:TRIGger:GLITch:LEVel?`

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

Return Format `<level_argument><NL>`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[":TRIGger:MODE"](#) on page 600
 - "[":TRIGger:GLITch:SOURce"](#) on page 617
 - "[":EXTERNAL:RANGE"](#) on page 276

:TRIGger:GLITch:POLarity**N** (see [page 794](#))

Command Syntax `:TRIGger:GLITch:POLarity <polarity>`
`<polarity> ::= {POSitive | NEGative}`

The :TRIGger:GLITch:POLarity command sets the polarity for the glitch pulse width trigger.

Query Syntax `:TRIGger:GLITch:POLarity?`

The :TRIGger:GLITch:POLarity? query returns the glitch pulse width trigger polarity.

Return Format `<polarity><NL>`
`<polarity> ::= {POS | NEG}`

See Also

- ["Introduction to :TRIGger Commands"](#) on page 591
- [":TRIGger:MODE"](#) on page 600
- [":TRIGger:GLITch:SOURce"](#) on page 617

:TRIGger:GLITch:QUALifier**N** (see [page 794](#))**Command Syntax** `:TRIGger:GLITch:QUALifier <operator>``<operator> ::= {GREaterthan | LESSthan | RANGE}`

This command sets the mode of operation of the glitch pulse width trigger. The oscilloscope can trigger on a pulse width that is greater than a time value, less than a time value, or within a range of time values.

Query Syntax `:TRIGger:GLITch:QUALifier?`

The `:TRIGger:GLITch:QUALifier?` query returns the glitch pulse width qualifier.

Return Format `<operator><NL>``<operator> ::= {GRE | LESS | RANG}`

- See Also**
- "Introduction to [:TRIGger Commands](#)" on page 591
 - "[:TRIGger:GLITch:SOURce](#)" on page 617
 - "[:TRIGger:MODE](#)" on page 600

:TRIGger:GLITch:RANGE

N (see [page 794](#))

Command Syntax `:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix]`

`<less_than_time>` ::= (15 ns - 10 seconds) in NR3 format

`<greater_than_time>` ::= (10 ns - 9.99 seconds) in NR3 format

`[suffix]` ::= {s | ms | us | ns | ps}

The :TRIGger:GLITch:RANGE command sets the pulse width duration for the selected :TRIGger:GLITch:SOURce. You can enter the parameters in any order – the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax `:TRIGger:GLITch:RANGE?`

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

Return Format `<less_than_time>, <greater_than_time><NL>`

See Also

- ["Introduction to :TRIGger Commands"](#) on page 591
- [":TRIGger:GLITch:SOURce"](#) on page 617
- [":TRIGger:GLITch:QUALifier"](#) on page 615
- [":TRIGger:MODE"](#) on page 600

:TRIGger:GLITch:SOURce

N (see [page 794](#))

Command Syntax :TRIGger:GLITch:SOURce <source>

```
<source> ::= {DIGItal<d> | CHANnel<n>}  
<n> ::= 1 to (# analog channels) in NR1 format  
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :TRIGger:GLITch:SOURce command selects the channel that produces the pulse width trigger.

Query Syntax :TRIGger:GLITch:SOURce?

The :TRIGger:GLITch:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE".

Return Format <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands" on page 591](#)
 - [":TRIGger:MODE" on page 600](#)
 - [":TRIGger:GLITch:LEVel" on page 613](#)
 - [":TRIGger:GLITch:POLarity" on page 614](#)
 - [":TRIGger:GLITch:QUALifier" on page 615](#)
 - [":TRIGger:GLITch:RANGE" on page 616](#)

Example Code

- ["Example Code" on page 608](#)

:TRIGger:PATTERn Commands

Table 92 :TRIGger:PATTERn Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTERn <string>[,<edge_source>,<edge>] (see page 619)	:TRIGger:PATTERn? (see page 620)	<string> ::= "nn...n" where n ::= {0 1 X R F} when <base> = ASCII <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$} when <base> = HEX <edge_source> ::= {CHANnel<n> NONE} for DSO models <edge_source> ::= {CHANnel<n> DIGital<d> NONE} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive NEGative}
:TRIGger:PATTERn:FORM at <base> (see page 621)	:TRIGger:PATTERn:FORM at? (see page 621)	<base> ::= {ASCII HEX}
:TRIGger:PATTERn:QUALifier <qualifier> (see page 622)	:TRIGger:PATTERn:QUALifier? (see page 622)	<qualifier> ::= ENTERed

:TRIGger:PATTERn

C (see page 794)

Command Syntax

```
:TRIGger:PATTERn <pattern>
<pattern> ::= <string>[,<edge_source>,<edge>]
<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when
            <base> = ASCII
<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X | $} when
            <base> = HEX
<edge_source> ::= {CHANnel<n> | NONE} for DSO models
<edge_source> ::= {CHANnel<n> | DIGItal<d>
                  | NONE} for MSO models
<n> ::= 1 to (# of analog channels) in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTERn command specifies the channel values to be used in the pattern trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog + 8 digital channels (mixed-signal)	Bits 0 through 7 - digital channels 0 through 7. Bits 8 through 11 - analog channels 4 through 1.
2 analog + 8 digital channels (mixed-signal)	Bits 0 through 7 - digital channels 0 through 7. Bits 8 and 9 - analog channels 2 and 1.
4 analog channels only	Bits 0 through 3 - analog channels 4 through 1.
2 analog channels only	Bits 0 and 1 - analog channels 2 and 1.

The format of the <string> parameter depends on the :TRIGger:PATTERn:FORMAT command setting:

- When the format is ASCII, the string looks just like the string you see on the oscilloscope's front panel, made up of 0, 1, X (don't care), R (rising edge), and F (falling edge) characters.
- When the format is HEX, the string begins with "0x" and contains hex digit characters or X (don't care for all four bits in the nibble).

With the hex format string, you can use the <edge_source> and <edge> parameters to specify an edge on one of the channels.

NOTE

The optional <edge_source> and <edge> parameters should be sent together or not at all. The edge can be specified in the ASCII <string> parameter. If the edge source and edge parameters are used, they take precedence.

You can only specify an edge on one channel. When an edge is specified, the :TRIGger:PATTERn:QUALifier does not apply.

Query Syntax :TRIGger:PATTERn?

The :TRIGger:PATTERn? query returns the pattern string, edge source, and edge.

Return Format <string>,<edge_source>,<edge><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[":TRIGger:PATTERn:FORMAT](#)" on page 621
 - "[":TRIGger:PATTERn:QUALifier](#)" on page 622
 - "[":TRIGger:MODE](#)" on page 600

:TRIGger:PATTERn:FORMAT

N (see [page 794](#))

Command Syntax `:TRIGger:PATTERn:FORMAT <base>`
 `<base> ::= {ASCii | HEX}`

The :TRIGger:PATTERn:FORMAT command sets the entry (and query) number base used by the :TRIGger:PATTERn command. The default <base> is ASCii.

Query Syntax `:TRIGger:PATTERn:FORMAT?`

The :TRIGger:PATTERn:FORMAT? query returns the currently set number base for pattern trigger patterns.

Return Format `<base><NL>`
 `<base> ::= {ASC | HEX}`

See Also

- ["Introduction to :TRIGger Commands"](#) on page 591
- [":TRIGger:PATTERn"](#) on page 619

:TRIGger:PATTERn:QUALifier**N** (see [page 794](#))

Command Syntax `:TRIGger:PATTERn:QUALifier <qualifier>`
 `<qualifier> ::= ENTERed`

The :TRIGger:PATTERn:QUALifier command qualifies when the trigger occurs.

In the InfiniiVision 2000 X-Series oscilloscopes, the trigger always occurs when the pattern is entered.

Query Syntax `:TRIGger:PATTERn:QUALifier?`

The :TRIGger:PATTERn:QUALifier? query returns the trigger duration qualifier.

Return Format `<qualifier><NL>`

See Also • "Introduction to :TRIGger Commands" on page 591

:TRIGger:TV Commands

Table 93 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 624)	:TRIGger:TV:LINE? (see page 624)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 625)	:TRIGger:TV:MODE? (see page 625)	<tv mode> ::= {FIELD1 FIELD2 AFIELDS ALINES LFIELD1 LFIELD2 LATERNATE}
:TRIGger:TV:POLarity <polarity> (see page 626)	:TRIGger:TV:POLarity? (see page 626)	<polarity> ::= {POSITIVE NEGATIVE}
:TRIGger:TV:SOURce <source> (see page 627)	:TRIGger:TV:SOURce? (see page 627)	<source> ::= {CHANNEL<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANDARD <standard> (see page 628)	:TRIGger:TV:STANDARD? (see page 628)	<standard> ::= {NTSC PAL PALM SECAM}

:TRIGger:TV:LINE

N (see [page 794](#))

Command Syntax `:TRIGger:TV:LINE <line_number>`

`<line_number> ::= integer in NR1 format`

The :TRIGger:TV:LINE command allows triggering on a specific line of video. The line number limits vary with the standard and mode, as shown in the following table.

Table 94 TV Trigger Line Number Limits

TV Standard	Mode		
	LField1	LField2	LALternate
NTSC	1 to 263	1 to 262	1 to 262
PAL	1 to 313	314 to 625	1 to 312
PAL-M	1 to 263	264 to 525	1 to 262
SECAM	1 to 313	314 to 625	1 to 312

Query Syntax `:TRIGger:TV:LINE?`

The :TRIGger:TV:LINE? query returns the current TV trigger line number setting.

Return Format `<line_number><NL>`

`<line_number> ::= integer in NR1 format`

See Also

- "Introduction to :TRIGger Commands" on page 591
- ":TRIGger:TV:STANDARD" on page 628
- ":TRIGger:TV:MODE" on page 625

:TRIGger:TV:MODE

N (see [page 794](#))

Command Syntax :TRIGger:TV:MODE <mode>

```
<mode> ::= {FIEld1 | FIEld2 | AFIElds | ALINes
             | LFIeld1 | LFIeld2 | LALTernate}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIElds	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEALT

Query Syntax :TRIGger:TV:MODE?

The :TRIGger:TV:MODE? query returns the TV trigger mode.

Return Format <value><NL>

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LFI1 | LFI2 | LALT}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 591
 - [":TRIGger:TV:STANDARD"](#) on page 628
 - [":TRIGger:MODE"](#) on page 600

:TRIGger:TV:POLarity**N** (see [page 794](#))**Command Syntax** `:TRIGger:TV:POLarity <polarity>` `<polarity> ::= {POSITIVE | NEGATIVE}`

The :TRIGger:TV:POLarity command sets the polarity for the TV trigger.

Query Syntax `:TRIGger:TV:POLarity?`

The :TRIGger:TV:POLarity? query returns the TV trigger polarity.

Return Format `<polarity><NL>` `<polarity> ::= {POS | NEG}`**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 591

- "[:TRIGger:MODE](#)" on page 600

- "[:TRIGger:TV:SOURce](#)" on page 627

:TRIGger:TV:SOURce**N** (see [page 794](#))**Command Syntax** `:TRIGger:TV:SOURce <source>` `<source> ::= {CHANnel<n>}` `<n> ::= 1 to (# analog channels) in NR1 format`

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

Query Syntax `:TRIGger:TV:SOURce?`

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

Return Format `<source><NL>` `<source> ::= {CHAN<n>}`

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 591
 - "[:TRIGger:MODE](#)" on page 600
 - "[:TRIGger:TV:POLarity](#)" on page 626

Example Code

- "[Example Code](#)" on page 608

:TRIGger:TV:STANDARD**N** (see [page 794](#))**Command Syntax** `:TRIGger:TV:STANDARD <standard>``<standard> ::= {NTSC | PALM | PAL | SECAM}`

The :TRIGger:TV:STANDARD command selects the video standard:

- NTSC
- PAL
- PAL-M
- SECAM

Query Syntax `:TRIGger:TV:STANDARD?`

The :TRIGger:TV:STANDARD? query returns the current TV trigger standard setting.

Return Format `<standard><NL>``<standard> ::= {NTSC | PALM | PAL | SEC}`

30 :WAVeform Commands

Provide access to waveform data. See "[Introduction to :WAVeform Commands](#)" on page 631.

Table 95 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 637)	:WAVeform:BYTeorder? (see page 637)	<value> ::= {LSBFFirst MSBFFirst}
n/a	:WAVeform:COUNT? (see page 638)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 639)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVeform:FORMAT <value> (see page 641)	:WAVeform:FORMAT? (see page 641)	<value> ::= {WORD BYTE ASCII}
:WAVeform:POINTS <# points> (see page 642)	:WAVeform:POINTS? (see page 642)	<# points> ::= {100 250 500 1000 <points_mode>} if waveform points mode is NORMAl <# points> ::= {100 250 500 1000 2000 ... 8000000 in 1-2-5 sequence <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMAl MAXimum RAW}

Table 95 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:POINTs:MODE <points_mode> (see page 644)	:WAVEform:POINTs:MODE ? (see page 644)	<points_mode> ::= {NORMal MAXimum RAW}
n/a	:WAVEform:PREamble? (see page 646)	<p><preamble_block> ::= <format NR1>, <type NR1>, <points NR1>, <count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>, <yincrement NR3>, <yorigin NR3>, <yreference NR1></p> <p><format> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for BYTE format • 1 for WORD format • 2 for ASCII format <p><type> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for NORMAL type • 1 for PEAK detect type • 3 for AVERAGE type • 4 for HRESolution type <p><count> ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format</p>
n/a	:WAVEform:SEGmented:COUNT? (see page 649)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see page 650)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see page 651)	:WAVEform:SOURce? (see page 651)	<p><source> ::= {CHANnel<n> FUNCtion MATH SBUS1} for DSO models</p> <p><source> ::= {CHANnel<n> POD{1 2} BUS{1 2} FUNCtion MATH SBUS1} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p>
:WAVEform:SOURce:SUBSource <subsource> (see page 655)	:WAVEform:SOURce:SUBSource? (see page 655)	<subsource> ::= {{SUB0 RX MOSI} {SUB1 TX MISO}}
n/a	:WAVEform:TYPE? (see page 656)	<return_mode> ::= {NORM PEAK AVER HRES}

Table 95 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:UNSIGNED { {0 OFF} {1 ON} } (see page 657)	:WAVEform:UNSIGNED? (see page 657)	{0 1}
:WAVEform:VIEW <view> (see page 658)	:WAVEform:VIEW? (see page 658)	<view> ::= {MAIN}
n/a	:WAVEform:XINCREMENT? (see page 659)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVEform:XORIGIN? (see page 660)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVEform:XREFERENCE? (see page 661)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVEform:YINCREMENT? (see page 662)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVEform:YORIGIN? (see page 663)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVEform:YREFERENCE? (see page 664)	<return_value> ::= y-reference value in the current preamble in NR1 format

Introduction to :WAVEform Commands The WAVEform subsystem is used to transfer data to a controller from the oscilloscope waveform memories. The queries in this subsystem will only operate when the channel selected by :WAVEform:SOURce is on.

Waveform Data and Preamble

The waveform record is actually contained in two portions: the preamble and waveform data. The waveform record must be read from the oscilloscope by the controller using two separate commands, :WAVEform:DATA (see [page 639](#)) and :WAVEform:PREamble (see [page 646](#)). The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data, which includes the number of points acquired, the format of acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that word and byte data can be translated to time and voltage values.

Data Acquisition Types

There are four types of waveform acquisitions that can be selected for analog channels with the :ACQuire:TYPE command (see [page 193](#)): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always acquired using NORMal. When the data is acquired using the :DIGItize command (see [page 157](#)) or :RUN command (see [page 174](#)), the data is placed in the channel buffer of the specified source.

Once you have acquired data with the :DIGItize command, the instrument is stopped. If the instrument is restarted (via the programming interface or the front panel), or if any instrument setting is changed, the data acquired with the :DIGItize command may be overwritten. You should first acquire the data with the :DIGItize command, then immediately read the data with the :WAVEform:DATA? query (see [page 639](#)) before changing any instrument setup.

A waveform record consists of either all of the acquired points or a subset of the acquired points. The number of points acquired may be queried using :ACQuire:POINts? (see [page 186](#)).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVEform:POINts command (see [page 642](#)). If :WAVEform:POINts MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes or as many as 8,000,000 for a digital channel on the mixed signal oscilloscope. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVEform:POINts may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVEform:POINts must be an even divisor of 1,000 or be set to MAXimum. :WAVEform:POINts determines the increment between time buckets that will be transferred. If POINts = MAXimum, the data cannot be decimated. For example:

- `:WAVEform:POINTS 1000` – returns time buckets 0, 1, 2, 3, 4, ..., 999.
- `:WAVEform:POINTS 500` – returns time buckets 0, 2, 4, 6, 8, ..., 998.
- `:WAVEform:POINTS 250` – returns time buckets 0, 4, 8, 12, 16, ..., 996.
- `:WAVEform:POINTS 100` – returns time buckets 0, 10, 20, 30, 40, ..., 990.

Analog Channel Data

NORMal Data

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over the programming interface in a linear fashion starting with time bucket 0 and going through time bucket n - 1, where n is the number returned by the :WAVEform:POINts? query (see [page 642](#)). Only the magnitude values of each data point are transmitted. The first voltage value corresponds to the first time bucket on the left side of the screen and the last value corresponds to the

next-to-last time bucket on the right side of the screen. Time buckets without data return 0. The time values for each data point correspond to the position of the data point in the data array. These time values are not transmitted.

AVERage Data

AVERage data consists of the average of the first n hits in a time bucket, where n is the value returned by the :ACQuire:COUNt query (see [page 184](#)). Time buckets that have fewer than n hits return the average of the data they do have. If a time bucket does not have any data in it, it returns 0.

This data is transmitted over the interface linearly, starting with time bucket 0 and proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query (see [page 642](#)). The first value corresponds to a point at the left side of the screen and the last value corresponds to one point away from the right side of the screen. The maximum number of points that can be returned in average mode is 1000 unless ACQuire:COUNt has been set to 1.

PEAK Data

Peak detect display mode is used to detect glitches for time base settings of 500 us/div and slower. In this mode, the oscilloscope can sample more data than it can store and display. So, when peak detect is turned on, the oscilloscope scans through the extra data, picks up the minimum and maximum for each time bucket, then stores the data in an array. Each time bucket contains two data sample.

The array is transmitted over the interface bus linearly, starting with time bucket 0 proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query (see [page 642](#)). In each time bucket, two values are transmitted, first the minimum, followed by the maximum. The first pair of values corresponds to the time bucket at the leftmost side of the screen. The last pair of values corresponds to the time bucket at the far right side of the screen. In :ACQuire:TYPE PEAK mode (see [page 193](#)), the value returned by the :WAVeform:XINCrement query (see [page 659](#)) should be doubled to find the time difference between the min-max pairs.

HRESolution Data

The high resolution (*smoothing*) mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

Data Conversion

Word or byte data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble. Each channel has its own waveform preamble.

In converting a data value to a voltage value, the following formula is used:

$$\text{voltage} = [(\text{data value} - \text{yreference}) * \text{yincrement}] + \text{yorigin}$$

If the :WAVeform:FORMat data format is ASCii (see [page 641](#)), the data values are converted internally and sent as floating point values separated by commas.

In converting a data value to time, the time value of a data point can be determined by the position of the data point. For example, the fourth data point sent with :WAVeform:XORigin = 16 ns, :WAVeform:XREFerence = 0, and :WAVeform:XINCrement = 2 ns, can be calculated using the following formula:

$$\text{time} = [(\text{data point number} - \text{xreference}) * \text{xincrement}] + \text{xorigin}$$

This would result in the following calculation for time bucket 3:

$$\text{time} = [(3 - 0) * 2 \text{ ns}] + 16 \text{ ns} = 22 \text{ ns}$$

In :ACQuire:TYPE PEAK mode (see [page 193](#)), because data is acquired in max-min pairs, modify the previous time formula to the following:

$$\text{time} = [(\text{data pair number} - \text{xreference}) * \text{xincrement} * 2] + \text{xorigin}$$

Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCii (see "[:WAVeform:FORMat](#)" on page 641). BYTE, WORD and ASCii formatted waveform records are transmitted using the arbitrary block program data format specified in IEEE 488.2.

When you use the block data format, the ASCII character string "#8<DD...D>" is sent prior to sending the actual data. The 8 indicates how many Ds follow. The Ds are ASCII numbers that indicate how many data bytes follow.

For example, if 1000 points will be transferred, and the WORD format was specified, the block header "#800001000" would be sent. The 8 indicates that eight length bytes follow, and 00001000 indicates that 1000 binary data bytes follow.

Use the :WAVeform:UNSIGNED command (see [page 657](#)) to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

Data Format for Transfer - ASCii format

The ASCii format (see "[:WAVeform:FORMat](#)" on page 641) provides access to the waveform data as real Y-axis values without using Y origin, Y reference, and Y increment to convert the binary data. Values are transferred as ASCii digits in floating point format separated by commas. In ASCii format, holes are represented by the value 9.9e+37. The setting of :WAVeform:BYTeorder (see [page 637](#)) and :WAVeform:UNSIGNED (see [page 657](#)) have no effect when the format is ASCii.

Data Format for Transfer - WORD format

WORD format (see "[:WAVeform:FORMat](#)" on page 641) provides 16-bit access to the waveform data. In the WORD format, the number of data bytes is twice the number of data points. The number of data points is the value returned by the :WAVeform:POINts? query (see [page 642](#)). If the data intrinsically has less than 16 bits of resolution, the data is left-shifted to provide 16 bits of resolution and the least significant bits are set to 0. Currently, the greatest intrinsic resolution of any data is 12 bits, so at least the lowest 4 bits of data will be 0. If there is a hole in the data, the hole is represented by a 16 bit value equal to 0.

Use :WAVeform:BYTeorder (see [page 637](#)) to determine if the least significant byte or most significant byte is to be transferred first. The :BYTeorder command can be used to alter the transmit sequence to match the storage sequence of an integer in the programming language being used.

Data Format for Transfer - BYTE format

The BYTE format (see "[:WAVeform:FORMat](#)" on page 641) allows 8-bit access to the waveform data. If the data intrinsically has more than 8 bits of resolution (averaged data), the data is right-shifted (truncated) to fit into 8 bits. If there is a hole in the data, the hole is represented by a value of 0. The BYTE-formatted data transfers over the programming interface faster than ASCII or WORD-formatted data, because in ASCII format, as many as 13 bytes per point are transferred, in BYTE format one byte per point is transferred, and in WORD format two bytes per point are transferred.

The :WAVeform:BYTeorder command (see [page 637](#)) has no effect when the data format is BYTE.

Digital Channel Data (MSO models only)

The waveform record for digital channels is similar to that of analog channels. The main difference is that the data points represent either DIGital0,...,7 (POD1), DIGital8,...,15 (POD2), or any grouping of digital channels (BUS1 or BUS2).

For digital channels, :WAVeform:UNSIGNED (see [page 657](#)) must be set to ON.

Digital Channel POD Data Format

Data for digital channels is only available in groups of 8 bits (Pod1 = D0 - D7, Pod2 = D8 - D15). The bytes are organized as:

:WAVeform:SOURce	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
POD1	D7	D6	D5	D4	D3	D2	D1	D0
POD2	D15	D14	D13	D12	D11	D10	D9	D8

If the :WAVeform:FORMat is WORD (see [page 641](#)) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see [page 637](#)) controls which byte is 0.

If a digital channel is not displayed, its bit value in the pod data byte is not defined.

Digital Channel BUS Data Format

Digital channel BUS definitions can include any or all of the digital channels. Therefore, data is always returned as 16-bit values. :BUS commands (see [page 195](#)) are used to select the digital channels for a bus.

Reporting the Setup

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

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

:WAVeform:BYTeorder



(see [page 794](#))

Command Syntax `:WAVeform:BYTeorder <value>`
`<value> ::= {LSBFFirst | MSBFFirst}`

The :WAVeform:BYTeorder command sets the output sequence of the WORD data.

- MSBFFirst – sets the most significant byte to be transmitted first.
- LSBFirst – sets the least significant byte to be transmitted first.

This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected.

The default setting is MSBFFirst.

Query Syntax `:WAVeform:BYTeorder?`

The :WAVeform:BYTeorder query returns the current output sequence.

Return Format `<value><NL>`
`<value> ::= {LSBF | MSBF}`

See Also

- "[Introduction to :WAVeform Commands](#)" on page 631
- "[":WAVeform:DATA](#)" on page 639
- "[":WAVeform:FORMat](#)" on page 641
- "[":WAVeform:PREamble](#)" on page 646

Example Code

- "[Example Code](#)" on page 652
- "[Example Code](#)" on page 647

:WAVeform:COUNT(see [page 794](#))**Query Syntax** `:WAVeform:COUNT?`

The `:WAVeform:COUNT?` query returns the count used to acquire the current waveform. This may differ from current values if the unit has been stopped and its configuration modified. For all acquisition types except average, this value is 1.

Return Format `<count_argument><NL>`

`<count_argument>` ::= an integer from 1 to 65536 in NR1 format

See Also

- ["Introduction to :WAVeform Commands"](#) on page 631
- [":ACQuire:COUNT"](#) on page 184
- [":ACQuire:TYPE"](#) on page 193

:WAVeform:DATA



(see [page 794](#))

Query Syntax :WAVeform:DATA?

The :WAVeform:DATA query returns the binary block of sampled data points transmitted using the IEEE 488.2 arbitrary block data format. The binary data is formatted according to the settings of the :WAVeform:UNSIGNED, :WAVeform:BYTeorder, :WAVeform:FORMAT, and :WAVeform:SOURce commands. The number of points returned is controlled by the :WAVeform:POINTS command.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 – Hole. Holes are locations where data has not yet been acquired.
- Another situation where there can be zeros in the data, incorrectly, is when programming over telnet port 5024. Port 5024 provides a command prompt and is intended for ASCII transfers. Use telnet port 5025 instead.
- 0x01 or 0x0001 – Clipped low. These are locations where the waveform is clipped at the bottom of the oscilloscope display.
- 0xFF or 0xFFFF – Clipped high. These are locations where the waveform is clipped at the top of the oscilloscope display.

Return Format <binary block data><NL>

See Also

- For a more detailed description of the data returned for different acquisition types, see: "[Introduction to :WAVeform Commands](#)" on page 631
- "[:WAVeform:UNSIGNED](#)" on page 657
- "[:WAVeform:BYTeorder](#)" on page 637
- "[:WAVeform:FORMAT](#)" on page 641
- "[:WAVeform:POINTS](#)" on page 642
- "[:WAVeform:PREamble](#)" on page 646
- "[:WAVeform:SOURce](#)" on page 651
- "[:WAVeform:TYPE](#)" on page 656

Example Code

```

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'     <header><waveform_data><NL>
'
' Where:
'     <header> = #800001000 (This is an example header)

```

```

' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) -
    Step (UBound(varQueryResult) / 20) - ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 -
            + varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) -
            * sngYIncrement + sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) -
            * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:WAVeform:FORMAT



(see page 794)

Command Syntax

```
:WAVeform:FORMAT <value>
<value> ::= {WORD | BYTE | ASCii}
```

The :WAVeform:FORMAT command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.
- ASCII formatted data is transferred ASCii text.
- WORD formatted data transfers 16-bit data as two bytes. The :WAVeform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

When the :WAVeform:SOURce is the serial decode bus SBUS1, ASCii is the only waveform format allowed.

When the :WAVeform:SOURce is one of the digital channel buses (BUS1 or BUS2), ASCii and WORD are the only waveform formats allowed.

Query Syntax

```
:WAVeform:FORMAT?
```

The :WAVeform:FORMAT query returns the current output format for the transfer of waveform data.

Return Format

```
<value><NL>
<value> ::= {WORD | BYTE | ASC}
```

See Also

- "[Introduction to :WAVeform Commands](#)" on page 631
- "[":WAVeform:BYTeorder](#)" on page 637
- "[":WAVeform:SOURce](#)" on page 651
- "[":WAVeform:DATA](#)" on page 639
- "[":WAVeform:PREamble](#)" on page 646

Example Code

- "[Example Code](#)" on page 652

:WAVEform:POINTs

 (see [page 794](#))

Command Syntax

```
:WAVEform:POINTs <# points>

<# points> ::= {100 | 250 | 500 | 1000 | <points mode>}
                if waveform points mode is NORMAl

<# points> ::= {100 | 250 | 500 | 1000 | 2000 | 5000 | 10000 | 20000
                | 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
                | 4000000 | 8000000 | <points mode>}
                if waveform points mode is MAXimum or RAW

<points mode> ::= {NORMAl | MAXimum | RAW}
```

NOTE

The <points_mode> option is deprecated. Use the :WAVEform:POINTs:MODE command instead.

The :WAVEform:POINTs command sets the number of waveform points to be transferred with the :WAVEform:DATA? query. This value represents the points contained in the waveform selected with the :WAVEform:SOURce command.

For the analog or digital sources, the records that can be transferred depend on the waveform points mode. The maximum number of points returned for math (function) waveforms is determined by the NORMAl waveform points mode. See the :WAVEform:POINTs:MODE command ([see page 644](#)) for more information.

Only data visible on the display will be returned.

When the :WAVEform:SOURce is the serial decode bus SBUS1, this command is ignored, and all available serial decode bus data is returned.

Query Syntax

:WAVEform:POINTs?

The :WAVEform:POINTs query returns the number of waveform points to be transferred when using the :WAVEform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVEform:POINTs:MODE command ([see page 644](#)) for more information).

When the :WAVEform:SOURce is the serial decode bus SBUS1, this query returns the number of messages that were decoded.

Return Format

```
<# points><NL>

<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>}
                if waveform points mode is NORMAl

<# points> ::= {100 | 250 | 500 | 1000 | 2000 | 5000 | 10000 | 20000
                | 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
                | 4000000 | 8000000 | <maximum # points>}
                if waveform points mode is MAXimum or RAW
```

NOTE

If a full screen of data is not displayed, the number of points returned will not be 1000 or an even divisor of it.

See Also

- "[Introduction to :WAVeform Commands](#)" on page 631
- "[:ACQuire:POINts](#)" on page 186
- "[:WAVeform:DATA](#)" on page 639
- "[:WAVeform:SOURce](#)" on page 651
- "[:WAVeform:VIEW](#)" on page 658
- "[:WAVeform:PREamble](#)" on page 646
- "[:WAVeform:POINts:MODE](#)" on page 644

Example Code

```
' WAVE_POINTS - Specifies the number of points to be transferred  
' using the ":WAVEFORM:DATA?" query.  
myScope.WriteString ":WAVEFORM:POINTS 1000"
```

See complete example programs at: [Chapter 38](#), "Programming Examples," starting on page 803

:WAVeform:POINts:MODE



(see [page 794](#))

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>
<points_mode> ::= {NORMAl | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog or digital sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.
- The second is referred to as the *measurement record* and is a 62,500-point (maximum) representation of the raw acquisition record. The measurement record can be retrieved from any source.

If the <points_mode> is NORMAl the measurement record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command (see [page 178](#)) or the :DIGitize command (see [page 157](#)) in the root subsystem) in order to return more than the *measurement record*.
- :TIMEbase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMAl or HRESolution.
- MAXimum or RAW will allow up to 100,000 points to be returned. The number of points returned will vary as the instrument's configuration is changed. Use the :WAVeform:POINts? MAXimum query to determine the maximum number of points that can be retrieved at the current settings.

Query Syntax

```
:WAVeform:POINts:MODE?
```

The :WAVeform:POINts:MODE? query returns the current points mode. Setting the points mode will affect what data is transferred. See the discussion above.

Return Format

```
<points_mode><NL>
```

```
<points_mode> ::= {NORMal | MAXimum | RAW}
```

- See Also
- "[Introduction to :WAVeform Commands](#)" on page 631
 - "[":WAVeform:DATA](#)" on page 639
 - "[":ACQuire:POINts](#)" on page 186
 - "[":WAVeform:VIEW](#)" on page 658
 - "[":WAVeform:PREamble](#)" on page 646
 - "[":WAVeform:POINts](#)" on page 642
 - "[":TIMEbase:MODE](#)" on page 581
 - "[":ACQuire:TYPE](#)" on page 193
 - "[":ACQuire:COUNt](#)" on page 184

:WAVEform:PREamble



(see page 794)

Query Syntax `:WAVEform:PREamble?`

The `:WAVEform:PREamble` query requests the preamble information for the selected waveform source. The preamble data contains information concerning the vertical and horizontal scaling of the data of the corresponding channel.

Return Format `<preamble_block><NL>`

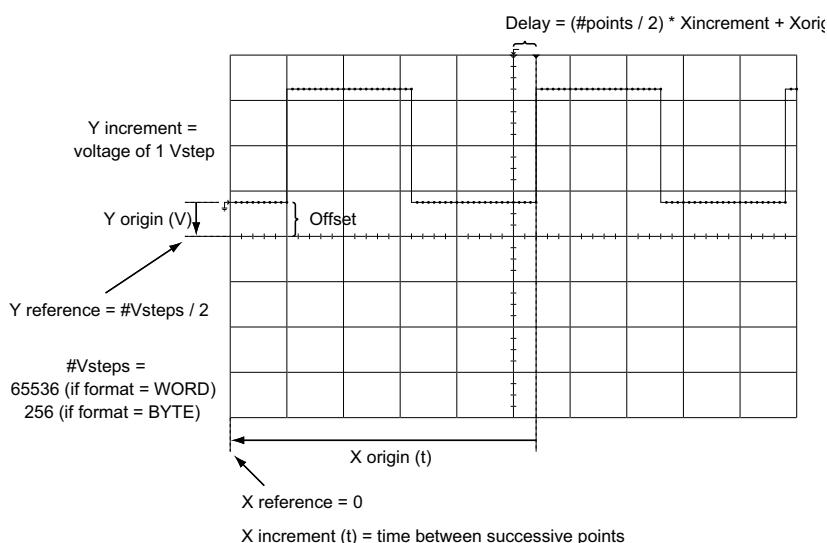
```

<preamble_block> ::= <format 16-bit NR1>,
                    <type 16-bit NR1>,
                    <points 32-bit NR1>,
                    <count 32-bit NR1>,
                    <xincrement 64-bit floating point NR3>,
                    <xorigin 64-bit floating point NR3>,
                    <xreference 32-bit NR1>,
                    <yincrement 32-bit floating point NR3>,
                    <yorigin 32-bit floating point NR3>,
                    <yreference 32-bit NR1>

<format> ::= 0 for BYTE format, 1 for WORD format, 4 for ASCII format;
            an integer in NR1 format (format set by :WAVEform:FORMAT).

<type> ::= 2 for AVERAGE type, 0 for NORMAL type, 1 for PEAK detect
            type; an integer in NR1 format (type set by :ACQuire:TYPE).

<count> ::= Average count or 1 if PEAK or NORMAL; an integer in NR1
            format (count set by :ACQuire:COUNT).
  
```



See Also

- "Introduction to :WAVEform Commands" on page 631
- ":ACQuire:COUNT" on page 184

- [":ACQuire:POINTs" on page 186](#)
- [":ACQuire:TYPE" on page 193](#)
- [":DIGitize" on page 157](#)
- [":WAVeform:COUNT" on page 638](#)
- [":WAVeform:DATA" on page 639](#)
- [":WAVeform:FORMAT" on page 641](#)
- [":WAVeform:POINTs" on page 642](#)
- [":WAVeform:TYPE" on page 656](#)
- [":WAVeform:XINCrement" on page 659](#)
- [":WAVeform:XORigin" on page 660](#)
- [":WAVeform:XREFERENCE" on page 661](#)
- [":WAVeform:YINCrement" on page 662](#)
- [":WAVeform:YORigin" on page 663](#)
- [":WAVeform:YREFERENCE" on page 664](#)

Example Code

```

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'   POINTS     : int32 - number of data points transferred.
'   COUNT       : int32 - 1 and is always 1.
'   XINCREMENT : float64 - time difference between data points.
'   XORIGIN    : float64 - always the first data point in memory.
'   XREFERENCE : int32 - specifies the data point associated with
'                      x-origin.
'   YINCREMENT : float32 - voltage diff between data points.
'   YORIGIN    : float32 - value is the voltage at center screen.
'   YREFERENCE : int32 - specifies the data point where y-origin
'                      occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"      ' Query for the preamble.
Preamble() = myScope.ReadList      ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)

```

```
    lngCount = Preamble(3)
    dblXIncrement = Preamble(4)
    dblXOrigin = Preamble(5)
    lngXReference = Preamble(6)
    sngYIncrement = Preamble(7)
    sngYOrigin = Preamble(8)
    lngYReference = Preamble(9)
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:WAVeform:SEGmented:COUNt

N (see [page 794](#))

Query Syntax :WAVeform:SEGmented:COUNt?

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

The :WAVeform:SEGmented:COUNt query returns the number of memory segments in the acquired data. You can use the :WAVeform:SEGmented:COUNt? query while segments are being acquired (although :DIGitize blocks subsequent queries until the full segmented acquisition is complete).

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGmented:COUNt command, and data is acquired using the :DIGITIZE, :SINGle, or :RUN commands.

Return Format <count> ::= an integer from 2 to 1000 in NR1 format (count set by :ACQuire:SEGmented:COUNt).

- See Also**
- [":ACQuire:MODE"](#) on page 185
 - [":ACQuire:SEGmented:COUNt"](#) on page 188
 - [":DIGITIZE"](#) on page 157
 - [":SINGle"](#) on page 176
 - [":RUN"](#) on page 174
 - ["Introduction to :WAVeform Commands"](#) on page 631

Example Code

- ["Example Code"](#) on page 189

:WAVeform:SEGmented:TTAG**N** (see [page 794](#))**Query Syntax** `:WAVeform:SEGmented:TTAG?`**NOTE** This command is available when the segmented memory option (Option SGM) is enabled.

The `:WAVeform:SEGmented:TTAG?` query returns the time tag of the currently selected segmented memory index. The index is selected using the `:ACQuire:SEGmented:INDex` command.

Return Format `<time_tag> ::= in NR3 format`**See Also**

- [":ACQuire:SEGmented:INDex"](#) on page 189
- ["Introduction to :WAVeform Commands"](#) on page 631

Example Code

- ["Example Code"](#) on page 189

:WAVeform:SOURce



(see [page 794](#))

Command Syntax

```
:WAVeform:SOURce <source>
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r> | SBUS1}
for DSO models

<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCTION
| MATH | WMEMORY<r> | SBUS1}
for MSO models

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

<r> ::= {1 | 2}
```

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, reference waveform, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS1), ASCii is the only waveform format allowed, and the :WAVeform:DATA? query returns a string with timestamps and associated bus decode information.

With MSO oscilloscope models, you can choose a POD or BUS as the waveform source. There are some differences between POD and BUS when formatting and getting data from the oscilloscope:

- When POD1 or POD2 is selected as the waveform source, you can choose the BYTE, WORD, or ASCii formats (see "[:WAVeform:FORMAT](#)" on page 641).

When the WORD format is chosen, every other data byte will be 0. The setting of :WAVeform:BYTeorder controls which byte is 0.

When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with unsigned decimal values separated by commas.

- When BUS1 or BUS2 is selected as the waveform source, you can choose the WORD or ASCii formats (but not BYTE because bus values are always returned as 16-bit values).

When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with hexadecimal bus values, for example: 0x1938,0xff38,...

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE

MATH is an alias for FUNCTION. The :WAVeform:SOURce? Query returns FUNC if the source is FUNCTION or MATH.

Return Format	<pre><source><NL> <source> ::= {CHAN<n> FUNC WMEM<r> SBUS1} for DSO models <source> ::= {CHAN<n> POD{1 2} BUS{1 2} FUNC WMEM<r> SBUS1} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= {1 2}</pre>
See Also	<ul style="list-style-type: none"> · "Introduction to :WAVeform Commands" on page 631 · ":DIGitize" on page 157 · ":WAVeform:FORMat" on page 641 · ":WAVeform:BYTeorder" on page 637 · ":WAVeform:DATA" on page 639 · ":WAVeform:PREamble" on page 646
Example Code	<pre>' WAVEFORM_DATA - To obtain waveform data, you must specify the ' WAVEFORM parameters for the waveform data prior to sending the ' ":WAVEFORM:DATA?" query. Once these parameters have been sent, ' the waveform data and the preamble can be read. ' ' WAVE_SOURCE - Selects the channel to be used as the source for ' the waveform commands. myScope.WriteString ":WAVEFORM:SOURCE CHAN1" ' WAVE_POINTS - Specifies the number of points to be transferred ' using the ":WAVEFORM:DATA?" query. myScope.WriteString ":WAVEFORM:POINTS 1000" ' WAVE_FORMAT - Sets the data transmission mode for the waveform ' data output. This command controls whether data is formatted in ' a word or byte format when sent from the oscilloscope. Dim lngVSteps As Long Dim intBytesPerData As Integer ' Data in range 0 to 65535. myScope.WriteString ":WAVEFORM:FORMAT WORD" lngVSteps = 65536 intBytesPerData = 2 ' Data in range 0 to 255. 'myScope.WriteString ":WAVEFORM:FORMAT BYTE" 'lngVSteps = 256 'intBytesPerData = 1 ' GET_PREAMBLE - The preamble block contains all of the current ' WAVEFORM settings. It is returned in the form <preamble_block><NL> ' where <preamble_block> is: ' FORMAT : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII. ' TYPE : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE ' POINTS : int32 - number of data points transferred. ' COUNT : int32 - 1 and is always 1. ' XINCREMENT : float64 - time difference between data points.</pre>

```

' XORIGIN      : float64 - always the first data point in memory.
' XREFERENCE   : int32 - specifies the data point associated with
'                   x-origin.
' YINCREMENT   : float32 - voltage diff between data points.
' YORIGIN      : float32 - value is the voltage at center screen.
' YREFERENCE   : int32 - specifies the data point where y-origin
'                   occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
strOutput = strOutput + "X increment = " +
'           FormatNumber(dblXIncrement * 1000000) + " us" + vbCrLf
strOutput = strOutput + "X origin = " +
'           FormatNumber(dblXOrigin * 1000000) + " us" + vbCrLf
strOutput = strOutput + "X reference = " +
'           CStr(lngXReference) + vbCrLf
strOutput = strOutput + "Y increment = " +
'           FormatNumber(sngYIncrement * 1000) + " mV" + vbCrLf
strOutput = strOutput + "Y origin = " +
'           FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Y reference = " +
'           CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " +
'           FormatNumber(lngVSteps * sngYIncrement / 8) + " V" + vbCrLf
strOutput = strOutput + "Offset = " +
'           FormatNumber((lngVSteps / 2 - lngYReference) * sngYIncrement + sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " +
'           FormatNumber(lngPoints * dblXIncrement / 10 * _

```

```

        1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " +
    FormatNumber(((lngPoints / 2 - lngXReference) * _
        dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
' <header><waveform_data><NL>
'
' Where:
'     <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

For lngI = 0 To UBound(varQueryResult) -
    Step (UBound(varQueryResult) / 20) - ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 -
            + varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) -
            * sngYIncrement + sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) -
            * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:WAVeform:SOURce:SUBSource



(see [page 794](#))

Command Syntax	<code>:WAVeform:SOURce:SUBSource <subsource></code>
	<code><subsource> ::= {{SUB0 RX MOSI} {SUB1 TX MISO}}</code>
	If the :WAVeform:SOURce is SBUS<n> (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.
	When using UART serial decode, this option lets you get "TX" data. (TX is an alias for SUB1.) The default, SUB0, specifies "RX" data. (RX is an alias for SUB0.)
	When using SPI serial decode, this option lets you get "MISO" data. (MISO is an alias for SUB1.) The default, SUB0, specifies "MOSI" data. (MOSI is an alias for SUB0.)
	If the :WAVeform:SOURce is not SBUS, or the :SBUS<n>:MODE is not UART or SPI, the only valid subsource is SUB0.
Query Syntax	<code>:WAVeform:SOURce:SUBSource?</code>
	The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.
Return Format	<code><subsource><NL></code>
	<code><subsource> ::= {SUB0 SUB1}</code>
See Also	<ul style="list-style-type: none"> • "Introduction to :WAVeform Commands" on page 631 • ":WAVeform:SOURce" on page 651

:WAVeform:TYPE(see [page 794](#))**Query Syntax** `:WAVeform:TYPE?`

The `:WAVeform:TYPE?` query returns the acquisition mode associated with the currently selected waveform. The acquisition mode is set by the `:ACQuire:TYPE` command.

Return Format `<mode><NL>`

`<mode> ::= {NORM | PEAK | AVER | HRES}`

NOTE

If the `:WAVeform:SOURce` is POD1 or POD2, the type is always NORM.

See Also

- "[Introduction to :WAVeform Commands](#)" on page 631
- "[":ACQuire:TYPE](#)" on page 193
- "[":WAVeform:DATA](#)" on page 639
- "[":WAVeform:PREamble](#)" on page 646
- "[":WAVeform:SOURce](#)" on page 651

:WAVeform:UNSIGNED



(see [page 794](#))

Command Syntax `:WAVeform:UNSIGNED <unsigned>`
`<unsigned> ::= {{0 | OFF} | {1 | ON}}`

The :WAVeform:UNSIGNED command turns unsigned mode on or off for the currently selected waveform. Use the WAVeform:UNSIGNED command to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCII.

If :WAVeform:SOURce is set to POD1, POD2, BUS1, or BUS2, WAVeform:UNSIGNED must be set to ON.

Query Syntax `:WAVeform:UNSIGNED?`

The :WAVeform:UNSIGNED? query returns the status of unsigned mode for the currently selected waveform.

Return Format `<unsigned><NL>`
`<unsigned> ::= {0 | 1}`

See Also

- "Introduction to :WAVeform Commands" on page 631
- "[:WAVeform:SOURce](#)" on page 651

:WAveform:VIEW(see [page 794](#))

Command Syntax `:WAveform:VIEW <view>`
 `<view> ::= {MAIN}`

The :WAveform:VIEW command sets the view setting associated with the currently selected waveform. Currently, the only legal value for the view setting is MAIN.

Query Syntax `:WAveform:VIEW?`

The :WAveform:VIEW? query returns the view setting associated with the currently selected waveform.

Return Format `<view><NL>`
 `<view> ::= {MAIN}`

See Also

- "Introduction to :WAveform Commands" on page 631
- "[:WAveform:POINts](#)" on page 642

:WAVeform:XINCrement(see [page 794](#))**Query Syntax** `:WAVeform:XINCrement?`

The `:WAVeform:XINCrement?` query returns the x-increment value for the currently specified source. This value is the time difference between consecutive data points in seconds.

Return Format `<value><NL>`

`<value>` ::= x-increment in the current preamble in 64-bit floating point NR3 format

See Also

- "Introduction to [:WAVeform Commands](#)" on page 631
- "[:WAVeform:PREamble](#)" on page 646

Example Code

- "[Example Code](#)" on page 647

:WAVeform:XORigin(see [page 794](#))**Query Syntax** `:WAVeform:XORigin?`

The `:WAVeform:XORigin?` query returns the x-origin value for the currently specified source. XORigin is the X-axis value of the data point specified by the `:WAVeform:XREFerence` value. In this product, that is always the X-axis value of the first data point (`XREFerence = 0`).

Return Format `<value><NL>`

`<value>` ::= x-origin value in the current preamble in 64-bit floating point NR3 format

See Also

- "Introduction to [:WAVeform Commands](#)" on page 631
- "[:WAVeform:PREamble](#)" on page 646
- "[:WAVeform:XREFerence](#)" on page 661

Example Code

- "[Example Code](#)" on page 647

:WAVeform:XREFerence(see [page 794](#))**Query Syntax** `:WAVeform:XREFerence?`

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

Return Format `<value><NL>`

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

See Also

- "Introduction to [:WAVeform Commands](#)" on page 631
- "[:WAVeform:PREamble](#)" on page 646
- "[:WAVeform:XORigin](#)" on page 660

Example Code

- "[Example Code](#)" on page 647

:WAVeform:YINCrement(see [page 794](#))**Query Syntax** `:WAVeform:YINCrement?`

The `:WAVeform:YINCrement?` query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values. The y-increment for digital waveforms is always "1".

Return Format `<value><NL>`

`<value>` ::= y-increment value in the current preamble in 32-bit floating point NR3 format

See Also

- ["Introduction to :WAVeform Commands" on page 631](#)
- [":WAVeform:PREamble" on page 646](#)

Example Code

- ["Example Code" on page 647](#)

:WAVeform:YORigin(see [page 794](#))**Query Syntax** `:WAVeform:YORigin?`

The `:WAVeform:YORigin?` query returns the y-origin value for the currently specified source. This value is the Y-axis value of the data value specified by the `:WAVeform:YREFerence` value. For this product, this is the Y-axis value of the center of the screen.

Return Format `<value><NL>`

`<value>` ::= y-origin in the current preamble in 32-bit floating point NR3 format

- See Also**
- "Introduction to [:WAVeform Commands](#)" on page 631
 - "[:WAVeform:PREamble](#)" on page 646
 - "[:WAVeform:YREFerence](#)" on page 664

Example Code

- "[Example Code](#)" on page 647

:WAVeform:YREFerence(see [page 794](#))**Query Syntax** `:WAVeform:YREFerence?`

The `:WAVeform:YREFerence?` query returns the y-reference value for the currently specified source. This value specifies the data point value where the y-origin occurs. In this product, this is the data point value of the center of the screen. It is undefined if the format is ASCII.

Return Format `<value><NL>`

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

See Also

- "Introduction to [:WAVeform Commands](#)" on page 631
- "[:WAVeform:PREamble](#)" on page 646
- "[:WAVeform:YORigin](#)" on page 663

Example Code

- "[Example Code](#)" on page 647

31 :WGEN Commands

When the built-in waveform generator is licensed (Option WGN), you can use it to output sine, square, ramp, pulse, DC, and noise waveforms. The :WGEN commands are used to select the waveform function and parameters. See ["Introduction to :WGEN Commands" on page 667](#).

Table 96 :WGEN Commands Summary

Command	Query	Options and Query Returns
:WGEN:FREQuency <frequency> (see page 668)	:WGEN:FREQuency? (see page 668)	<frequency> ::= frequency in Hz in NR3 format
:WGEN:FUNCTION <signal> (see page 669)	:WGEN:FUNCTION? (see page 670)	<signal> ::= {SINusoid SQuare RAMP PULSe NOISE DC}
:WGEN:FUNCTION:PULSe:WIDTh <width> (see page 671)	:WGEN:FUNCTION:PULSe:WIDTh? (see page 671)	<width> ::= pulse width in seconds in NR3 format
:WGEN:FUNCTION:RAMP:SYMmetry <percent> (see page 672)	:WGEN:FUNCTION:RAMP:SYMmetry? (see page 672)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format
:WGEN:FUNCTION:SQUare:DCYCLE <percent> (see page 673)	:WGEN:FUNCTION:SQUare:DCYCLE? (see page 673)	<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
:WGEN:MODulation:AM:DDEPTH <percent> (see page 674)	:WGEN:MODulation:AM:DDEPTH? (see page 674)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format
:WGEN:MODulation:AM:FREQuency <frequency> (see page 675)	:WGEN:MODulation:AM:FREQuency? (see page 675)	<frequency> ::= modulating waveform frequency in Hz in NR3 format
:WGEN:MODulation:FM:DEViation <frequency> (see page 676)	:WGEN:MODulation:FM:DEViation? (see page 676)	<frequency> ::= frequency deviation in Hz in NR3 format

Table 96 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:MODulation:FM:FREQuency <frequency> (see page 677)	:WGEN:MODulation:FM:FREQuency? (see page 677)	<frequency> ::= modulating waveform frequency in Hz in NR3 format
:WGEN:MODulation:FSKey:FREQuency <percent> (see page 678)	:WGEN:MODulation:FSKey:FREQuency? (see page 678)	<frequency> ::= hop frequency in Hz in NR3 format
:WGEN:MODulation:FSKey:RATE <rate> (see page 679)	:WGEN:MODulation:FSKey:RATE? (see page 679)	<rate> ::= FSK modulation rate in Hz in NR3 format
:WGEN:MODulation:FUNCTION <shape> (see page 680)	:WGEN:MODulation:FUNCTION? (see page 680)	<shape> ::= {SINusoid SQuare RAMP}
:WGEN:MODulation:FUNCTION:RAMP:SYMMetry <percent> (see page 681)	:WGEN:MODulation:FUNCTION:RAMP:SYMMetry? (see page 681)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format
:WGEN:MODulation:NOISE <percent> (see page 682)	:WGEN:MODulation:NOISE? (see page 682)	<percent> ::= 0 to 100
:WGEN:MODulation:STATE {{0 OFF} {1 ON}} (see page 683)	:WGEN:MODulation:STATE? (see page 683)	{0 1}
:WGEN:MODulation:TYPE <type> (see page 684)	:WGEN:MODulation:TYPE? (see page 684)	<type> ::= {AM FM FSK}
:WGEN:OUTPut {{0 OFF} {1 ON}} (see page 686)	:WGEN:OUTPut? (see page 686)	{0 1}
:WGEN:OUTPut:LOAD <impedance> (see page 687)	:WGEN:OUTPut:LOAD? (see page 687)	<impedance> ::= {ONEMeg FIFTy}
:WGEN:PERiod <period> (see page 688)	:WGEN:PERiod? (see page 688)	<period> ::= period in seconds in NR3 format
:WGEN:RST (see page 689)	n/a	n/a
:WGEN:VOLTage <amplitude> (see page 690)	:WGEN:VOLTage? (see page 690)	<amplitude> ::= amplitude in volts in NR3 format
:WGEN:VOLTage:HIGH <high> (see page 691)	:WGEN:VOLTage:HIGH? (see page 691)	<high> ::= high-level voltage in volts, in NR3 format

Table 96 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:VOLTage:LOW <low> (see page 692)	:WGEN:VOLTage:LOW? (see page 692)	<low> ::= low-level voltage in volts, in NR3 format
:WGEN:VOLTage:OFFSet <offset> (see page 693)	:WGEN:VOLTage:OFFSet? (see page 693)	<offset> ::= offset in volts in NR3 format

Introduction to :WGEN Commands The :WGEN subsystem provides commands to select the waveform generator function and parameters.

Reporting the Setup

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

Return Format

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

```
:WGEN:FUNC SIN;OUTP 0;FREQ +1.0000E+03;VOLT +500.0E-03;VOLT:OFFS
+0.0E+00;:WGEN:OUTP:LOAD ONEM
```

:WGEN:FREQuency

N (see [page 794](#))

Command Syntax `:WGEN:FREQuency <frequency>`

`<frequency> ::= frequency in Hz in NR3 format`

For all waveforms except Noise and DC, the :WGEN:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN:PERiod command.

Query Syntax `:WGEN:FREQuency?`

The :WGEN:FREQuency? query returns the currently set waveform generator frequency.

Return Format `<frequency><NL>`

`<frequency> ::= frequency in Hz in NR3 format`

See Also

- "[Introduction to :WGEN Commands](#)" on page 667

- "[":WGEN:FUNCTION](#)" on page 669

- "[":WGEN:PERiod](#)" on page 688

:WGEN:FUNCTION

N (see [page 794](#))

Command Syntax :WGEN:FUNCTION <signal>

<signal> ::= {SINusoid | SQuare | RAMP | PULSe | NOISE | DC}

The :WGEN:FUNCTION command selects the type of waveform:

Waveform Type	Characteristics
SINusoid	<p>Use these commands to set the sine signal parameters:</p> <ul style="list-style-type: none"> ▪ ":WGEN:FREQuency" on page 668 ▪ ":WGEN:PERiod" on page 688 ▪ ":WGEN:VOLTage" on page 690 ▪ ":WGEN:VOLTage:OFFSET" on page 693 ▪ ":WGEN:VOLTage:HIGH" on page 691 ▪ ":WGEN:VOLTage:LOW" on page 692 <p>The frequency can be adjusted from 100 mHz to 20 MHz.</p>
SQuare	<p>Use these commands to set the square wave signal parameters:</p> <ul style="list-style-type: none"> ▪ ":WGEN:FREQuency" on page 668 ▪ ":WGEN:PERiod" on page 688 ▪ ":WGEN:VOLTage" on page 690 ▪ ":WGEN:VOLTage:OFFSET" on page 693 ▪ ":WGEN:VOLTage:HIGH" on page 691 ▪ ":WGEN:VOLTage:LOW" on page 692 ▪ ":WGEN:FUNCTION:SQuare:DCYcle" on page 673 <p>The frequency can be adjusted from 100 mHz to 10 MHz. The duty cycle can be adjusted from 20% to 80%.</p>
RAMP	<p>Use these commands to set the ramp signal parameters:</p> <ul style="list-style-type: none"> ▪ ":WGEN:FREQuency" on page 668 ▪ ":WGEN:PERiod" on page 688 ▪ ":WGEN:VOLTage" on page 690 ▪ ":WGEN:VOLTage:OFFSET" on page 693 ▪ ":WGEN:VOLTage:HIGH" on page 691 ▪ ":WGEN:VOLTage:LOW" on page 692 ▪ ":WGEN:FUNCTION:RAMP:SYMMetry" on page 672 <p>The frequency can be adjusted from 100 mHz to 100 kHz. Symmetry represents the amount of time per cycle that the ramp waveform is rising and can be adjusted from 0% to 100%.</p>

Waveform Type	Characteristics
PULSe	<p>Use these commands to set the pulse signal parameters:</p> <ul style="list-style-type: none"> ▪ ":WGEN:FREQuency" on page 668 ▪ ":WGEN:PERiod" on page 688 ▪ ":WGEN:VOLTage" on page 690 ▪ ":WGEN:VOLTage:OFFSet" on page 693 ▪ ":WGEN:VOLTage:HIGH" on page 691 ▪ ":WGEN:VOLTage:LOW" on page 692 ▪ ":WGEN:FUNCTION:PULSe:WIDTH" on page 671 <p>The frequency can be adjusted from 100 mHz to 10 MHz. The pulse width can be adjusted from 20 ns to the period minus 20 ns.</p>
DC	<p>Use this command to set the DC level:</p> <ul style="list-style-type: none"> ▪ ":WGEN:VOLTage:OFFSet" on page 693
NOISe	<p>Use these commands to set the noise signal parameters:</p> <ul style="list-style-type: none"> ▪ ":WGEN:VOLTage" on page 690 ▪ ":WGEN:VOLTage:OFFSet" on page 693 ▪ ":WGEN:VOLTage:HIGH" on page 691 ▪ ":WGEN:VOLTage:LOW" on page 692

For all waveform types, the output amplitude, into 50 Ω , can be adjusted from 10 mVpp to 2.5 Vpp (or from 20 mVpp to 5 Vpp into an open-circuit load).

Query Syntax `:WGEN:FUNCTION?`

The `:WGEN:FUNCTION?` query returns the currently selected signal type.

Return Format `<signal><NL>`
`<signal> ::= {SIN | SQU | RAMP | PULS | NOIS | DC}`

See Also • ["Introduction to :WGEN Commands" on page 667](#)

:WGEN:FUNCTION:PULSe:WIDTH

N (see [page 794](#))

Command Syntax :WGEN:FUNCTION:PULSe:WIDTH <width>

<width> ::= pulse width in seconds in NR3 format

For Pulse waveforms, the :WGEN:FUNCTION:PULSe:WIDTH command specifies the width of the pulse.

The pulse width can be adjusted from 20 ns to the period minus 20 ns.

Query Syntax :WGEN:FUNCTION:PULSe:WIDTH?

The :WGEN:FUNCTION:PULSe:WIDTH? query returns the currently set pulse width.

Return Format <width><NL>

<width> ::= pulse width in seconds in NR3 format

See Also • ["Introduction to :WGEN Commands"](#) on page 667

• [":WGEN:FUNCTION"](#) on page 669

:WGEN:FUNCTION:RAMP:SYMMetry

N (see [page 794](#))

Command Syntax	<code>:WGEN:FUNCTION:RAMP:SYMMetry <percent></code> <code><percent> ::= symmetry percentage from 0% to 100% in NR1 format</code>
	For Ramp waveforms, the :WGEN:FUNCTION:RAMP:SYMMetry command specifies the symmetry of the waveform.
	Symmetry represents the amount of time per cycle that the ramp waveform is rising.
Query Syntax	<code>:WGEN:FUNCTION:RAMP:SYMMetry?</code>
	The :WGEN:FUNCTION:RAMP:SYMMetry? query returns the currently set ramp symmetry.
Return Format	<code><percent><NL></code> <code><percent> ::= symmetry percentage from 0% to 100% in NR1 format</code>
See Also	<ul style="list-style-type: none">"Introduction to :WGEN Commands" on page 667":WGEN:FUNCTION" on page 669

:WGEN:FUNCTION:SQUare:DCYCLE

N (see [page 794](#))

Command Syntax :WGEN:FUNCTION:SQUare:DCYCLE <percent>
<percent> ::= duty cycle percentage from 20% to 80% in NR1 format

For Square waveforms, the :WGEN:FUNCTION:SQUare:DCYCLE command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

Query Syntax :WGEN:FUNCTION:SQUare:DCYCLE?

The :WGEN:FUNCTION:SQUare:DCYCLE? query returns the currently set square wave duty cycle.

Return Format <percent><NL>
<percent> ::= duty cycle percentage from 20% to 80% in NR1 format

See Also • ["Introduction to :WGEN Commands"](#) on page 667
• [":WGEN:FUNCTION"](#) on page 669

:WGEN:MODulation:AM:DEPTH

N (see [page 794](#))

Command Syntax	<code>:WGEN:MODulation:AM:DEPTH <percent></code> <code><percent> ::= AM depth percentage from 0% to 100% in NR1 format</code>
	The :WGEN:MODulation:AM:DEPTH command specifies the amount of amplitude modulation.
	AM Depth refers to the portion of the amplitude range that will be used by the modulation. For example, a depth setting of 80% causes the output amplitude to vary from 10% to 90% ($90\% - 10\% = 80\%$) of the original amplitude as the modulating signal goes from its minimum to maximum amplitude.
Query Syntax	<code>:WGEN:MODulation:AM:DEPTH?</code>
	The :WGEN:MODulation:AM:DEPTH? query returns the AM depth percentage setting.
Return Format	<code><percent><NL></code> <code><percent> ::= AM depth percentage from 0% to 100% in NR1 format</code>
See Also	<ul style="list-style-type: none"> · ":WGEN:MODulation:AM:FREQuency" on page 675 · ":WGEN:MODulation:FM:DEViation" on page 676 · ":WGEN:MODulation:FM:FREQuency" on page 677 · ":WGEN:MODulation:FSKey:FREQuency" on page 678 · ":WGEN:MODulation:FSKey:RATE" on page 679 · ":WGEN:MODulation:FUNCTION" on page 680 · ":WGEN:MODulation:FUNCTION:RAMP:SYMMetry" on page 681 · ":WGEN:MODulation:STATE" on page 683 · ":WGEN:MODulation:TYPE" on page 684

:WGEN:MODulation:AM:FREQuency

N (see [page 794](#))

Command Syntax	<code>:WGEN:MODulation:AM:FREQuency <frequency></code> <code><frequency> ::= modulating waveform frequency in Hz in NR3 format</code>
	The :WGEN:MODulation:AM:FREQuency command specifies the frequency of the modulating signal.
Query Syntax	<code>:WGEN:MODulation:AM:FREQuency?</code>
	The :WGEN:MODulation:AM:FREQuency? query returns the frequency of the modulating signal.
Return Format	<code><frequency><NL></code> <code><frequency> ::= modulating waveform frequency in Hz in NR3 format</code>
See Also	<ul style="list-style-type: none"> · ":WGEN:MODulation:AM:DEPTH" on page 674 · ":WGEN:MODulation:FM:DEViation" on page 676 · ":WGEN:MODulation:FM:FREQuency" on page 677 · ":WGEN:MODulation:FSKey:FREQuency" on page 678 · ":WGEN:MODulation:FSKey:RATE" on page 679 · ":WGEN:MODulation:FUNCTION" on page 680 · ":WGEN:MODulation:FUNCTION:RAMP:SYMMetry" on page 681 · ":WGEN:MODulation:STATE" on page 683 · ":WGEN:MODulation:TYPE" on page 684

:WGEN:MODulation:FM:DEViation

N (see [page 794](#))

Command Syntax `:WGEN:MODulation:FM:DEViation <frequency>`

`<frequency> ::= frequency deviation in Hz in NR3 format`

The :WGEN:MODulation:FM:DEViation command specifies the frequency deviation from the original carrier signal frequency.

When the modulating signal is at its maximum amplitude, the output frequency is the carrier signal frequency plus the deviation amount, and when the modulating signal is at its minimum amplitude, the output frequency is the carrier signal frequency minus the deviation amount.

The frequency deviation cannot be greater than the original carrier signal frequency.

Also, the sum of the original carrier signal frequency and the frequency deviation must be less than or equal to the maximum frequency for the selected waveform generator function plus 100 kHz.

Query Syntax `:WGEN:MODulation:FM:DEViation?`

The :WGEN:MODulation:FM:DEViation? query returns the frequency deviation setting.

Return Format `<frequency><NL>`

`<frequency> ::= frequency deviation in Hz in NR3 format`

See Also

- "[:WGEN:MODulation:AM:DEPTH](#)" on page 674
- "[:WGEN:MODulation:AM:FREQuency](#)" on page 675
- "[:WGEN:MODulation:FM:FREQuency](#)" on page 677
- "[:WGEN:MODulation:FSKey:FREQuency](#)" on page 678
- "[:WGEN:MODulation:FSKey:RATE](#)" on page 679
- "[:WGEN:MODulation:FUNCTION](#)" on page 680
- "[:WGEN:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 681
- "[:WGEN:MODulation:STATe](#)" on page 683
- "[:WGEN:MODulation:TYPE](#)" on page 684

:WGEN:MODulation:FM:FREQuency

N (see [page 794](#))

Command Syntax	<code>:WGEN:MODulation:FM:FREQuency <frequency></code> <code><frequency> ::= modulating waveform frequency in Hz in NR3 format</code>
	The :WGEN:MODulation:FM:FREQuency command specifies the frequency of the modulating signal.
Query Syntax	<code>:WGEN:MODulation:FM:FREQuency?</code>
	The :WGEN:MODulation:FM:FREQuency? query returns the frequency of the modulating signal.
Return Format	<code><frequency><NL></code> <code><frequency> ::= modulating waveform frequency in Hz in NR3 format</code>
See Also	<ul style="list-style-type: none"> · ":WGEN:MODulation:AM:DEPTH" on page 674 · ":WGEN:MODulation:AM:FREQuency" on page 675 · ":WGEN:MODulation:FM:DEViation" on page 676 · ":WGEN:MODulation:FSKey:FREQuency" on page 678 · ":WGEN:MODulation:FSKey:RATE" on page 679 · ":WGEN:MODulation:FUNCTION" on page 680 · ":WGEN:MODulation:FUNCTION:RAMP:SYMMetry" on page 681 · ":WGEN:MODulation:STATE" on page 683 · ":WGEN:MODulation:TYPE" on page 684

:WGEN:MODulation:FSKey:FREQuency

N (see [page 794](#))

Command Syntax	<code>:WGEN:MODulation:FSKey:FREQuency <frequency></code> <code><frequency> ::= hop frequency in Hz in NR3 format</code>
	The :WGEN:MODulation:FSKey:FREQuency command specifies the "hop frequency".
	The output frequency "shifts" between the original carrier frequency and this "hop frequency".
Query Syntax	<code>:WGEN:MODulation:FSKey:FREQuency?</code>
	The :WGEN:MODulation:FSKey:FREQuency? query returns the "hop frequency" setting.
Return Format	<code><frequency><NL></code> <code><frequency> ::= hop frequency in Hz in NR3 format</code>
See Also	<ul style="list-style-type: none"> · ":WGEN:MODulation:AM:DEPTH" on page 674 · ":WGEN:MODulation:AM:FREQuency" on page 675 · ":WGEN:MODulation:FM:DEViation" on page 676 · ":WGEN:MODulation:FM:FREQuency" on page 677 · ":WGEN:MODulation:FSKey:RATE" on page 679 · ":WGEN:MODulation:FUNCTION" on page 680 · ":WGEN:MODulation:FUNCTION:RAMP:SYMMetry" on page 681 · ":WGEN:MODulation:STATE" on page 683 · ":WGEN:MODulation:TYPE" on page 684

:WGEN:MODulation:FSKey:RATE

N (see [page 794](#))

Command Syntax :WGEN:MODulation:FSKey:RATE <rate>

<rate> ::= FSK modulation rate in Hz in NR3 format

The :WGEN:MODulation:FSKey:RATE command specifies the rate at which the output frequency "shifts".

The FSK rate specifies a digital square wave modulating signal.

Query Syntax :WGEN:MODulation:FSKey:RATE?

The :WGEN:MODulation:FSKey:RATE? query returns the FSK rate setting.

Return Format <rate><NL>

<rate> ::= FSK modulation rate in Hz in NR3 format

See Also

- "[:WGEN:MODulation:AM:DEPTh](#)" on page 674
- "[:WGEN:MODulation:AM:FREQuency](#)" on page 675
- "[:WGEN:MODulation:FM:DEViation](#)" on page 676
- "[:WGEN:MODulation:FM:FREQuency](#)" on page 677
- "[:WGEN:MODulation:FSKey:FREQuency](#)" on page 678
- "[:WGEN:MODulation:FUNCTION](#)" on page 680
- "[:WGEN:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 681
- "[:WGEN:MODulation:STATE](#)" on page 683
- "[:WGEN:MODulation:TYPE](#)" on page 684

:WGEN:MODulation:FUNCTION

N (see [page 794](#))

Command Syntax `:WGEN:MODulation:FUNCTION <shape>`
`<shape> ::= {SINusoid | SQUare| RAMP}`

The :WGEN:MODulation:FUNCTION command specifies the shape of the modulating signal.

When the RAMP shape is selected, you can specify the amount of time per cycle that the ramp waveform is rising with the :WGEN:MODulation:FUNCTION:RAMP:SYMMetry command.

This command applies to AM and FM modulation. (The FSK modulation signal is a square wave shape.)

Query Syntax `:WGEN:MODulation:FUNCTION?`

The :WGEN:MODulation:FUNCTION? query returns the specified modulating signal shape.

Return Format `<shape><NL>`
`<shape> ::= {SIN | SQU| RAMP}`

- See Also**
- "[":WGEN:MODulation:AM:DEPTH](#)" on page 674
 - "[":WGEN:MODulation:AM:FREQuency](#)" on page 675
 - "[":WGEN:MODulation:FM:DEViation](#)" on page 676
 - "[":WGEN:MODulation:FM:FREQuency](#)" on page 677
 - "[":WGEN:MODulation:FSKey:FREQuency](#)" on page 678
 - "[":WGEN:MODulation:FSKey:RATE](#)" on page 679
 - "[":WGEN:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 681
 - "[":WGEN:MODulation:STATE](#)" on page 683
 - "[":WGEN:MODulation:TYPE](#)" on page 684

:WGEN:MODulation:FUNCTION:RAMP:SYMMetry

N (see [page 794](#))

Command Syntax	<code>:WGEN:MODulation:FUNCTION:RAMP:SYMMetry <percent></code> <code><percent> ::= symmetry percentage from 0% to 100% in NR1 format</code>
	The :WGEN:MODulation:FUNCTION:RAMP:SYMMetry command specifies the amount of time per cycle that the ramp waveform is rising. The ramp modulating waveform shape is specified with the :WGEN:MODulation:FUNCTION command.
Query Syntax	<code>:WGEN:MODulation:FUNCTION:RAMP:SYMMetry?</code>
	The :WGEN:MODulation:FUNCTION:RAMP:SYMMetry? query returns ramp symmetry percentage setting.
Return Format	<code><percent><NL></code> <code><percent> ::= symmetry percentage from 0% to 100% in NR1 format</code>
See Also	<ul style="list-style-type: none"> · ":WGEN:MODulation:AM:DEPTH" on page 674 · ":WGEN:MODulation:AM:FREQuency" on page 675 · ":WGEN:MODulation:FM:DEViation" on page 676 · ":WGEN:MODulation:FM:FREQuency" on page 677 · ":WGEN:MODulation:FSKey:FREQuency" on page 678 · ":WGEN:MODulation:FSKey:RATE" on page 679 · ":WGEN:MODulation:FUNCTION" on page 680 · ":WGEN:MODulation:STATe" on page 683 · ":WGEN:MODulation:TYPE" on page 684

:WGEN:MODulation:NOISe**N** (see [page 794](#))

Command Syntax `:WGEN:MODulation:NOISe <percent>`
 `<percent> ::= 0 to 100`

The :WGEN:MODulation:NOISe command adds noise to the currently selected signal. The sum of the amplitude between the original signal and injected noise is limited to the regular amplitude limit (for example, 5 Vpp in 1 MΩ), so the range for <percent> varies according to current amplitude.

Note that adding noise affects edge triggering on the waveform generator source as well as the waveform generator sync pulse output signal (which can be sent to TRIG OUT). This is because the trigger comparator is located after the noise source.

Query Syntax `:WGEN:MODulation:NOISe?`

The :WGEN:MODulation:NOISe query returns the percent of added noise.

Return Format `<percent><NL>`
 `<percent> ::= 0 to 100`

See Also • [":WGEN:FUNCTION" on page 669](#)

:WGEN:MODulation:STATe

N (see [page 794](#))

Command Syntax `:WGEN:MODulation:STATe <setting>`
`<setting> ::= {{OFF | 0} | {ON | 1}}`

The :WGEN:MODulation:STATe command enables or disables modulated waveform generator output.

You can enable modulation for all waveform generator function types except pulse, DC, and noise.

Query Syntax `:WGEN:MODulation:STATe?`

The :WGEN:MODulation:STATe? query returns whether the modulated waveform generator output is enabled or disabled.

Return Format `<setting><NL>`
`<setting> ::= {0 | 1}`

- See Also**
- "[:WGEN:MODulation:AM:DEPTH](#)" on page 674
 - "[:WGEN:MODulation:AM:FREQuency](#)" on page 675
 - "[:WGEN:MODulation:FM:DEViation](#)" on page 676
 - "[:WGEN:MODulation:FM:FREQuency](#)" on page 677
 - "[:WGEN:MODulation:FSKey:FREQuency](#)" on page 678
 - "[:WGEN:MODulation:FSKey:RATE](#)" on page 679
 - "[:WGEN:MODulation:FUNCTION](#)" on page 680
 - "[:WGEN:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 681
 - "[:WGEN:MODulation:TYPE](#)" on page 684

:WGEN:MODulation:TYPE

N (see [page 794](#))

Command Syntax `:WGEN:MODulation:TYPE <type>`
`<type> ::= {AM | FM | FSK}`

The :WGEN:MODulation:TYPE command selects the modulation type:

- AM (amplitude modulation) – the amplitude of the original carrier signal is modified according to the amplitude of the modulating signal.

Use the :WGEN:MODulation:AM:FREQuency command to set the modulating signal frequency.

Use the :WGEN:MODulation:AM:DEPTh command to specify the amount of amplitude modulation.

- FM (frequency modulation) – the frequency of the original carrier signal is modified according to the amplitude of the modulating signal.

Use the :WGEN:MODulation:FM:FREQuency command to set the modulating signal frequency.

Use the :WGEN:MODulation:FM:DEViation command to specify the frequency deviation from the original carrier signal frequency.

- FSK (frequency-shift keying modulation) – the output frequency "shifts" between the original carrier frequency and a "hop frequency" at the specified FSK rate.

The FSK rate specifies a digital square wave modulating signal.

Use the :WGEN:MODulation:FSKey:FREQuency command to specify the "hop frequency".

Use the :WGEN:MODulation:FSKey:RATE command to specify the rate at which the output frequency "shifts".

Query Syntax `:WGEN:MODulation:TYPE?`

The :WGEN:MODulation:TYPE? query returns the selected modulation type.

Return Format `<type><NL>`

`<type> ::= {AM | FM | FSK}`

- See Also**
- "[:WGEN:MODulation:AM:DEPTh](#)" on page 674
 - "[:WGEN:MODulation:AM:FREQuency](#)" on page 675
 - "[:WGEN:MODulation:FM:DEViation](#)" on page 676
 - "[:WGEN:MODulation:FM:FREQuency](#)" on page 677
 - "[:WGEN:MODulation:FSKey:FREQuency](#)" on page 678
 - "[:WGEN:MODulation:FSKey:RATE](#)" on page 679

- [":WGEN:MODulation:FUNCTION" on page 680](#)
- [":WGEN:MODulation:FUNCTION:RAMP:SYMMetry" on page 681](#)
- [":WGEN:MODulation:STATe" on page 683](#)

:WGEN:OUTPut

N (see [page 794](#))

Command Syntax `:WGEN:OUTPut <on_off>`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :WGEN:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

Query Syntax `:WGEN:OUTPut?`

The :WGEN:OUTPut? query returns the current state of the waveform generator output setting.

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also · "Introduction to :WGEN Commands" on page 667

:WGEN:OUTPut:LOAD

N (see [page 794](#))

Command Syntax `:WGEN:OUTPut:LOAD <impedance>`
`<impedance> ::= {ONEMeg | FIFTy}`

The :WGEN:OUTPut:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax `:WGEN:OUTPut:LOAD?`

The :WGEN:OUTPut:LOAD? query returns the current expected output load impedance.

Return Format `<impedance><NL>`
`<impedance> ::= {ONEM | FIFT}`

See Also • ["Introduction to :WGEN Commands"](#) on page 667

:WGEN:PERiod

N (see [page 794](#))

Command Syntax `:WGEN:PERiod <period>`

`<period> ::= period in seconds in NR3 format`

For all waveforms except Noise and DC, the :WGEN:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN:FREQuency command.

Query Syntax `:WGEN:PERiod?`

The :WGEN:PERiod? query returns the currently set waveform generator period.

Return Format `<period><NL>`

`<period> ::= period in seconds in NR3 format`

See Also

- ["Introduction to :WGEN Commands"](#) on page 667
- [":WGEN:FUNCTION"](#) on page 669
- [":WGEN:FREQuency"](#) on page 668

:WGEN:RST

N (see [page 794](#))

Command Syntax :WGEN:RST

The :WGEN:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

See Also

- "[Introduction to :WGEN Commands](#)" on page 667
- "[:WGEN:FUNCTION](#)" on page 669
- "[:WGEN:FREQuency](#)" on page 668

:WGEN:VOLTage

N (see [page 794](#))

Command Syntax `:WGEN:VOLTage <amplitude>`

`<amplitude> ::= amplitude in volts in NR3 format`

For all waveforms except DC, the :WGEN:VOLTage command specifies the waveform's amplitude. Use the :WGEN:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

Query Syntax `:WGEN:VOLTage?`

The :WGEN:VOLTage? query returns the currently specified waveform amplitude.

Return Format `<amplitude><NL>`

`<amplitude> ::= amplitude in volts in NR3 format`

See Also

- "Introduction to :WGEN Commands" on page 667

- "[:WGEN:FUNCTION](#)" on page 669

- "[:WGEN:VOLTage:OFFSet](#)" on page 693

- "[:WGEN:VOLTage:HIGH](#)" on page 691

- "[:WGEN:VOLTage:LOW](#)" on page 692

:WGEN:VOLTage:HIGH

N (see [page 794](#))

Command Syntax :WGEN:VOLTage:HIGH <high>

<high> ::= high-level voltage in volts, in NR3 format

For all waveforms except DC, the :WGEN:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

Query Syntax :WGEN:VOLTage:HIGH?

The :WGEN:VOLTage:HIGH? query returns the currently specified waveform high-level voltage.

Return Format <high><NL>

<high> ::= high-level voltage in volts, in NR3 format

See Also

- "[Introduction to :WGEN Commands](#)" on page 667
- "[":WGEN:FUNCTION](#)" on page 669
- "[":WGEN:VOLTage:LOW](#)" on page 692
- "[":WGEN:VOLTage](#)" on page 690
- "[":WGEN:VOLTage:OFFSet](#)" on page 693

:WGEN:VOLTage:LOW

N (see [page 794](#))

Command Syntax `:WGEN:VOLTage:LOW <low>`

`<low>` ::= low-level voltage in volts, in NR3 format

For all waveforms except DC, the :WGEN:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

Query Syntax `:WGEN:VOLTage:LOW?`

The :WGEN:VOLTage:LOW? query returns the currently specified waveform low-level voltage.

Return Format `<low><NL>`

`<low>` ::= low-level voltage in volts, in NR3 format

See Also

- "[Introduction to :WGEN Commands](#)" on page 667
- "[":WGEN:FUNCTION](#)" on page 669
- "[":WGEN:VOLTage:LOW](#)" on page 692
- "[":WGEN:VOLTage](#)" on page 690
- "[":WGEN:VOLTage:OFFSet](#)" on page 693

:WGEN:VOLTage:OFFSet

N (see [page 794](#))

Command Syntax :WGEN:VOLTage:OFFSet <offset>

<offset> ::= offset in volts in NR3 format

The :WGEN:VOLTage:OFFSet command specifies the waveform's offset voltage or the DC level. Use the :WGEN:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

Query Syntax :WGEN:VOLTage:OFFSet?

The :WGEN:VOLTage:OFFSet? query returns the currently specified waveform offset voltage.

Return Format <offset><NL>

<offset> ::= offset in volts in NR3 format

See Also

- "[Introduction to :WGEN Commands](#)" on page 667
- "[":WGEN:FUNCTION"](#) on page 669
- "[":WGEN:VOLTage"](#) on page 690
- "[":WGEN:VOLTage:HIGH"](#) on page 691
- "[":WGEN:VOLTage:LOW"](#) on page 692

32 :WMEMory<r> Commands

Control reference waveforms.

Table 97 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 697)	n/a	<r> ::= 1-2 in NR1 format
:WMEMory<r>:DISPlay { {0 OFF} {1 ON} } (see page 698)	:WMEMory<r>:DISPlay? (see page 698)	<r> ::= 1-2 in NR1 format {0 1}
:WMEMory<r>:LABEL <string> (see page 699)	:WMEMory<r>:LABEL? (see page 699)	<r> ::= 1-2 in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 700)	n/a	<r> ::= 1-2 in NR1 format <source> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1 to (# analog channels) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 701)	:WMEMory<r>:SKEW? (see page 701)	<r> ::= 1-2 in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMory<r>:YOFFset <offset>[suffix] (see page 702)	:WMEMory<r>:YOFFset? (see page 702)	<r> ::= 1-2 in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V mV}

Table 97 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:YRANGE <range>[suffix] (see page 703)	:WMEMory<r>:YRANGE? (see page 703)	<r> ::= 1-2 in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V mV}
:WMEMory<r>:YScale <scale>[suffix] (see page 704)	:WMEMory<r>:YScale? (see page 704)	<r> ::= 1-2 in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V mV}

:WMEMory<r>:CLEar

N (see [page 794](#))

Command Syntax :WMEMory<r>:CLEar

<r> ::= 1-2 in NRI format

The :WMEMory<r>:CLEar command clears the specified reference waveform location.

See Also

- [Chapter 32, “:WMEMory<r> Commands,” starting on page 695](#)
- [":WMEMory<r>:SAVE" on page 700](#)
- [":WMEMory<r>:DISPLAY" on page 698](#)

:WMEMory<r>:DISPlay

N (see [page 794](#))

Command Syntax `:WMEMory<r>:DISPlay <on_off>`

`<r> ::= 1-2 in NRI format`

`<on_off> ::= {{1 | ON} | {0 | OFF}}`

The :WMEMory<r>:DISPlay command turns the display of the specified reference waveform on or off.

There are two reference waveform locations, but only one reference waveform can be displayed at a time. That means, if :WMEMory1:DISPlay is ON, sending the :WMEMory2:DISPlay ON command will automatically set :WMEMory1:DISPlay OFF.

Query Syntax `:WMEMory<r>:DISPlay?`

The :WMEMory<r>:DISPlay? query returns the current display setting for the reference waveform.

Return Format `<on_off><NL>`

`<on_off> ::= {1 | 0}`

See Also

- [Chapter 32](#), “:WMEMory<r> Commands,” starting on page 695
- [":WMEMory<r>:CLEar"](#) on page 697
- [":WMEMory<r>:LABEL"](#) on page 699

:WMEMory<r>:LABel

N (see [page 794](#))

Command Syntax :WMEMory<r>:LABel <string>

<r> ::= 1-2 in NRI format

<string> ::= quoted ASCII string

NOTE

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

The :WMEMory<r>:LABel command sets the reference waveform label to the string that follows.

Setting a label for a reference waveform also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

Query Syntax :WMEMory<r>:LABel?

The :WMEMory<r>:LABel? query returns the label associated with a particular reference waveform.

Return Format <string><NL>

<string> ::= quoted ASCII string

See Also

- [Chapter 32](#), “:WMEMory<r> Commands,” starting on page 695
- [":WMEMory<r>:DISPLAY"](#) on page 698

:WMEMory<r>:SAVE

N (see [page 794](#))

Command Syntax `:WMEMory<r>:SAVE <source>`

`<r>` ::= 1-2 in NR1 format

`<source>` ::= {CHANnel<n> | FUNCTion | MATH}

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

The :WMEMory<r>:SAVE command copies the analog channel or math function waveform to the specified reference waveform location.

NOTE

Only ADD or SUBtract math operations can be saved as reference waveforms.

See Also

- [Chapter 32, “:WMEMory<r> Commands,” starting on page 695](#)
- [":WMEMory<r>:DISPlay" on page 698](#)

:WMEMory<r>:SKEW

N (see [page 794](#))

Command Syntax `:WMEMory<r>:SKEW <skew>`

`<r> ::= 1-2 in NR1 format`

`<skew> ::= time in seconds in NR3 format`

The :WMEMory<r>:SKEW command sets the skew factor for the specified reference waveform.

Query Syntax `:WMEMory<r>:SKEW?`

The :WMEMory<r>:SKEW? query returns the current skew setting for the selected reference waveform.

Return Format `<skew><NL>`

`<skew> ::= time in seconds in NR3 format`

See Also

- [Chapter 32, “:WMEMory<r> Commands,” starting on page 695](#)
- [":WMEMory<r>:DISPlay" on page 698](#)
- [":WMEMory<r>:YOFFset" on page 702](#)
- [":WMEMory<r>:YRANge" on page 703](#)
- [":WMEMory<r>:YScale" on page 704](#)

:WMEMory<r>:YOFFset

N (see [page 794](#))

Command Syntax `:WMEMory<r>:YOFFset <offset> [<suffix>]`

`<r> ::= 1-2 in NR1 format`

`<offset> ::= vertical offset value in NR3 format`

`<suffix> ::= {v | mV}`

The :WMEMory<r>:YOFFset command sets the value that is represented at center screen for the selected reference waveform.

The range of legal values varies with the value set by the :WMEMory<r>:YRANge or :WMEMory<r>:YScale commands. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value. Legal values are affected by the probe attenuation setting.

Query Syntax `:WMEMory<r>:YOFFset?`

The :WMEMory<r>:YOFFset? query returns the current offset value for the selected reference waveform.

Return Format `<offset><NL>`

`<offset> ::= vertical offset value in NR3 format`

See Also

- [Chapter 32, “:WMEMory<r> Commands,” starting on page 695](#)
- [":WMEMory<r>:DISPlay" on page 698](#)
- [":WMEMory<r>:YRANge" on page 703](#)
- [":WMEMory<r>:YScale" on page 704](#)
- [":WMEMory<r>:SKEW" on page 701](#)

:WMEMory<r>:YRANge

N (see [page 794](#))

Command Syntax :WMEMory<r>:YRANge <range>[<suffix>]

<r> ::= 1-2 in NR1 format

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {v | mV}

The :WMEMory<r>:YRANge command defines the full-scale vertical axis of the selected reference waveform.

Legal values for the range are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax :WMEMory<r>:YRANge?

The :WMEMory<r>:YRANge? query returns the current full-scale range setting for the specified reference waveform.

Return Format <range><NL>

<range> ::= vertical full-scale range value in NR3 format

- See Also**
- [Chapter 32, “:WMEMory<r> Commands,” starting on page 695](#)
 - [":WMEMory<r>:DISPlay"](#) on page 698
 - [":WMEMory<r>:YOFFset"](#) on page 702
 - [":WMEMory<r>:SKEW"](#) on page 701
 - [":WMEMory<r>:YScale"](#) on page 704

:WMEMory<r>:YSCale

N (see [page 794](#))

Command Syntax `:WMEMory<r>:YSCale <scale>[<suffix>]`

`<r>` ::= 1-2 in NR1 format

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

`<suffix>` ::= {v | mV}

The :WMEMory<r>:YSCale command sets the vertical scale, or units per division, of the selected reference waveform.

Legal values for the scale are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax `:WMEMory<r>:YSCale?`

The :WMEMory<r>:YSCale? query returns the current scale setting for the specified reference waveform.

Return Format `<scale><NL>`

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

- See Also**
- [Chapter 32, “:WMEMory<r> Commands,” starting on page 695](#)
 - [":WMEMory<r>:DISPlay" on page 698](#)
 - [":WMEMory<r>:YOFFset" on page 702](#)
 - [":WMEMory<r>:YRANge" on page 703](#)
 - [":WMEMory<r>:SKEW" on page 701](#)

33 Obsolete and Discontinued Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs (see "[Obsolete Commands](#)" on page 794).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see page 219)	
ANALog<n>:COUpling	:CHANnel<n>:COUpling (see page 220)	
ANALog<n>:INVert	:CHANnel<n>:INVert (see page 223)	
ANALog<n>:LABEL	:CHANnel<n>:LABEL (see page 224)	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see page 225)	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see page 226)	
ANALog<n>:PMODe	none	
ANALog<n>:RANGE	:CHANnel<n>:RANGE (see page 232)	
:CHANnel:ACTivity (see page 710)	:ACTivity (see page 149)	
:CHANnel:LABEL (see page 711)	:CHANnel<n>:LABEL (see page 224) or :DIGital<d>:LABEL (see page 246)	use CHANnel<n>:LABEL for analog channels and use DIGital<d>:LABEL for digital channels

Obsolete Command	Current Command Equivalent	Behavior Differences
:CHANnel:THReShold (see page 712)	:POD<n>:THReShold (see page 417) or :DIGItal<d>:THReShold (see page 249)	
:CHANnel2:SKEW (see page 713)	:CHANnel<n>:PROBe:SKEW (see page 229)	
:CHANnel<n>:INPut (see page 714)	:CHANnel<n>:IMPedance (see page 222)	
:CHANnel<n>:PMODe (see page 715)	none	
:DISPlay:CONNect (see page 716)	:DISPlay:VECTors (see page 263)	
:DISPlay:ORDer (see page 717)	none	
:ERASe (see page 718)	:DISPlay:CLEar (see page 257)	
:EXternal:PMODe (see page 719)	none	
FUNCTION1, FUNCTION2	:FUNCTION Commands (see page 279)	ADD not included
:FUNCTION:SOURce (see page 720)	:FUNCTION:SOURce1 (see page 295)	Obsolete command has ADD, SUBTract, and MULTIply parameters; current command has GOFT parameter.
:FUNCTION:VIEW (see page 721)	:FUNCTION:DISPlay (see page 282)	
:HARDcopy:DESTination (see page 722)	:HARDcopy:FILEname (see page 723)	
:HARDcopy:FILEname (see page 723)	:RECall:FILEname (see page 421) :SAVE:FILEname (see page 421)	
:HARDcopy:GRAYscale (see page 724)	:HARDcopy:PALETTE (see page 311)	
:HARDcopy:IGColors (see page 725)	:HARDcopy:INKSaver (see page 303)	
:HARDcopy:PDRiver (see page 726)	:HARDcopy:APRinter (see page 300)	
:MEASure:LOWer (see page 727)	:MEASure:DEFine:THresholds (see page 346)	MEASure:DEFine:THresholds can define absolute values or percentage

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:SCRatch (see page 728)	:MEASure:CLEar (see page 345)	
:MEASure:TDELta (see page 729)	:MARKer:XDELta (see page 326)	
:MEASure:THResholds (see page 730)	:MEASure:DEFine:THResholds (see page 346)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TSTArt (see page 731)	:MARKer:X1Position (see page 322)	
:MEASure:TSTOp (see page 732)	:MARKer:X2Position (see page 324)	
:MEASure:TVOLT (see page 733)	:MEASure:TVALue (see page 367)	TVALue measures additional values such as dB, Vs, etc.
:MEASure:UPPer (see page 734)	:MEASure:DEFine:THResholds (see page 346)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 735)	:MARKer:YDELta (see page 331)	
:MEASure:VSTARt (see page 736)	:MARKer:Y1Position (see page 329)	
:MEASure:VSTOP (see page 737)	:MARKer:Y2Position (see page 330)	
:MTEST:AMASK:{SAVE STORe} (see page 738)	:SAVE:MASK[:START] (see page 437)	
:MTEST:AVERage (see page 739)	:ACQuire:TYPE AVERage (see page 193)	
:MTEST:AVERage:COUNT (see page 740)	:ACQuire:COUNT (see page 184)	
:MTEST:LOAD (see page 741)	:RECall:MASK[:START] (see page 422)	
:MTEST:RUMode (see page 742)	:MTEST:RMODe (see page 398)	
:MTEST:RUMode:SOFailure (see page 743)	:MTEST:RMODe:FACTion:STOP (see page 402)	
:MTEST:{START STOP} (see page 744)	:RUN (see page 174) or :STOP (see page 178)	
:MTEST:TRIGger:SOURce (see page 745)	:TRIGger Commands (see page 591)	There are various commands for setting the source with different types of triggers.

Obsolete Command	Current Command Equivalent	Behavior Differences
:PRINt? (see page 746)	:DISPlay:DATA? (see page 258)	
:SAVE:IMAGe:AREA (see page 748)	none	
:TIMEbase:DELay (see page 749)	:TIMEbase:POSIon (see page 582) or :TIMEbase:WINDOW:POSIon (see page 587)	TIMEbase:POSIon is position value of main time base; TIMEbase:WINDOW:POSIon is position value of zoomed (delayed) time base window.
:TRIGger:THReShold (see page 750)	:POD<n>:THReShold (see page 417) or :DIGItal<d>:THReShold (see page 249)	
:TRIGger:TV:TMode (see page 751)	:TRIGger:TV:MODE (see page 625)	

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 2000 X-Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERsistence INFinite (see page 262)	
CHANnel:MATH	:FUNCTION:OPERation (see page 291)	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection (see page 231)	Previous form of this command was used to enable/disable 50Ω protection. The new command resets a tripped protect and the query returns the status of TRIPed or NORMAL.
DISPlay:INVerse	none	
DISPlay:COLumn	none	
DISPlay:FREeze	none	
DISPlay:GRID	none	
DISPlay:LINE	none	
DISPlay:PIXel	none	

Discontinued Command	Current Command Equivalent	Comments
DISPlay:POSITION	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCTION:MOVE	none	
FUNCTION:PEAKS	none	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test) (see page 143)	
TRACE subsystem	none	All commands discontinued, feature not available
TRIGger:ADVanced subsystem		Use new GLITCH, PATTERN, or TV trigger modes
TRIGger:TV:FIELD	:TRIGger:TV:MODE (see page 625)	
TRIGger:TV:VHFREJ		
TRIGger:TV:VIR	none	
VAUToscale	none	

Discontinued Parameters Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision 2000 X-Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

:CHANnel:ACTivity**0** (see [page 794](#))**Command Syntax** :CHANnel:ACTivity

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

NOTE The :CHANnel:ACTivity command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :ACTivity command (see [page 149](#)) instead.

Query Syntax :CHANnel:ACTivity?

The :CHANnel:ACTivity? query returns the active edges since the last clear, and returns the current logic levels.

Return Format <edges>,<levels><NL>

<edges> ::= presence of edges (32-bit integer in NR1 format).

<levels> ::= logical highs or lows (32-bit integer in NR1 format).

NOTE A bit equal to zero indicates that no edges were detected at the specified threshold since the last clear on that channel. Edges may have occurred that were not detected because of the threshold setting.

A bit equal to one indicates that edges have been detected at the specified threshold since the last clear on that channel.

:CHANnel:LABEL (see [page 794](#))**Command Syntax** :CHANnel:LABEL <source_text><string>

<source_text> ::= {CHANnel1 | CHANnel2 | DIGital<d>}

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

<string> ::= quoted ASCII string

The :CHANnel:LABEL command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

NOTE

The :CHANnel:LABEL command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABEL command (see [page 224](#)) or :DIGital<n>:LABEL command (see [page 246](#)).

Query Syntax :CHANnel:LABEL?

The :CHANnel:LABEL? query returns the label associated with a particular analog channel.

Return Format <string><NL>

<string> ::= quoted ASCII string

:CHANnel:THreshold

 (see [page 794](#))

Command Syntax

```
:CHANnel:THreshold <channel group>, <threshold type> [, <value>]
<channel group> ::= {POD1 | POD2}
<threshold type> ::= {CMOS | ECL | TTL | USERdef}
<value> ::= voltage for USERdef in NR3 format [volt_type]
[volt_type] ::= {V | mV (-3) | uV (-6)}
```

The :CHANnel:THreshold command sets the threshold for a group of channels. The threshold is either set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is ignored.

NOTE

The :CHANnel:THreshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THreshold command (see [page 417](#)) or :DIGital<n>:THreshold command (see [page 249](#)).

Query Syntax

```
:CHANnel:THreshold? <channel group>
```

The :CHANnel:THreshold? query returns the voltage and threshold text for a specific group of channels.

Return Format

```
<threshold type> [, <value>]<NL>
<threshold type> ::= {CMOS | ECL | TTL | USERdef}
<value> ::= voltage for USERdef (float 32 NR3)
```

NOTE

- CMOS = 2.5V
- TTL = 1.5V
- ECL = -1.3V
- USERdef ::= -6.0V to 6.0V

:CHANnel2:SKEW

 (see [page 794](#))

Command Syntax :CHANnel2:SKEW <skew value>

<skew value> ::= skew time in NR3 format

<skew value> ::= -100 ns to +100 ns

The :CHANnel2:SKEW command sets the skew between channels 1 and 2. The maximum skew is +/-100 ns. You can use the oscilloscope's analog probe skew control to remove cable delay errors between channel 1 and channel 2.

NOTE

The :CHANnel2:SKEW command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:PROBe:SKEW command (see [page 229](#)) instead.

NOTE

This command is only valid for the two channel oscilloscope models.

Query Syntax

:CHANnel2:SKEW?

The :CHANnel2:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

<skew value><NL>

<skew value> ::= skew value in NR3 format

See Also • "Introduction to :CHANnel<n> Commands" on page 217

:CHANnel<n>:INPut (see [page 794](#))**Command Syntax** `:CHANnel<n>:INPut <impedance>``<impedance> ::= {ONEMeg | FIFTy}``<n> ::= 1 to (# analog channels) in NR1 format`

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

NOTE

The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see [page 222](#)) instead.

Query Syntax `:CHANnel<n>:INPut?`

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

Return Format `<impedance value><NL>``<impedance value> ::= {ONEM | FIFT}`

:CHANnel<n>:PMODe

 (see [page 794](#))

Command Syntax :CHANnel<n>:PMODe <pmode value>

<pmode value> ::= {AUTo | MANual}

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

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODe sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE

The :CHANnel<n>:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax :CHANnel<n>:PMODe?

The :CHANnel<n>:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format <pmode value><NL>

<pmode value> ::= {AUT | MAN}

:DISPlay:CONNect**0** (see [page 794](#))**Command Syntax** `:DISPlay:CONNect <connect>``<connect> ::= {{ 1 | ON} | {0 | OFF}}`

The :DISPlay:CONNect command turns vectors on and off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

NOTE

The :DISPlay:CONNect command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:VECTors command (see [page 263](#)) instead.

Query Syntax `:DISPlay:CONNect?`

The :DISPlay:CONNect? query returns the current state of the vectors setting.

Return Format `<connect><NL>``<connect> ::= {1 | 0}`**See Also** • " [":DISPlay:VECTors](#)" on page 263

:DISPlay:ORDer

 (see page 794)

Query Syntax :DISPLAY:ORDer?

The :DISPLAY:ORDer? query returns a list of digital channel numbers in screen order, from top to bottom, separated by commas. Busing is displayed as digital channels with no separator. For example, in the following list, the bus consists of digital channels 4 and 5: DIG1, DIG4 DIG5, DIG7.

NOTE

The :DISPLAY:ORDer command is an obsolete command provided for compatibility to previous oscilloscopes. This command is only available on the MSO models.

Return Format

```
<order><NL>
<order> ::= Unquoted ASCII string
```

NOTE

A return value is included for each digital channel. A return value of NONE indicates that a channel is turned off.

See Also

- [":DIGItal<d>:POsition"](#) on page 247

Example Code

```
' DISP_ORDER - Set the order the channels are displayed on the
' analyzer. You can enter between 1 and 32 channels at one time.
' If you leave out channels, they will not be displayed.

' Display ONLY channel 0 and channel 10 in that order.
myScope.WriteString ":DISPLAY:ORDER 0,10"
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:ERASe

 (see [page 794](#))

Command Syntax :ERASe

The :ERASe command erases the screen.

NOTE

The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPLAY:CLEAR command (see [page 257](#)) instead.

:EXTernal:PMODE**0** (see [page 794](#))**Command Syntax** `:EXTernal:PMODE <pmode value>``<pmode value> ::= {AUTo | MANual}`

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the pmode sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE

The :EXTernal:PMODE command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax `:EXTernal:PMODE?`

The :EXTernal:PMODE? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format `<pmode value><NL>``<pmode value> ::= {AUT | MAN}`

:FUNCTION:SOURce

 (see [page 794](#))

Command Syntax `:FUNCTION:SOURce <value>`

```
<value> ::= {CHANnel<n> | ADD | SUBTract | MULTiply}  
<n> ::= 1 to (# analog channels) in NR1 format
```

The :FUNCTION:SOURce command is only used when an FFT (Fast Fourier Transform) operation is selected (see the :FUNCTION:OPERation command for more information about selecting an operation). The :FUNCTION:SOURce command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for FFT operations specified by the :FUNCTION:OPERation command.

NOTE

The :FUNCTION:SOURce command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :FUNCTION:SOURce1 command (see [page 295](#)) instead.

Query Syntax `:FUNCTION:SOURce?`

The :FUNCTION:SOURce? query returns the current source for function operations.

Return Format `<value><NL>`

```
<value> ::= {CHAN<n> | ADD | SUBT | MULT}  
<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "[Introduction to :FUNCTION Commands](#)" on page 280
- "[":FUNCTION:OPERation](#)" on page 291

:FUNCTION:VIEW

 (see [page 794](#))

Command Syntax `:FUNCTION:VIEW <view>`
`<view> ::= {{1 | ON} | (0 | OFF)}`

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

NOTE

The :FUNCTION:VIEW command is provided for backward compatibility to previous oscilloscopes. Use the :FUNCTION:DISPLAY command (see [page 282](#)) instead.

Query Syntax `:FUNCTION:VIEW?`

The :FUNCTION:VIEW? query returns the current state of the selected function.

Return Format `<view><NL>`
`<view> ::= {1 | 0}`

:HARDcopy:DESTination

 (see [page 794](#))

Command Syntax `:HARDcopy:DESTination <destination>`
`<destination> ::= {CENTronics | FLOPpy}`

The :HARDcopy:DESTination command sets the hardcopy destination.

NOTE

The :HARDcopy:DESTination command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FILEname command (see [page 723](#)) instead.

Query Syntax `:HARDcopy:DESTination?`

The :HARDcopy:DESTination? query returns the selected hardcopy destination.

Return Format `<destination><NL>`
`<destination> ::= {CENT | FLOP}`

See Also

- "Introduction to :HARDcopy Commands" on page 298

:HARDcopy:FILEname

 (see [page 794](#))

Command Syntax `:HARDcopy:FILEname <string>`
`<string> ::= quoted ASCII string`

The HARDcopy:FILEname command sets the output filename for those print formats whose output is a file.

NOTE

The :HARDcopy:FILEname command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :SAVE:FILEname command ([see page 430](#)) and :RECall:FILEname command ([see page 421](#)) instead.

Query Syntax `:HARDcopy:FILEname?`

The :HARDcopy:FILEname? query returns the current hardcopy output filename.

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

See Also · "Introduction to :HARDcopy Commands" on page 298

:HARDcopy:GRAYscale (see [page 794](#))**Command Syntax** `:HARDcopy:GRAYscale <gray>``<gray> ::= {{OFF | 0} | {ON | 1}}`

The :HARDcopy:GRAYscale command controls whether grayscaling is performed in the hardcopy dump.

NOTE

The :HARDcopy:GRAYscale command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:PAlette command (see [page 311](#)) instead. ("":HARDcopy:GRAYscale ON" is the same as "":HARDcopy:PAlette GRAYscale" and "":HARDcopy:GRAYscale OFF" is the same as "":HARDcopy:PAlette COLOR".)

Query Syntax `:HARDcopy:GRAYscale?`

The :HARDcopy:GRAYscale? query returns a flag indicating whether grayscaling is performed in the hardcopy dump.

Return Format `<gray><NL>``<gray> ::= {0 | 1}`**See Also** · ["Introduction to :HARDcopy Commands"](#) on page 298

:HARDcopy:IGColors (see [page 794](#))**Command Syntax** `:HARDcopy:IGColors <value>``<value> ::= {{OFF | 0} | {ON | 1}}`

The HARDcopy:IGColors command controls whether the graticule colors are inverted or not.

NOTE

The :HARDcopy:IGColors command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:INKSaver (see [page 303](#)) command instead.

Query Syntax `:HARDcopy:IGColors?`

The :HARDcopy:IGColors? query returns a flag indicating whether graticule colors are inverted or not.

Return Format `<value><NL>``<value> ::= {0 | 1}`**See Also** · "Introduction to :HARDcopy Commands" on page 298

:HARDcopy:PDRiver

 (see [page 794](#))

Command Syntax `:HARDcopy:PDRiver <driver>`

```
<driver> ::= {AP2XXX | AP21XX | {AP2560 | AP25} | {DJ350 | DJ35} |
               DJ6XX | {DJ630 | DJ63} | DJ6Special | DJ6Photo |
               DJ8Special | DJ8XX | DJ9Vip | OJPRokx50 | DJ9XX | GVIP |
               DJ55XX | {PS470 | PS47} {PS100 | PS10} | CLASer |
               MLASer | LJFastraster | POSTscript}
```

The HARDcopy:PDRiver command sets the hardcopy printer driver used for the selected printer.

If the correct driver for the selected printer can be identified, it will be selected and cannot be changed.

NOTE

The :HARDcopy:PDRiver command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:APRinter (see [page 300](#)) command instead.

Query Syntax `:HARDcopy:PDRiver?`

The :HARDcopy:PDRiver? query returns the selected hardcopy printer driver.

Return Format `<driver><NL>`

```
<driver> ::= {AP2X | AP21 | AP25 | DJ35 | DJ6 | DJ63 | DJ6S | DJ6P |
               DJ8S | DJ8 | DJ9V | OJPR | DJ9 | GVIP | DJ55 | PS10 |
               PS47 | CLAS | MLAS | LJF | POST}
```

See Also · "Introduction to :HARDcopy Commands" on page 298

:MEASure:LOWER

 (see [page 794](#))

Command Syntax :MEASure:LOWER <voltage>

The :MEASure:LOWER command sets the lower measurement threshold value. This value and the UPPer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THresholds command.

NOTE

The :MEASure:LOWER command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THresholds command (see [page 346](#)) instead.

Query Syntax :MEASure:LOWER?

The :MEASure:LOWER? query returns the current lower threshold level.

Return Format <voltage><NL>

<voltage> ::= the user-defined lower threshold in volts in NR3 format

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:THresholds](#)" on page 730
- "[":MEASure:UPPer](#)" on page 734

:MEASure:SCRatch

 (see [page 794](#))

Command Syntax :MEASure:SCRatch

The :MEASure:SCRatch command clears all selected measurements and markers from the screen.

NOTE

The :MEASure:SCRatch command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:CLEar command (see [page 345](#)) instead.

:MEASure:TDELta

 (see [page 794](#))

Query Syntax `:MEASure:TDELta?`

The `:MEASure:TDELta?` query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

$$\text{Tdelta} = \text{Tstop} - \text{Tstart}$$

Tstart is the time at the start marker (X1 cursor) and Tstop is the time at the stop marker (X2 cursor). No measurement is made when the `:MEASure:TDELta?` query is received by the oscilloscope. The delta time value that is output is the current value. This is the same value as the front-panel cursors delta X value.

NOTE

The `:MEASure:TDELta` command is an obsolete command provided for compatibility to previous oscilloscopes. Use the `:MARKer:XDELta` command (see [page 326](#)) instead.

Return Format

`<value><NL>`

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

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MARKer:X1Position](#)" on page 322
- "[:MARKer:X2Position](#)" on page 324
- "[:MARKer:XDELta](#)" on page 326
- "[:MEASure:TSTArt](#)" on page 731
- "[:MEASure:TSTOP](#)" on page 732

:MEASure:THResholds

 (see [page 794](#))

Command Syntax `:MEASure:THResholds {T1090 | T2080 | VOLTage}`

The :MEASure:THResholds command selects the thresholds used when making time measurements.

NOTE

The :MEASure:THResholds command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see [page 346](#)) instead.

Query Syntax `:MEASure:THResholds?`

The :MEASure:THResholds? query returns the current thresholds selected when making time measurements.

Return Format `{T1090 | T2080 | VOLTage}<NL>`

`{T1090}` uses the 10% and 90% levels of the selected waveform.

`{T2080}` uses the 20% and 80% levels of the selected waveform.

`{VOLTage}` uses the upper and lower voltage thresholds set by the UPPer and LOWER commands on the selected waveform.

See Also

- "[Introduction to :MEASure Commands](#)" on page 342
- "[":MEASure:LOWER](#)" on page 727
- "[":MEASure:UPPer](#)" on page 734

:MEASure:TSTArt

 (see [page 794](#))

Command Syntax `:MEASure:TSTArt <value> [suffix]`

`<value>` ::= time at the start marker in seconds

`[suffix]` ::= {s | ms | us | ns | ps}

The :MEASure:TSTArt command moves the start marker (X1 cursor) to the specified time with respect to the trigger time.

NOTE

The short form of this command, TSTA, does not follow the defined Long Form to Short Form Truncation Rules (see [page 796](#)). The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTArt command produces an error.

NOTE

The :MEASure:TSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X1Position command (see [page 322](#)) instead.

Query Syntax

`:MEASure:TSTArt?`

The :MEASure:TSTArt? query returns the time at the start marker (X1 cursor).

Return Format

`<value><NL>`

`<value>` ::= time at the start marker in NR3 format

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MARKer:X1Position](#)" on page 322
- "[:MARKer:X2Position](#)" on page 324
- "[:MARKer:XDELta](#)" on page 326
- "[:MEASure:TDELta](#)" on page 729
- "[:MEASure:TSTOp](#)" on page 732

:MEASure:TSTOp

 (see [page 794](#))

Command Syntax `:MEASure:TSTOp <value> [suffix]`

`<value>` ::= time at the stop marker in seconds

`[suffix]` ::= {s | ms | us | ns | ps}

The :MEASure:TSTOp command moves the stop marker (X2 cursor) to the specified time with respect to the trigger time.

NOTE

The short form of this command, TSTO, does not follow the defined Long Form to Short Form Truncation Rules (see [page 796](#)). The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTOp command produces an error.

NOTE

The :MEASure:TSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X2Position command (see [page 324](#)) instead.

Query Syntax

`:MEASure:TSTOp?`

The :MEASure:TSTOp? query returns the time at the stop marker (X2 cursor).

Return Format

`<value><NL>`

`<value>` ::= time at the stop marker in NR3 format

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MARKer:X1Position](#)" on page 322
- "[:MARKer:X2Position](#)" on page 324
- "[:MARKer:XDELta](#)" on page 326
- "[:MEASure:TDELta](#)" on page 729
- "[:MEASure:TSTArt](#)" on page 731

:MEASure:TVOLt

0 (see [page 794](#))

Query Syntax `:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]`

`<value>` ::= the voltage level that the waveform must cross.

`<slope>` ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

`<occurrence>` ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

`<source>` ::= {`<digital channels>` | CHANnel<n> | FUNCTion | MATH}

`<digital channels>` ::= {DIGItal<d>} for the MSO models

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

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

When the :MEASure:TVOLt? query is sent, the displayed signal is searched for the specified voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified voltage can be negative or positive. To specify a negative voltage, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE

The :MEASure:TVOLt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:TVALue command (see [page 367](#)).

Return Format

`<value><NL>`

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

:MEASure:UPPer**0** (see [page 794](#))**Command Syntax** `:MEASure:UPPer <value>`

The :MEASure:UPPer command sets the upper measurement threshold value. This value and the LOWER value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THReholds command.

NOTE

The :MEASure:UPPer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THReholds command (see [page 346](#)) instead.

Query Syntax `:MEASure:UPPer?`

The :MEASure:UPPer? query returns the current upper threshold level.

Return Format `<value><NL>`

`<value>` ::= the user-defined upper threshold in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 342
 - "[":MEASure:LOWER"](#) on page 727
 - "[":MEASure:THReholds"](#) on page 730

:MEASure:VDELta

 (see [page 794](#))

Query Syntax `:MEASure:VDELta?`

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

$VDELta = \text{value at marker 2} - \text{value at marker 1}$

NOTE

The `:MEASure:VDELta` command is an obsolete command provided for compatibility to previous oscilloscopes. Use the `:MARKer:YDELta` command (see [page 331](#)) instead.

Return Format `<value><NL>`

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

See Also

- "[Introduction to :MARKer Commands](#)" on page 320
- "[Introduction to :MEASure Commands](#)" on page 342
- "[:MARKer:Y1Position](#)" on page 329
- "[:MARKer:Y2Position](#)" on page 330
- "[:MARKer:YDELta](#)" on page 331
- "[:MEASure:TDELta](#)" on page 729
- "[:MEASure:TSTArt](#)" on page 731

:MEASure:VSTArt

 (see [page 794](#))

Command Syntax `:MEASure:VSTArt <vstart_argument>`

`<vstart_argument> ::= value for vertical marker 1`

The :MEASure:VSTArt command moves the vertical marker (Y1 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X1Y1source command.

NOTE

The short form of this command, VSTA, does not follow the defined Long Form to Short Form Truncation Rules (see [page 796](#)). The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTArt command produces an error.

NOTE

The :MEASure:VSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y1Position command (see [page 329](#)) instead.

Query Syntax

`:MEASure:VSTArt?`

The :MEASure:VSTArt? query returns the current value of the Y1 cursor.

Return Format

`<value><NL>`

`<value> ::= voltage at voltage marker 1 in NR3 format`

See Also

- ["Introduction to :MARKer Commands"](#) on page 320
- ["Introduction to :MEASure Commands"](#) on page 342
- [":MARKer:Y1Position"](#) on page 329
- [":MARKer:Y2Position"](#) on page 330
- [":MARKer:YDELta"](#) on page 331
- [":MARKer:X1Y1source"](#) on page 323
- [":MEASure:SOURce"](#) on page 363
- [":MEASure:TDELta"](#) on page 729
- [":MEASure:TSTArt"](#) on page 731

:MEASure:VSTOp

 (see [page 794](#))

Command Syntax `:MEASure:VSTOp <vstop_argument>`

`<vstop_argument> ::= value for Y2 cursor`

The :MEASure:VSTOp command moves the vertical marker 2 (Y2 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X2Y2source command.

NOTE

The short form of this command, VSTO, does not follow the defined Long Form to Short Form Truncation Rules (see [page 796](#)). The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTOp command produces an error.

NOTE

The :MEASure:VSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y2Position command (see [page 330](#)) instead.

Query Syntax

`:MEASure:VSTOp?`

The :MEASure:VSTOp? query returns the current value of the Y2 cursor.

Return Format

`<value><NL>`

`<value> ::= value of the Y2 cursor in NR3 format`

See Also

- ["Introduction to :MARKer Commands"](#) on page 320
- ["Introduction to :MEASure Commands"](#) on page 342
- [":MARKer:Y1Position"](#) on page 329
- [":MARKer:Y2Position"](#) on page 330
- [":MARKer:YDELta"](#) on page 331
- [":MARKer:X2Y2source"](#) on page 325
- [":MEASure:SOURce"](#) on page 363
- [":MEASure:TDELta"](#) on page 729
- [":MEASure:TSTArt"](#) on page 731

:MTEST:AMASK:{SAVE | STORe} (see [page 794](#))**Command Syntax** :MTEST:AMASK:{SAVE | STORe} "<filename>"

The :MTEST:AMASK:SAVE command saves the automask generated mask to a file. If an automask has not been generated, an error occurs.

The <filename> parameter is an MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The filename assumes the present working directory if a path does not precede the file name.

NOTE

The :MTEST:AMASK:{SAVE | STORe} command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :SAVE:MASK[:STARt] command (see [page 437](#)) instead.

See Also • ["Introduction to :MTEST Commands"](#) on page 381

:MTEST:AVERage

 (see [page 794](#))

Command Syntax

```
:MTEST:AVERage <on_off>  
<on_off> ::= {{1 | ON} | {0 | OFF}}
```

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

NOTE

The :MTEST:AVERage command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :ACQuire:TYPE AVERage command (see [page 193](#)) instead.

Query Syntax

```
:MTEST:AVERage?
```

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

Return Format

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

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:AVERage:COUNt](#)" on page 740

:MTEST:AVERage:COUNt

 (see [page 794](#))

Command Syntax :MTEST:AVERage:COUNt <count>

<count> ::= an integer from 2 to 65536 in NR1 format

The :MTEST:AVERage:COUNt command sets the number of averages for the waveforms. With the AVERage acquisition type, the :MTEST:AVERage:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

NOTE

The :MTEST:AVERage:COUNt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :ACQuire:COUNt command (see [page 184](#)) instead.

Query Syntax :MTEST:AVERage:COUNt?

The :MTEST:AVERage:COUNt? query returns the currently selected count value.

Return Format <count><NL>

<count> ::= an integer from 2 to 65536 in NR1 format

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:AVERage"](#) on page 739

:MTEST:LOAD

 (see [page 794](#))

Command Syntax :MTEST:LOAD "<filename>"

The :MTEST:LOAD command loads the specified mask file.

The <filename> parameter is an MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used).

NOTE

The :MTEST:LOAD command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :RECall:MASK[:START] command (see [page 422](#)) instead.

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:AMASK:{SAVE | STORe}](#)" on page 738

:MTEST:RUMode

 (see [page 794](#))

Command Syntax `:MTEST:RUMode {FORever | TIME,<seconds> | {WAVeforms,<wfm_count>}}`
`<seconds> ::= from 1 to 86400 in NR3 format`
`<wfm_count> ::= number of waveforms in NR1 format`
`from 1 to 1,000,000,000`

The :MTEST:RUMode command determines the termination conditions for the mask test. The choices are FORever, TIME, or WAVeforms.

- FORever – runs the Mask Test until the test is turned off.
- TIME – sets the amount of time in seconds that a mask test will run before it terminates. The <seconds> parameter is a real number from 1 to 86400 seconds.
- WAVeforms – sets the maximum number of waveforms that are required before the mask test terminates. The <wfm_count> parameter indicates the number of waveforms that are to be acquired; it is an integer from 1 to 1,000,000,000.

NOTE

The :MTEST:RUMode command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MTEST:RMODe command (see [page 398](#)) instead.

Query Syntax

`:MTEST:RUMode?`

The :MTEST:RUMode? query returns the currently selected termination condition and value.

Return Format

`{FOR | TIME,<seconds> | {WAV,<wfm_count>}}<NL>`
`<seconds> ::= from 1 to 86400 in NR3 format`
`<wfm_count> ::= number of waveforms in NR1 format`
`from 1 to 1,000,000,000`

See Also

- "[Introduction to :MTEST Commands](#)" on page 381
- "[":MTEST:RUMode:SOFailure](#)" on page 743

:MTEST:RUMode:SOFailure

 (see [page 794](#))

Command Syntax

```
:MTEST:RUMode:SOFailure <on_off>
<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MTEST:RUMode:SOFailure command enables or disables the Stop On Failure run until criteria. When a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

NOTE

The :MTEST:RUMode:SOFailure command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MTEST:RMODe:FACtion:STOP command (see [page 402](#)) instead.

Query Syntax

```
:MTEST:RUMode:SOFailure?
```

The :MTEST:RUMode:SOFailure? query returns the current state of the Stop on Failure control.

Return Format

```
<on_off><NL>
```

```
<on_off> ::= {1 | 0}
```

See Also

- ["Introduction to :MTEST Commands"](#) on page 381
- [":MTEST:RUMode"](#) on page 742

:MTEST:{STARt | STOP}

 (see [page 794](#))

Command Syntax :MTEST:{START | STOP}

The :MTEST:{STARt | STOP} command starts or stops the acquisition system.

NOTE

The :MTEST:STARt and :MTEST:STOP commands are obsolete and are provided for backward compatibility to previous oscilloscopes. Use the :RUN command (see [page 174](#)) and :STOP command (see [page 178](#)) instead.

See Also • ["Introduction to :MTEST Commands"](#) on page 381

:MTEST:TRIGger:SOURce (see [page 794](#))**Command Syntax** `:MTEST:TRIGger:SOURce <source>` `<source> ::= CHANnel<n>` `<n> ::= 1 to (# analog channels) in NR1 format`

The :MTEST:TRIGger:SOURce command sets the channel to use as the trigger.

NOTE

The :MTEST:TRIGger:SOURce command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the trigger source commands (see [page 591](#)) instead.

Query Syntax `:MTEST:TRIGger:SOURce?`

The :MTEST:TRIGger:SOURce? query returns the currently selected trigger source.

Return Format `<source> ::= CHAN<n>` `<n> ::= 1 to (# analog channels) in NR1 format`**See Also**

- ["Introduction to :MTEST Commands"](#) on page 381

:PRINt?

 (see [page 794](#))

Query Syntax `:PRINt? [<options>]`

```
<options> ::= [<print option>] [,...,<print option>]
<print option> ::= {COLOR | GRAYscale | BMP8bit | BMP}
```

The :PRINt? query pulls image data back over the bus for storage.

NOTE

The :PRINT command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPLAY:DATA command (see [page 258](#)) instead.

Print Option	:PRINt command	:PRINt? query	Query Default
COLOR	Sets palette=COLOR		
GRAYscale	Sets palette=GRAYscale		palette=COLOR
PRINTER0,1	Causes the USB printer #0,1 to be selected as destination (if connected)	Not used	N/A
BMP8bit	Sets print format to 8-bit BMP	Selects 8-bit BMP formatting for query	N/A
BMP	Sets print format to BMP	Selects BMP formatting for query	N/A
FACTors	Selects outputting of additional settings information for :PRINT	Not used	N/A
NOFACTORS	Deselects outputting of additional settings information for :PRINT	Not used	N/A

Old Print Option:	Is Now:
Hires	COLOR
Lores	GRAYscale
Parallel	PRINTER0
DISK	invalid
PCL	invalid

NOTE

The PRINt? query is not a core command.

See Also

- "[Introduction to Root \(:\)](#) [Commands](#)" on page 148
- "[Introduction to :HARDcopy Commands](#)" on page 298
- "[:HARDcopy:FACTors](#)" on page 301
- "[:HARDcopy:GRAYscale](#)" on page 724
- "[:DISPlay:DATA](#)" on page 258

:SAVE:IMAGe:AREA

 (see [page 794](#))

Query Syntax `:SAVE:IMAGe:AREA?`

The `:SAVE:IMAGe:AREA?` query returns the selected image area.

When saving images, this query returns SCR (screen). When saving setups or waveform data, this query returns GRAT (graticule) even though graticule images are not saved.

Return Format `<area><NL>`

`<area> ::= {GRAT | SCR}`

See Also

- ["Introduction to :SAVE Commands"](#) on page 429

- [":SAVE:IMAGe\[:STARt\]"](#) on page 431
- [":SAVE:IMAGe:FACTors"](#) on page 432
- [":SAVE:IMAGe:FORMAT"](#) on page 433
- [":SAVE:IMAGe:INKSaver"](#) on page 434
- [":SAVE:IMAGe:PALETTE"](#) on page 435

:TIMEbase:DElay

 (see [page 794](#))

Command Syntax `:TIMEbase:DElay <delay_value>`
`<delay_value> ::= time in seconds from trigger to the delay reference point on the screen.`

The valid range for delay settings depends on the time/division setting for the main time base.

The :TIMEbase:DElay command sets the main time base delay. This delay is the time between the trigger event and the delay reference point on the screen. The delay reference point is set with the :TIMEbase:REFerence command (see [page 584](#)).

NOTE

The :TIMEbase:DElay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMEbase:POSition command (see [page 582](#)) instead.

Query Syntax `:TIMEbase:DElay?`

The :TIMEbase:DElay query returns the current delay value.

Return Format `<delay_value><NL>`
`<delay_value> ::= time from trigger to display reference in seconds in NR3 format.`

Example Code

```
' TIMEBASE_DELAY - Sets the time base delay. This delay
' is the internal time between the trigger event and the
' onscreen delay reference point.

' Set time base delay to 0.0.
myScope.WriteString ":TIMEBASE:DELAY 0.0"
```

See complete example programs at: [Chapter 38](#), “Programming Examples,” starting on page 803

:TRIGger:THreshold

 (see [page 794](#))

Command Syntax

```
:TRIGger:THreshold <channel group>, <threshold type> [, <value>]
<channel group> ::= {POD1 | POD2}
<threshold type> ::= {CMOS | ECL | TTL | USERdef}
<value> ::= voltage for USERdef (floating-point number) [Volt type]
[Volt type] ::= {V | mV | uV}
```

The :TRIGger:THreshold command sets the threshold (trigger level) for a pod of 8 digital channels (either digital channels 0 through 7 or 8 through 15). The threshold can be set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is not required.

NOTE

This command is only available on the MSO models.

NOTE

The :TRIGger:THreshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THreshold command (see [page 417](#)), :DIGItal<d>:THreshold command (see [page 249](#)), or :TRIGger[:EDGE]:LEVel command (see [page 605](#)).

Query Syntax

```
:TRIGger:THreshold? <channel group>
```

The :TRIGger:THreshold? query returns the voltage and threshold text for analog channel 1 or 2, or POD1 or POD2.

Return Format

```
<threshold type> [, <value>]<NL>
<threshold type> ::= {CMOS | ECL | TTL | USER}
CMOS ::= 2.5V
TTL ::= 1.5V
ECL ::= -1.3V
USERdef ::= range from -8.0V to +8.0V.
<value> ::= voltage for USERdef (a floating-point number in NR1.
```

:TRIGger:TV:TMode

0 (see [page 794](#))

Command Syntax :TRIGger:TV:TMode <mode>

```
<mode> ::= {FIELD1 | FIELD2 | AFIELDS | ALINES | LINE | VERTical
             | LFIELD1 | LFIELD2 | LALTernate | LVERTical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERTical parameter is only available when :TRIGger:TV:STANDARD is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANDARD is GENeric (see [page 628](#)).

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIELD1	F1
FIELD2	F2
AFIELD	ALLFIELDS, ALLFLDS
ALINES	ALLLINES
LFIELD1	LINEF1, LINEFIELD1
LFIELD2	LINEF2, LINEFIELD2
LALTernate	LINEALT
LVERTical	LINEVert

NOTE

The :TRIGger:TV:TMODE command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command (see [page 625](#)) instead.

Query Syntax :TRIGger:TV:TMODE?

The :TRIGger:TV:TMODE? query returns the TV trigger mode.

Return Format <value><NL>

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
             | LALT | LVER}
```

33 Obsolete and Discontinued Commands

34 Error Messages

-440, Query UNTERMINATED after indefinite response

-430, Query DEADLOCKED

-420, Query UNTERMINATED

-410, Query INTERRUPTED

-400, Query error

-340, Calibration failed

-330, Self-test failed

-321, Out of memory

-320, Storage fault

-315, Configuration memory lost

-314, Save/recall memory lost

-313, Calibration memory lost

-311, Memory error

-310, System error

-300, Device specific error

-278, Macro header not found

-277, Macro redefinition not allowed

-276, Macro recursion error

-273, Illegal macro label

-272, Macro execution error

-258, Media protected

-257, File name error

-256, File name not found

-255, Directory full

-254, Media full

-253, Corrupt media

-252, Missing media

-251, Missing mass storage

-250, Mass storage error

-241, Hardware missing

This message can occur when a feature is unavailable or unlicensed.

For example, some serial bus decode commands are only available when the serial decode options are licensed.

-240, Hardware error

-231, Data questionable

-230, Data corrupt or stale

-224, Illegal parameter value

-223, Too much data

-222, Data out of range

-221, Settings conflict

-220, Parameter error

-200, Execution error

-183, Invalid inside macro definition

-181, Invalid outside macro definition

-178, Expression data not allowed

-171, Invalid expression

-170, Expression error

-168, Block data not allowed

-161, Invalid block data

-158, String data not allowed

-151, Invalid string data

-150, String data error

-148, Character data not allowed

-138, Suffix not allowed

-134, Suffix too long

-131, Invalid suffix

-128, Numeric data not allowed

-124, Too many digits

-123, Exponent too large

-121, Invalid character in number

-120, Numeric data error

-114, Header suffix out of range

-113, Undefined header

-112, Program mnemonic too long

-109, Missing parameter

-108, Parameter not allowed

-105, GET not allowed

-104, Data type error

-103, Invalid separator

-102, Syntax error

-101, Invalid character

-100, Command error

+10, Software Fault Occurred

+100, File Exists

+101, End-Of-File Found

+102, Read Error

+103, Write Error

+104, Illegal Operation

+105, Print Canceled

+106, Print Initialization Failed

+107, Invalid Trace File

+108, Compression Error

+109, No Data For Operation

A remote operation wants some information, but there is no information available. For example, you may request a stored TIFF image using the :DISPLAY:DATA? query, but there may be no image stored.

+112, Unknown File Type

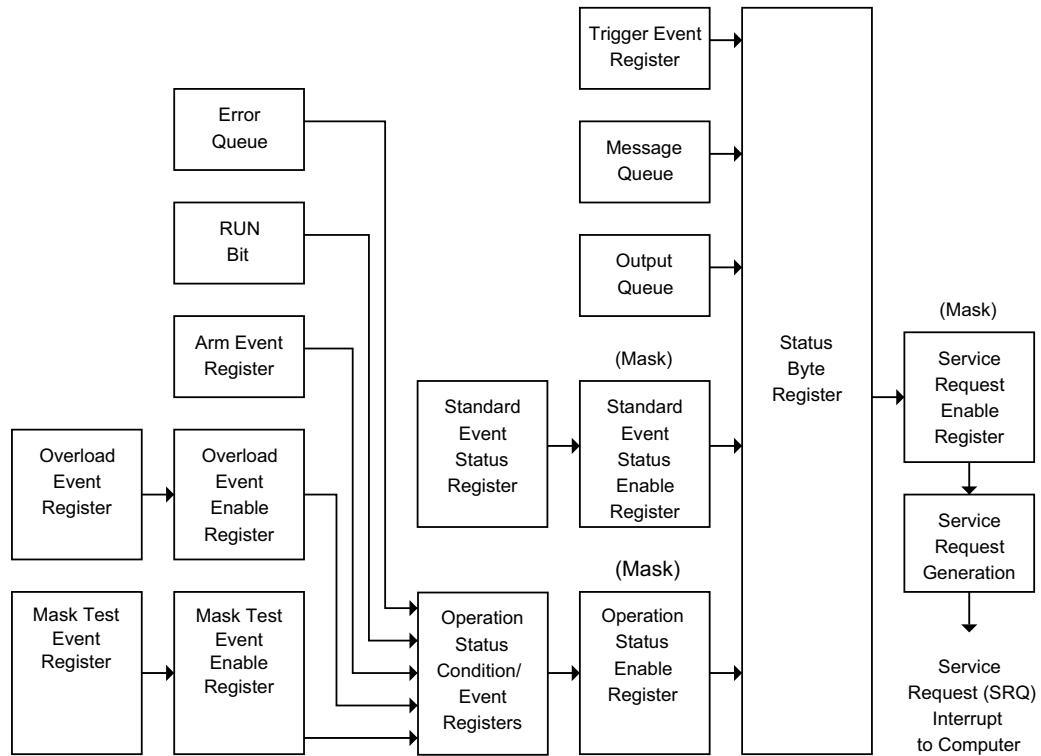
+113, Directory Not Supported

35 Status Reporting

Status Reporting Data Structures /	763
Status Byte Register (STB) /	766
Service Request Enable Register (SRE) /	768
Trigger Event Register (TER) /	769
Output Queue /	770
Message Queue /	771
(Standard) Event Status Register (ESR) /	772
(Standard) Event Status Enable Register (ESE) /	773
Error Queue /	774
Operation Status Event Register (:OPERegister[:EVENT]) /	775
Operation Status Condition Register (:OPERegister:CONDition) /	776
Arm Event Register (AER) /	777
Overload Event Register (:OVLRegister) /	778
Mask Test Event Event Register (:MTERegister[:EVENT]) /	779
Clearing Registers and Queues /	780
Status Reporting Decision Chart /	781

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting (for example, the Status Byte Register and the Standard Event Status Register). There are also instrument-defined structures and bits (for example, the Operation Status Event Register and the Overload Event Register).

An overview of the oscilloscope's status reporting structure is shown in the following block diagram. The status reporting structure allows monitoring specified events in the oscilloscope. The ability to monitor and report these events allows determination of such things as the status of an operation, the availability and reliability of the measured data, and more.



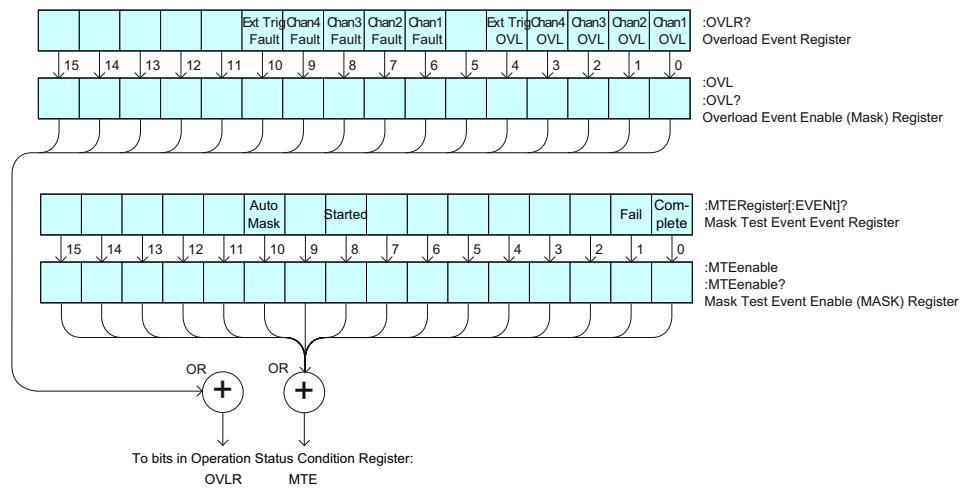
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the Service Request Enable register).

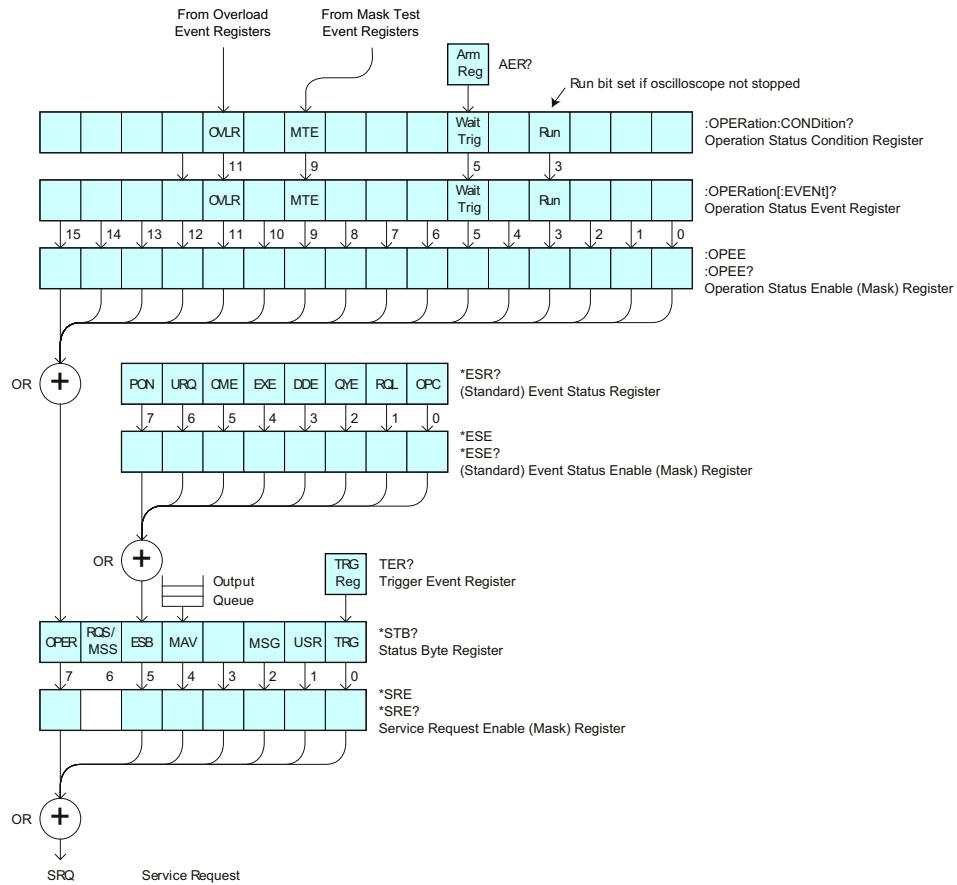
The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled with the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The *CLS command clears all event registers and all queues except the output queue. If you send *CLS immediately after a program message terminator, the output queue is also cleared.

Status Reporting Data Structures

The following figure shows how the status register bits are masked and logically OR'ed to generate service requests (SRQ) on particular events.





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

- [Table 48](#)
- [Table 46](#)
- [Table 53](#)
- [Table 54](#)
- [Table 56](#)
- [Table 51](#)

The status registers picture above shows how the different status reporting data structures work together. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, the bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to an external controller, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Status Byte Register (STB)

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

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

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

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

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

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

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

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

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

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

NOTE

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

Service Request Enable Register (SRE)

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

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

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

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

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

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

Trigger Event Register (TER)

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

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

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

Output Queue

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

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

Message Queue

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

(Standard) Event Status Register (ESR)

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

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

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

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

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

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

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

(Standard) Event Status Enable Register (ESE)

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

Set bits are read with the *ESE? query.

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

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

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

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

NOTE

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

Error Queue

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

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

The error queue is cleared when:

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

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

The Operation Status Event Register register hosts these bits:

Name	Location	Description
RUN bit	bit 3	Is set whenever the instrument goes from a stop state to a single or running state.
WAIT TRIG bit	bit 5	Is set by the Trigger Armed Event Register and indicates that the trigger is armed.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.

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

Operation Status Condition Register (:OPERegister:CONDITION)

The Operation Status Condition Register register hosts these bits:

Name	Location	Description
RUN bit	bit 3	Is set whenever the instrument is not stopped.
WAIT TRIG bit	bit 5	Is set by the Trigger Armed Event Register and indicates that the trigger is armed.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.

The :OPERegister:CONDITION? query returns the value of the Operation Status Condition Register.

Arm Event Register (AER)

This register sets bit 5 (Wait Trig bit) in the Operation Status Register and the OPER bit (bit 7) in the Status Byte Register when the instrument becomes armed.

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

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

Overload Event Register (:OVLRegister)

The Overload Event Register register hosts these bits:

Name	Location	Description
Channel 1 Fault	bit 6	Fault has occurred on Channel 1 input.
Channel 2 Fault	bit 7	Fault has occurred on Channel 2 input.
Channel 3 Fault	bit 8	Fault has occurred on Channel 3 input.
Channel 4 Fault	bit 9	Fault has occurred on Channel 4 input.
External Trigger Fault	bit 10	Fault has occurred on External Trigger input.

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

The Mask Test Event Event Register register hosts these bits:

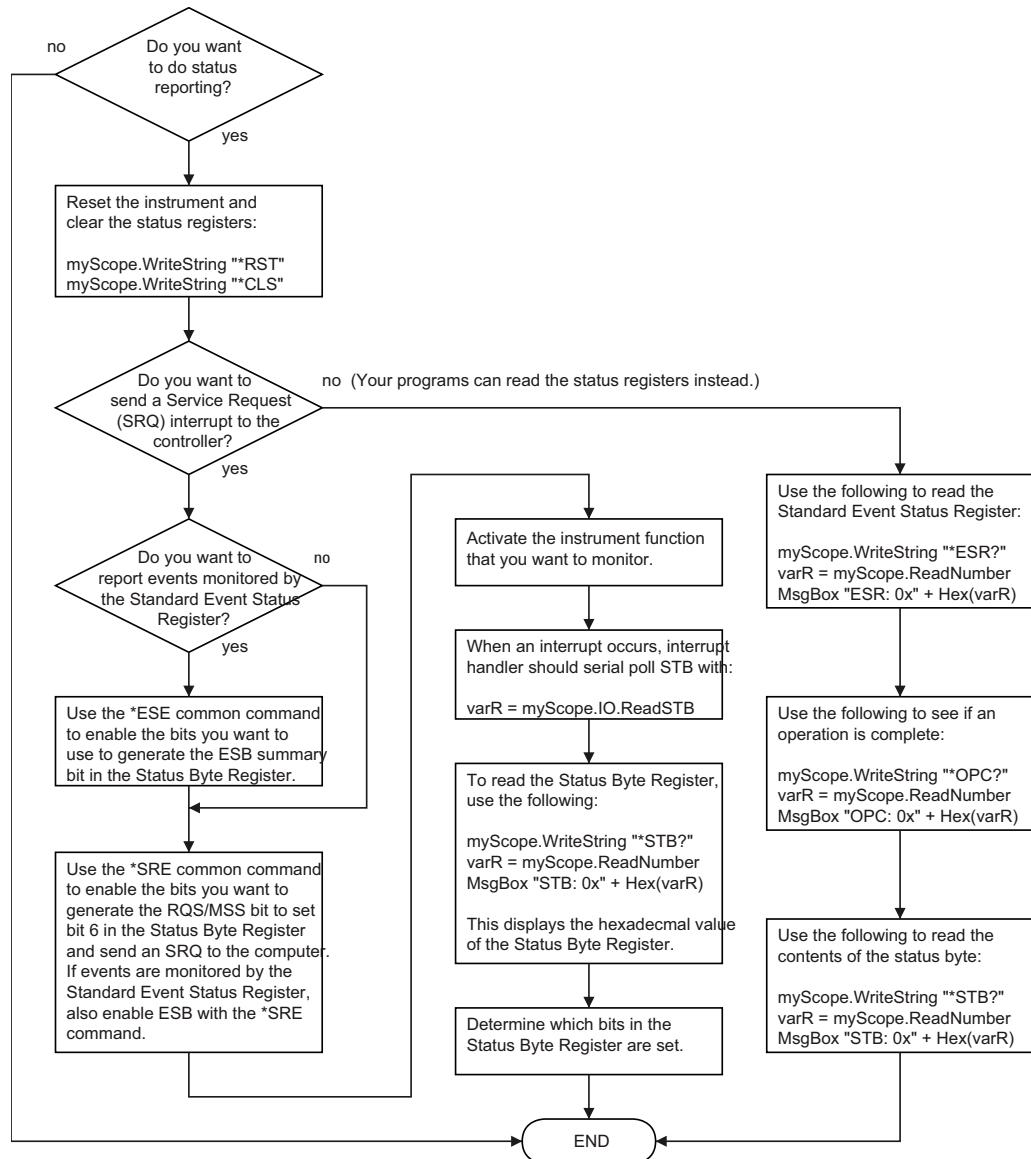
Name	Location	Description
Complete	bit 0	Is set when the mask test is complete.
Fail	bit 1	Is set when there is a mask test failure.
Started	bit 8	Is set when mask testing is started.
Auto Mask	bit 10	Is set when auto mask creation is completed.

The :MTERegister[:EVENT]? query returns the value of, and clears, the Mask Test Event Event Register.

Clearing Registers and Queues

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

Status Reporting Decision Chart



36 Synchronizing Acquisitions

Synchronization in the Programming Flow /	784
Blocking Synchronization /	785
Polling Synchronization With Timeout /	786
Synchronizing with a Single-Shot Device Under Test (DUT) /	788
Synchronization with an Averaging Acquisition /	790

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

Synchronization in the Programming Flow

Most remote programming follows these three general steps:

- 1 Set up the oscilloscope and device under test (see [page 784](#)).
- 2 Acquire a waveform (see [page 784](#)).
- 3 Retrieve results (see [page 784](#)).

Set Up the Oscilloscope

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

NOTE

It is not necessary to use *OPC?, hard coded waits, or status checking when setting up the oscilloscope. After the oscilloscope is configured, it is ready for an acquisition.

Acquire a Waveform

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

	Blocking Wait	Polling Wait
Use When	You know the oscilloscope <i>will</i> trigger based on the oscilloscope setup and device under test.	You know the oscilloscope <i>may or may not</i> trigger on the oscilloscope setup and device under test.
Advantages	No need for polling. Fastest 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. Requires polling loop. Requires known maximum wait time.
Implementation Details	See " Blocking Synchronization " on page 785.	See " Polling Synchronization With Timeout " on page 786.

Retrieve Results

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

Blocking Synchronization

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

```

'
' Synchronizing acquisition using blocking.
' =====

Option Explicit

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

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488
    Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
    myScope.IO.Clear      ' Clear the interface.

    ' Set up.
    ' -----
    myScope.WriteString ":TRIGGER:MODE EDGE"
    myScope.WriteString ":TRIGGER:EDGE:LEVEL 2"
    myScope.WriteString ":TIMEBASE:SCALE 5e-8"

    ' Acquire.
    ' -----
    myScope.WriteString ":DIGITIZE"

    ' Get results.
    ' -----
    myScope.WriteString ":MEASURE:RISETIME"
    myScope.WriteString ":MEASURE:RISETIME?"
    varQueryResult = myScope.ReadNumber      ' Read risetime.
    Debug.Print "Risetime: " + _
               FormatNumber(varQueryResult * 1000000000, 1) + " ns"

    Exit Sub

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

End Sub

```

Polling Synchronization With Timeout

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

```

'
' Synchronizing acquisition using polling.
' =====

Option Explicit

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

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

Sub Main()

On Error GoTo VisaComError

' Create the VISA COM I/O resource.
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488
Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
myScope.IO.Clear      ' Clear the interface.

' Set up.
' -----
' Set up the trigger and horizontal scale.
myScope.WriteString ":TRIGger:MODE EDGE"
myScope.WriteString ":TRIGger:EDGE:LEVel 2"
myScope.WriteString ":TIMEbase:SCALE 5e-8"

' Stop acquisitions and wait for the operation to complete.
myScope.WriteString ":STOP"
myScope.WriteString "*OPC?"
strQueryResult = myScope.ReadString

' Acquire.
' -----
' Start a single acquisition.
myScope.WriteString ":SINGLE"

' Oscilloscope is armed and ready, enable DUT here.
Debug.Print "Oscilloscope is armed and ready, enable DUT."

' Look for RUN bit = stopped (acquisition complete).
Dim lngTimeout As Long      ' Max millisecs to wait for single-shot.
Dim lngElapsed As Long
lngTimeout = 10000      ' 10 seconds.
lngElapsed = 0

Do While lngElapsed <= lngTimeout

```

```
myScope.WriteString ":OPERegister:CONDITION?"
varQueryResult = myScope.ReadNumber
' Mask RUN bit (bit 3, &H8).
If (varQueryResult And &H8) = 0 Then
    Exit Do
Else
    Sleep 100      ' Small wait to prevent excessive queries.
    lngElapsed = lngElapsed + 100
End If
Loop

' Get results.
' -----
If lngElapsed < lngTimeout Then
    myScope.WriteString ":MEASure:RISetime"
    myScope.WriteString ":MEASure:RISetime?"
    varQueryResult = myScope.ReadNumber      ' Read risetime.
    Debug.Print "Risetime: " + _
        FormatNumber(varQueryResult * 1000000000, 1) + " ns"
Else
    Debug.Print "Timeout waiting for single-shot trigger."
End If

Exit Sub

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

End Sub
```

Synchronizing with a Single-Shot Device Under Test (DUT)

The examples in ["Blocking Synchronization" on page 785](#) and ["Polling Synchronization With Timeout" on page 786](#) assume the DUT is continually running and therefore the oscilloscope will have more than one opportunity to trigger. With a single shot DUT, there is only one opportunity for the oscilloscope to trigger, so it is necessary for the oscilloscope to be armed and ready before the DUT is enabled.

NOTE

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

This example is the same ["Polling Synchronization With Timeout" on page 786](#) with the addition of checking for the armed event status.

```
' Synchronizing single-shot acquisition using polling.
' =====

Option Explicit

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

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

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488
    Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
    myScope.IO.Clear      ' Clear the interface.

    ' Set up.
    ' -----
    ' Set up the trigger and horizontal scale.
    myScope.WriteString ":TRIGger:MODE EDGE"
    myScope.WriteString ":TRIGger:EDGE:LEVel 2"
    myScope.WriteString ":TIMEbase:SCALE 5e-8"

    ' Stop acquisitions and wait for the operation to complete.
    myScope.WriteString ":STOP"
    myScope.WriteString "*OPC?"
    strQueryResult = myScope.ReadString

    ' Acquire.
```

```

' -----
' Start a single acquisition.
myScope.WriteString ":SINGLE"

' Wait until the trigger system is armed.
Do
    Sleep 100      ' Small wait to prevent excessive queries.
    myScope.WriteString ":AER?"
    varQueryResult = myScope.ReadNumber
Loop Until varQueryResult = 1

' Oscilloscope is armed and ready, enable DUT here.
Debug.Print "Oscilloscope is armed and ready, enable DUT."

' Now, look for RUN bit = stopped (acquisition complete).
Dim lngTimeout As Long      ' Max millisecs to wait for single-shot.
Dim lngElapsed As Long
lngTimeout = 10000      ' 10 seconds.
lngElapsed = 0

Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERRegister:CONDITION?"
    varQueryResult = myScope.ReadNumber
    ' Mask RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
        Exit Do
    Else
        Sleep 100      ' Small wait to prevent excessive queries.
        lngElapsed = lngElapsed + 100
    End If
Loop

' Get results.
' -----
If lngElapsed < lngTimeout Then
    myScope.WriteString ":MEASure:RISetime"
    myScope.WriteString ":MEASure:RISetime?"
    varQueryResult = myScope.ReadNumber      ' Read risetime.
    Debug.Print "Risetime: " + _
        FormatNumber(varQueryResult * 1000000000, 1) + " ns"
Else
    Debug.Print "Timeout waiting for single-shot trigger."
End If

Exit Sub

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

End Sub

```

Synchronization with an Averaging Acquisition

When averaging, it is necessary to know when the average count has been reached. The :SINGle command does not average.

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

The example below polls during the :DIGItize to prevent a timeout on the connection.

```
' Synchronizing in averaging acquisition mode.
' =====

Option Explicit

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

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

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488
    Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
    myScope.IO.Clear      ' Clear the interface.
    myScope.IO.Timeout = 5000

    ' Set up.
    ' -----
    ' Set up the trigger and horizontal scale.
    myScope.WriteString ":TRIGger:SWEep NORMal"
    myScope.WriteString ":TRIGger:MODE EDGE"
    myScope.WriteString ":TRIGger:EDGE:LEVel 2"
    myScope.WriteString ":TIMEbase:SCALe 5e-8"

    ' Stop acquisitions and wait for the operation to complete.
    myScope.WriteString ":STOP"
    myScope.WriteString "*OPC?"
    strQueryResult = myScope.ReadString

    ' Set up average acquisition mode.
    Dim lngAverages As Long
    lngAverages = 256
    myScope.WriteString ":ACQuire:COUNT " + CStr(lngAverages)
    myScope.WriteString ":ACQuire:TYPE AVERAGE"
```

```

' Save *ESE (Standard Event Status Enable register) mask
' (so it can be restored later).
Dim varInitialESE As Variant
myScope.WriteString "*ESE?"
varInitialESE = myScope.ReadNumber

' Set *ESE mask to allow only OPC (Operation Complete) bit.
myScope.WriteString "*ESE " + CStr(CInt("&H01"))

' Acquire using :DIGitize. Set up OPC bit to be set when the
' operation is complete.
' -----
myScope.WriteString ":DIGitize"
myScope.WriteString "*OPC"

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

' Wait until OPC becomes true (bit 5 of Status Byte register, STB,
' from Standard Event Status register, ESR, is set). STB can be
' read during :DIGitize without generating a timeout.
Do
    Sleep 4000      ' Poll more often than the timeout setting.
    varQueryResult = myScope.IO.ReadSTB
Loop While (varQueryResult And &H20) = 0

' Clear ESR and restore previously saved *ESE mask.
myScope.WriteString "*ESR?"      ' Clear ESR by reading it.
varQueryResult = myScope.ReadNumber
myScope.WriteString "*ESE " + CStr(varInitialESE)

' Get results.
' -----
myScope.WriteString ":WAVeform:COUNt?"
varQueryResult = myScope.ReadNumber
Debug.Print "Averaged waveforms: " + CStr(varQueryResult)

myScope.WriteString ":MEASure:RISetime"
myScope.WriteString ":MEASure:RISetime?"
varQueryResult = myScope.ReadNumber      ' Read risetime.
Debug.Print "Risetime: " +
    FormatNumber(varQueryResult * 1000000000, 1) + " ns"

Exit Sub

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

End Sub

```


37 More About Oscilloscope Commands

- Command Classifications / 794
- Valid Command/Query Strings / 795
- Query Return Values / 801
- All Oscilloscope Commands Are Sequential / 802

Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of Keysight InfiniiVision oscilloscopes, commands are classified by the following categories:

- ["Core Commands"](#) on page 794
- ["Non-Core Commands"](#) on page 794
- ["Obsolete Commands"](#) on page 794

C Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Keysight InfiniiVision oscilloscopes. Core commands are unlikely to be modified in the future. If you restrict your programs to core commands, the programs should work across product offerings in the future, assuming appropriate programming methods are employed.

N Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all Keysight InfiniiVision oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Keysight's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

O Obsolete Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

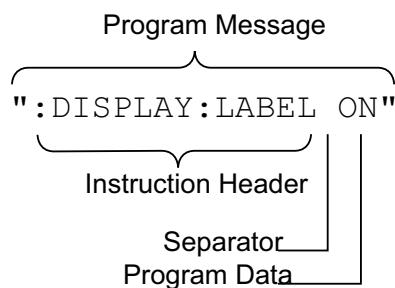
- [Chapter 33](#), “Obsolete and Discontinued Commands,” starting on page 705

Valid Command/Query Strings

- "Program Message Syntax" on page 795
- "Duplicate Mnemonics" on page 799
- "Tree Traversal Rules and Multiple Commands" on page 799

Program Message Syntax

To program the instrument remotely, you must understand the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, program data, and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses are formatted. The following figure shows the main syntactical parts of a typical program statement.



Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as Visual Basic or C/C++. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as <learn string>. There are only a few instructions that use block data.

Program messages can have long or short form commands (and data in some cases – see "Long Form to Short Form Truncation Rules" on page 796), and upper and/or lower case ASCII characters may be used. (Query responses, however, are always returned in upper case.)

Instructions are composed of two main parts:

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

Instruction Header The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument.

"":DISPLAY:LABEL ON" is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, "":DISPLAY:LABEL?". Many instructions can be used as either commands or queries, depending on whether or

not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:

- "Simple Command Headers" on page 797
- "Compound Command Headers" on page 797
- "Common Command Headers" on page 797

White Space (Separator) White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. White space is defined as one or more space characters. ASCII defines a space to be character 32 (in decimal).

Program Data Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data, as well as the values they accept. "Program Data Syntax Rules" on page 798 describes all of the general rules about acceptable values.

When there is more than one data parameter, they are separated by commas(,). Spaces can be added around the commas to improve readability.

Program Message Terminator The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the programming interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

NOTE

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

Long Form to Short Form Truncation Rules

To get the short form of a command/keyword:

- When the command/keyword is longer than four characters, use the first four characters of the command/keyword unless the fourth character is a vowel; when the fourth character is a vowel, use the first three characters of the command/keyword.
- When the command/keyword is four or fewer characters, use all of the characters.

Long Form	Short form
RANGE	RANG
PATTERn	PATT

Long Form	Short form
TIMebase	TIM
DELay	DEL
TYPE	TYPE

In the oscilloscope programmer's documentation, the short form of a command is indicated by uppercase characters.

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

Simple Command Headers

Simple command headers contain a single mnemonic. :AUToscale and :DIGitize are examples of simple command headers typically used in the oscilloscope. The syntax is:

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

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :DIGITIZE CHANnel1), white space is added to separate the data from the header. The syntax is:

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

Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example, to execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example, :CHANnel1:BWLIMIT ON

Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Program Data Syntax Rules

Program data is used to convey a parameter information related to the command header. At least one space must separate the command header or query header from the program data.

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

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

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

For example, :MEASure:DELay CHANnel1,CHANnel2 has two program data: CHANnel1 and CHANnel2.

Two main types of program data are used in commands: character and numeric.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEbase:MODE command can be set to normal, zoomed (delayed), XY, or ROLL. The character program data in this case may be MAIN, WINDOW, XY, or ROLL. The command :TIMEbase:MODE WINDOW sets the time base mode to zoomed.

The available mnemonics for character program data are always included with the command's syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEbase:RANGE requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

```
28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.
```

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

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGe may be used to change the vertical range or to change the horizontal range:

```
:CHANnel1:RANGE .4
```

Sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMEbase:RANGE 1
```

Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMEbase are subsystem selectors and determine which range is being modified.

Tree Traversal Rules and Multiple Commands

Command headers are created by traversing down the command tree. A legal command header would be :TIMEbase:RANGe. This is referred to as a *compound header*. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces.

The following rules apply to traversing the tree:

- A leading colon (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command lets you access that subsystem until a leading colon or a program message terminator (<NL>) or EOI true is found.
- In the command tree, use the last mnemonic in the compound header as the reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMEbase:). That is the point where the parser resides. Any command below that point can be sent within the current program message without sending the mnemonics which appear above them (for example, POSition).

The output statements in the examples are written using the Keysight VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

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

For example:

```
myScope.WriteString ":TIMEbase:RANGE 0.5;POSITION 0"
```

NOTE

The colon between TIMebase and RANGe is necessary because TIMebase:RANGE is a compound command. The semicolon between the RANGE command and the POSition command is the required program message unit separator. The POSition command does not need TIMebase preceding it because the TIMebase:RANGE command sets the parser to the TIMebase node in the tree.

Example 2:
Program Message Terminator Sets Parser Back to Root

```
myScope.WriteString ":TIMebase:REFerence CENTER;POSITION 0.00001"
```

or

```
myScope.WriteString ":TIMebase:REFerence CENTER"
myScope.WriteString ":TIMebase:POSITION 0.00001"
```

NOTE

In the first line of example 2, the subsystem selector is implied for the POSITION command in the compound command. The POSITION command must be in the same program message as the REFerence command because the program message terminator places the parser back at the root of the command tree.

A second way to send these commands is by placing TIMebase: before the POSITION command as shown in the second part of example 2. The space after POSITION is required.

Example 3:
Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

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

For example:

```
myScope.WriteString ":TIMebase:REFerence CENTER;:DISPlay:VECTors ON"
```

NOTE

The leading colon before DISPlay:VECTors ON tells the parser to go back to the root of the command tree. The parser can then see the DISPlay:VECTors ON command. The space between REFerence and CENter is required; so is the space between VECTors and ON.

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

Query Return Values

Command headers immediately followed by a question mark (?) are queries. Queries are used to get results of measurements made by the instrument or to find out how the instrument is currently configured.

After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued.

When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMEbase:RANGE? places the current time base setting in the output queue. When using the Keysight VISA COM library in Visual Basic, the controller statements:

```
Dim strQueryResult As String  
myScope.WriteString ":TIMEbase:RANGE?"  
strQueryResult = myScope.ReadString
```

pass the value across the bus to the controller and place it in the variable strQueryResult.

NOTE

Read Query Results Before Sending Another Command. Sending another command or query before reading the result of a query clears the output buffer (the current response) and places a Query INTERRUPTED error in the error queue.

Infinity Representation

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

All Oscilloscope Commands Are Sequential

IEEE 488.2 makes the distinction between sequential and overlapped commands:

- *Sequential commands* finish their task before the execution of the next command starts.
- *Overlapped commands* run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

All of the oscilloscope commands are sequential.

38 Programming Examples

VISA COM Examples / 804

VISA Examples / 837

SICL Examples / 884

SCPI.NET Examples / 904

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 804
- "VISA COM Example in C#" on page 813
- "VISA COM Example in Visual Basic .NET" on page 822
- "VISA COM Example in Python" on page 830

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 Keysight VISA COM library:
 - a Choose **Tools > References...** from the main menu.
 - b In the References dialog, check the "VISA COM 5.5 Type Library".
 - c Click **OK**.
- 4 Choose **Insert > Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the VISA address of your oscilloscope, and save the changes.
- 7 Run the program.

```

'
' Keysight VISA COM Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' -----


Option Explicit

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

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' -----


Sub Main()

```

```

On Error GoTo VisaComError

' Create the VISA COM I/O resource.
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488
Set myScope.IO =
    myMgr.Open("USB0::0x0957::0x17A6::US50210029::0::INSTR")
myScope.IO.Clear      ' Clear the interface.
myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

' Initialize - start from a known state.
Initialize

' Capture data.
Capture

' Analyze the captured waveform.
Analyze

Exit Sub

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

End Sub

'
' Initialize the oscilloscope to a known state.
' -----
Private Sub Initialize()

On Error GoTo VisaComError

' Get and display the device's *IDN? string.
strQueryResult = DoQueryString("*IDN?")
Debug.Print "Identification string: " + strQueryResult

' Clear status and load the default setup.
DoCommand "*CLS"
DoCommand "*RST"

Exit Sub

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

End Sub

'
' Capture the waveform.
' -----
Private Sub Capture()

```

```

On Error GoTo VisaComError

' Use auto-scale to automatically configure oscilloscope.
' -----
DoCommand ":AUToscale"

' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
    DoQueryString(":TRIGger:MODE?")

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
Debug.Print "Trigger edge source: " + _
    DoQueryString(":TRIGger:EDGE:SOURce?")

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
    DoQueryString(":TRIGger:EDGE:LEVel?")

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
    DoQueryString(":TRIGger:EDGE:SLOPe?")

' Save oscilloscope configuration.
' -----
varQueryResult = DoQueryIEEEBlock_UI1(":SYSTem:SETup?")

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , varQueryResult ' Write data.
Close hFile ' Close file.
Debug.Print "Setup bytes saved: " + CStr(LenB(varQueryResult))

' Change settings with individual commands:
' -----

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and offset.
DoCommand ":TIMEbase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMEbase:SCALe?")

DoCommand ":TIMEbase:POSItion 0.0"
Debug.Print "Timebase position: " + _

```

```

DoQueryString(":TIMEbase:POSITION?")

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
DoCommand ":ACQuire:TYPE NORMal"
Debug.Print "Acquire type: " + _
DoQueryString(":ACQuire:TYPE?")

' Or, configure by loading a previously saved setup.
' -----
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile    ' Open file for input.
Get hFile, , varSetupString    ' Read data.
Close hFile    ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup"
' command:
DoCommandIEEEBlock ":SYSTem:SETup", varSetupString
Debug.Print "Setup bytes restored: " + CStr(LenB(varSetupString))

' Capture an acquisition using :DIGItize.
' -----
DoCommand ":DIGItize CHANnel1"

Exit Sub

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

End Sub

'
' Analyze the captured waveform.
' -----
Private Sub Analyze()

On Error GoTo VisaComError

' Make a couple of measurements.
' -----
DoCommand ":MEASure:SOURce CHANnel1"
Debug.Print "Measure source: " + _
DoQueryString(":MEASure:SOURce?")

DoCommand ":MEASure:FREQuency"
varQueryResult = DoQueryNumber(":MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
FormatNumber(varQueryResult / 1000, 4) + " kHz"

DoCommand ":MEASure:VAMPplitude"
varQueryResult = DoQueryNumber(":MEASure:VAMPplitude?")
MsgBox "Vertical amplitude:" + vbCrLf + _
FormatNumber(varQueryResult, 4) + " V"

' Download the screen image.

```

```

' -----
' Get screen image.
DoCommand ":HARDcopy:INKSaver OFF"
Dim byteData() As Byte
byteData = DoQueryIEEEBlock_UI1(":DISPLAY:DATA? PNG, COLOR")

' Save screen image to a file.
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
    Kill strPath      ' Remove file if it exists.
End If

Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , byteData      ' Write data.
Close hFile      ' Close file.
MsgBox "Screen image (" + CStr(UBound(byteData) + 1) + _
       " bytes) written to " + strPath

' Download waveform data.
' -----

' Set the waveform points mode.
DoCommand ":WAVeform:POINTs:MODE RAW"
Debug.Print "Waveform points mode: " + _
           DoQueryString(":WAVeform:POINTs:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
           DoQueryString(":WAVeform:POINTs?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
           DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMAT BYTE"
Debug.Print "Waveform format: " + _
           DoQueryString(":WAVeform:FORMAT?")

' Display the waveform settings:
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long

```

```

Preamble() = DoQueryNumbers(":WAVEform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 4 Then
    Debug.Print "Waveform format: ASCII"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMAL"
ElseIf intType = 1 Then
    Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
    Debug.Print "Acquisition type: AVERAGE"
ElseIf intType = 3 Then
    Debug.Print "Acquisition type: HRESOLUTION"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
varQueryResult = DoQueryIEEEBlock_UI1(":WAVEform:DATA?")

```

```

Debug.Print "Number of data values: " + _
CStr(UBound(varQueryResult) + 1)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long
Dim lngI As Long

For lngI = 0 To UBound(varQueryResult)
    lngDataValue = varQueryResult(lngI)

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngDataValue - lngYReference) * _
        sngYIncrement) + sngYOrigin)

    Next lngI

    ' Close output file.
    Close hFile ' Close file.
    MsgBox "Waveform format BYTE data written to " + _
        "c:\scope\data\waveform_data.csv."

    Exit Sub

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

End Sub

Private Sub DoCommand(command As String)

    On Error GoTo VisaComError

    myScope.WriteString command
    CheckInstrumentErrors

    Exit Sub

VisaComError:
    MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
        Err.Source + ", " + _
        Err.Description, vbExclamation, "VISA COM Error"
End

End Sub

Private Sub DoCommandIEEEBlock(command As String, data As Variant)

```

```

On Error GoTo VisaComError

Dim strErrors As String

myScope.WriteLineEEBlock command, data
CheckInstrumentErrors

Exit Sub

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
       Err.Source + ", " + _
       Err.Description, vbExclamation, "VISA COM Error"
End

End Sub

Private Function DoQueryString(query As String) As String

On Error GoTo VisaComError

myScope.WriteString query
DoQueryString = myScope.ReadString
CheckInstrumentErrors

Exit Function

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
       Err.Source + ", " + _
       Err.Description, vbExclamation, "VISA COM Error"
End

End Function

Private Function DoQueryNumber(query As String) As Variant

On Error GoTo VisaComError

myScope.WriteString query
DoQueryNumber = myScope.ReadNumber
CheckInstrumentErrors

Exit Function

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
       Err.Source + ", " + _
       Err.Description, vbExclamation, "VISA COM Error"
End

End Function

Private Function DoQueryNumbers(query As String) As Variant()

On Error GoTo VisaComError

```

```

        Dim strErrors As String

        myScope.WriteString query
        DoQueryNumbers = myScope.ReadList
        CheckInstrumentErrors

        Exit Function

VisaComError:
        MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
            Err.Source + ", " + _
            Err.Description, vbExclamation, "VISA COM Error"
    End

End Function

Private Function DoQueryIEEEBlock_UI1(query As String) As Variant

    On Error GoTo VisaComError

    myScope.WriteString query
    DoQueryIEEEBlock_UI1 = myScope.ReadIEEEBlock(BinaryType_UI1)
    CheckInstrumentErrors

    Exit Function

VisaComError:
    MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
        Err.Source + ", " + _
        Err.Description, vbExclamation, "VISA COM Error"
    End

End Function

Private Sub CheckInstrumentErrors()

    On Error GoTo VisaComError

    Dim strErrVal As String
    Dim strOut As String

    myScope.WriteString ":SYSTem:ERRor?"      ' Query any errors data.
    strErrVal = myScope.ReadString           ' Read: Errnum, "Error String".
    While Val(strErrVal) <> 0              ' End if find: 0, "No Error".
        strOut = strOut + "INST Error: " + strErrVal
        myScope.WriteString ":SYSTem:ERRor?"    ' Request error message.
        strErrVal = myScope.ReadString         ' Read error message.
    Wend

    If Not strOut = "" Then
        MsgBox strOut, vbExclamation, "INST Error Messages"
        myScope.FlushWrite (False)
        myScope.FlushRead
    End If

    Exit Sub

```

```

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

End Sub

```

VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

- 1** Open Visual Studio.
- 2** Create a new Visual C#, Windows, Console Application project.
- 3** Cut-and-paste the code that follows into the C# source file.
- 4** Edit the program to use the VISA address of your oscilloscope.
- 5** Add a reference to the VISA COM 5.5 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 5.5 Type Library**; then click **OK**.
- 6** Build and run the program.

For more information, see the VISA COM Help that comes with Keysight IO Libraries Suite 16.

```

/*
 * Keysight VISA COM Example in C#
 * -----
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * -----
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
    class VisaComInstrumentApp
    {
        private static VisaComInstrument myScope;

        public static void Main(string[] args)
        {
            try
            {
                myScope = new

```

```

        VisaComInstrument ("USB0::0x0957::0x17A6::US50210029::0::INSTR"
) ;
myScope.SetTimeoutSeconds(10);

// Initialize - start from a known state.
Initialize();

// Capture data.
Capture();

// Analyze the captured waveform.
Analyze();
}

catch (System.ApplicationException err)
{
    Console.WriteLine("**** VISA COM Error : " + err.Message);
}
catch (System.SystemException err)
{
    Console.WriteLine("**** System Error Message : " + err.Message);
}
catch (System.Exception err)
{
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("**** Unexpected Error : " + err.Message);
}
finally
{
    myScope.Close();
}
}

/*
 * Initialize the oscilloscope to a known state.
 * -----
 */
private static void Initialize()
{
    string strResults;

    // Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?");
    Console.WriteLine("*IDN? result is: {0}", strResults);

    // Clear status and load the default setup.
    myScope.DoCommand("*CLS");
    myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * -----
 */
private static void Capture()
{
    // Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale");
}

```

```

// Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
myScope.DoCommand(":TRIGger:MODE EDGE");
Console.WriteLine("Trigger mode: {0}",
    myScope.DoQueryString(":TRIGger:MODE?"));

// Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
Console.WriteLine("Trigger edge source: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
Console.WriteLine("Trigger edge level: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
Console.WriteLine("Trigger edge slope: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

// Save oscilloscope configuration.
byte[] ResultsArray; // Results array.
int nLength; // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
nLength = ResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05");
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"));

myScope.DoCommand(":CHANnel1:OFFSet -1.5");
Console.WriteLine("Channel 1 vertical offset: {0}",
    myScope.DoQueryString(":CHANnel1:OFFSet?"));

// Set horizontal scale and offset.
myScope.DoCommand(":TIMEbase:SCALe 0.0002");
Console.WriteLine("Timebase scale: {0}",
    myScope.DoQueryString(":TIMEbase:SCALe?"));

myScope.DoCommand(":TIMEbase:POSition 0.0");
Console.WriteLine("Timebase position: {0}",
    myScope.DoQueryString(":TIMEbase:POSition?"));

// Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESolution
).

```

```

myScope.DoCommand(":ACQuire:TYPE NORMal");
Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACQuire:TYPE?"));

// Or, configure by loading a previously saved setup.
byte[] dataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.stp";
dataArray = File.ReadAllBytes(strPath);
nBytesWritten = dataArray.Length;

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", dataArray);
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

// Capture an acquisition using :DIGItize.
myScope.DoCommand(":DIGItize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * -----
 */
private static void Analyze()
{
    byte[] ResultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Make a couple of measurements.
    // -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1");
    Console.WriteLine("Measure source: {0}",
        myScope.DoQueryString(":MEASure:SOURce?"));

    double fResult;
    myScope.DoCommand(":MEASure:FREQuency");
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

    myScope.DoCommand(":MEASure:VAMplitude");
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?");
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

    // Download the screen image.
    // -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF");

    // Get the screen data.
    ResultsArray =
        myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, COLOR");
    nLength = ResultsArray.Length;

    // Store the screen data to a file.
    strPath = "c:\\scope\\data\\screen.png";
}

```

```

FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// -----
// Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTS:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"));

// Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVEform:POINTS?"));

// Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVEform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII).
myScope.DoCommand(":WAVEform:FORMAT BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVEform:FORMAT?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
    Console.WriteLine("Waveform format: ASCII");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
    Console.WriteLine("Acquire type: NORMAL");
}
else if (fType == 1.0)
{
    Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
}

```

```

        Console.WriteLine("Acquire type: AVERage");
    }
else if (fType == 3.0)
{
    Console.WriteLine("Acquire type: HRESolution");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Waveform X reference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

// Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEFORM:DATA?");
nLength = ResultsArray.Length;
Console.WriteLine("Number of data values: {0}", nLength);

// Set up output file:
strPath = "c:\\scope\\data\\waveform_data.csv";
if (File.Exists(strPath)) File.Delete(strPath);

// Open file for output.
StreamWriter writer = File.CreateText(strPath);

// Output waveform data in CSV format.
for (int i = 0; i < nLength - 1; i++)
    writer.WriteLine("{0:f9}, {1:f6}",
                    fXorigin + ((float)i * fXincrement),
                    (((float)ResultsArray[i] - fYreference)
                     * fYincrement) + fYorigin);

// Close output file.
writer.Close();
Console.WriteLine("Waveform format BYTE data written to {0}",
                 strPath);
}
}

```

```

class VisaComInstrument
{
    private ResourceManagerClass m_ResourceManager;
    private FormattedIO488Class m_IoObject;
    private string m_strVisaAddress;

    // Constructor.
    public VisaComInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA COM IO object.
        OpenIo();

        // Clear the interface.
        m_IoObject.IO.Clear();
    }

    public void DoCommand(string strCommand)
    {
        // Send the command.
        m_IoObject.WriteString(strCommand, true);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public void DoCommandIEEEBlock(string strCommand,
                                  byte[] DataArray)
    {
        // Send the command to the device.
        m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public string DoQueryString(string strQuery)
    {
        // Send the query.
        m_IoObject.WriteString(strQuery, true);

        // Get the result string.
        string strResults;
        strResults = m_IoObject.ReadString();

        // Check for inst errors.
        CheckInstrumentErrors(strQuery);

        // Return results string.
        return strResults;
    }

    public double DoQueryNumber(string strQuery)
    {
        // Send the query.

```

```

m_IoObject.WriteString(strQuery, true);

// Get the result number.
double fResult;
fResult = (double)m_IoObject.ReadNumber(
    IEEEASCIIType.ASCIIType_R8, true);

// Check for inst errors.
CheckInstrumentErrors(strQuery);

// Return result number.
return fResult;
}

public double[] DoQueryNumbers(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result numbers.
    double[] fResultsArray;
    fResultsArray = (double[])m_IoObject.ReadList(
        IEEEASCIIType.ASCIIType_R8, ",;");

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return result numbers.
    return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the results array.
    System.Threading.Thread.Sleep(2000); // Delay before reading.
    byte[] ResultsArray;
    ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
        IEEEBinaryType.BinaryType_UI1, false, true);

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return results array.
    return ResultsArray;
}

private void CheckInstrumentErrors(string strCommand)
{
    // Check for instrument errors.
    string strInstrumentError;
    bool bFirstError = true;

    do // While not "0,No error".
    {

```

```

m_IoObject.WriteString(":SYSTem:ERRor?", true);
strInstrumentError = m_IoObject.ReadString();

if (!strInstrumentError.ToString().StartsWith("+0,"))
{
    if (bFirstError)
    {
        Console.WriteLine("ERROR(s) for command '{0}': ",
                          strCommand);
        bFirstError = false;
    }
    Console.Write(strInstrumentError);
}
} while (!strInstrumentError.ToString().StartsWith("+0,"));
}

private void OpenIo()
{
    m_ResourceManager = new ResourceManagerClass();
    m_IoObject = new FormattedIO488Class();

    // Open the default VISA COM IO object.
    try
    {
        m_IoObject.IO =
            (IMessage)m_ResourceManager.Open(m_strVisaAddress,
                                              AccessMode.NO_LOCK, 0, "");
    }
    catch (Exception e)
    {
        Console.WriteLine("An error occurred: {0}", e.Message);
    }
}

public void SetTimeoutSeconds(int nSeconds)
{
    m_IoObject.IO.Timeout = nSeconds * 1000;
}

public void Close()
{
    try
    {
        m_IoObject.IO.Close();
    }
    catch { }

    try
    {
        Marshal.ReleaseComObject(m_IoObject);
    }
    catch { }

    try
    {
        Marshal.ReleaseComObject(m_ResourceManager);
    }
}

```

```
        catch {  
    }  
}
```

VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

- 1** Open Visual Studio.
 - 2** Create a new Visual Basic, Windows, Console Application project.
 - 3** Cut-and-paste the code that follows into the C# source file.
 - 4** Edit the program to use the VISA address of your oscilloscope.
 - 5** Add a reference to the VISA COM 5.5 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 5.5 Type Library**; then click **OK**.
 - e** Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.
 - 6** Build and run the program.

For more information, see the VISA COM Help that comes with Keysight IO Libraries Suite 16.

```
' Keysight VISA COM Example in Visual Basic .NET
' -----
' This program illustrates a few commonly used programming
' features of your Keysight oscilloscope.
' ----

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace InfiniiVision
    Class VisaComInstrumentApp
        Private Shared myScope As VisaComInstrument

        Public Shared Sub Main(ByVal args As String())
            Try
                myScope = New _
                    VisaComInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR")
            Catch ex As Exception
                MsgBox(ex.Message)
            End Try
        End Sub
    End Class
End Namespace
```

```

)
myScope.SetTimeoutSeconds(10)

' Initialize - start from a known state.
Initialize()

' Capture data.
Capture()

' Analyze the captured waveform.
Analyze()

Catch err As System.ApplicationException
    Console.WriteLine("**** VISA Error Message : " + err.Message)
Catch err As System.SystemException
    Console.WriteLine("**** System Error Message : " + err.Message)
Catch err As System.Exception
    System.Diagnostics.Debug.Fail("Unexpected Error")
    Console.WriteLine("**** Unexpected Error : " + err.Message)
Finally
    myScope.Close()
End Try
End Sub

' Initialize the oscilloscope to a known state.
' -----
Private Shared Sub Initialize()
    Dim strResults As String

    ' Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?")
    Console.WriteLine("*IDN? result is: {0}", strResults)

    ' Clear status and load the default setup.
    myScope.DoCommand("*CLS")
    myScope.DoCommand("*RST")

End Sub

' Capture the waveform.
' -----
Private Shared Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale")

    ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE")
    Console.WriteLine("Trigger mode: {0}", _
        myScope.DoQueryString(":TRIGger:MODE?"))

    ' Set EDGE trigger parameters.
    myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
    Console.WriteLine("Trigger edge source: {0}", _
        myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

```

```

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()      ' Results array.
Dim nLength As Integer         ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMEbase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMEbase:SCALe?"))

myScope.DoCommand(":TIMEbase:POSition 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMEbase:POSition?"))

' Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESolution)

myScope.DoCommand(":ACQuire:TYPE NORMAL")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim DataArray As Byte()
Dim nBytesWritten As Integer

' Read setup string from file.

```

```

strPath = "c:\scope\config\setup.stp"
DataArray = File.ReadAllBytes(strPath)
nBytesWritten = DataArray.Length

' Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETUp", DataArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGItize.
myScope.DoCommand(":DIGItize CHANnel1")

End Sub

' Analyze the captured waveform.
' ----

Private Shared Sub Analyze()

Dim fResult As Double
Dim ResultsArray As Byte()      ' Results array.
Dim nLength As Integer         ' Number of bytes returned from inst.
Dim strPath As String

' Make a couple of measurements.
' -----
myScope.DoCommand(":MEASure:SOURce CHANnel1")
Console.WriteLine("Measure source: {0}", _
    myScope.DoQueryString(":MEASure:SOURce?"))

myScope.DoCommand(":MEASure:FREQuency")
fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

myScope.DoCommand(":MEASure:VAMplitude")
fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?")
Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

' Download the screen image.
' -----
myScope.DoCommand(":HARDcopy:INKSaver OFF")

' Get the screen data.
ResultsArray = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLOR")
nLength = ResultsArray.Length

' Store the screen data to a file.
strPath = "c:\scope\data\screen.png"
Dim fStream As FileStream
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' -----

```

```

' Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTS:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"))

' Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVEform:POINTS?"))

' Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVEform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMAT BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVEform:FORMAT?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
    Console.WriteLine("Waveform format: ASCII")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
    Console.WriteLine("Acquire type: NORMAL")
ElseIf fType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
    Console.WriteLine("Acquire type: AVERAGE")
ElseIf fType = 3 Then
    Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fxincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fxincrement)

Dim fxorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fxorigin)

Dim fxreference As Double = fResultsArray(6)

```

```

Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEFORM:DATA?")
nLength = ResultsArray.Length
Console.WriteLine("Number of data values: {0}", nLength)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"
If File.Exists(strPath) Then
    File.Delete(strPath)
End If

' Open file for output.
Dim writer As StreamWriter = File.CreateText(strPath)

' Output waveform data in CSV format.
For index As Integer = 0 To nLength - 1
    ' Write time value, voltage value.
    writer.WriteLine("{0:f9}, {1:f6}", _
                    fXorigin + (CSng(index) * fXincrement), _
                    ((CSng(ResultsArray(index)) - fYreference) * fYincrement) + fYorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", strPath)

End Sub

End Class

Class VisaComInstrument
    Private m_ResourceManager As ResourceManagerClass
    Private m_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()
    End Sub
End Class

```

```

' Clear the interface.
m_IoObject.IO.Clear()

End Sub

Public Sub DoCommand(ByVal strCommand As String)

    ' Send the command.
    m_IoObject.WriteString(strCommand, True)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

End Sub

Public Sub DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte())

    ' Send the command to the device.
    m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result string.
    Dim strResults As String
    strResults = m_IoObject.ReadString()

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return results string.
    Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result number.
    Dim fResult As Double
    fResult = _
        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(ByVal strQuery As String) As Byte()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the results array.
    System.Threading.Thread.Sleep(2000) ' Delay before reading data.
    Dim ResultsArray As Byte()
    ResultsArray = _
        m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
        False, True)

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return results array.
    Return ResultsArray
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
    ' Check for instrument errors.
    Dim strInstrumentError As String
    Dim bFirstError As Boolean = True
    Do      ' While not "0,No error".
        m_IoObject.WriteString(":SYSTem:ERRor?", True)
        strInstrumentError = m_IoObject.ReadString()

        If Not strInstrumentError.ToString().StartsWith("+0,") Then
            If bFirstError Then
                Console.WriteLine("ERROR(s) for command '{0}': {1}, _", _
                    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_Manager.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_Manager)
    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 Keysight 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
```

```

#
# Keysight VISA COM Example in Python using "comtypes"
# ****
# This program illustrates a few commonly used programming
# features of your Keysight 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():

    # 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:EDGE:LEVel 1.5")
qresult = do_query_string(":TRIGger:EDGE:LEVel?")
print "Trigger edge level: %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(":SYSTem:SETUp?")
nLength = len(setup_bytes)
f = open("c:\scope\config\setup.stp", "wb")
f.write(bytarray(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 0.05")
qresult = do_query_number(":CHANnel1:SCALe?")
print "Channel 1 vertical scale: %f" % qresult

do_command(":CHANnel1:OFFSet -1.5")
qresult = do_query_number(":CHANnel1:OFFSet?")
print "Channel 1 offset: %f" % qresult

# Set horizontal scale and offset.
do_command(":TIMEbase:SCALe 0.0002")
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 type.
do_command(":ACQuire:TYPE NORMAL")
qresult = do_query_string(":ACQuire:TYPE?")
print "Acquire type: %s" % qresult

# Or, configure by loading a previously saved setup.
f = open("c:\scope\config\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)

# Capture an acquisition using :DIGItize.
do_command(":DIGItize CHANnel1")

# =====

```

```

# 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:VAMPplitude")
    qresult = do_query_string(":MEASure:VAMPplitude?")
    print "Measured vertical amplitude on channel 1: %s" % qresult

    # Download the screen image.
    # -----
    do_command(":HARDcopy:INKSaver OFF")

    image_bytes = do_query_ieee_block(":DISPLAY:DATA? PNG, COLOR")
    nLength = len(image_bytes)
    f = open("c:\scope\data\screen.png", "wb")
    f.write(bytarray(image_bytes))
    f.close()
    print "Screen image written to c:\scope\data\screen.png."

    # Download waveform data.
    # ----

    # Set the waveform points mode.
    do_command(":WAVeform:POINTS:MODE RAW")
    qresult = do_query_string(":WAVeform:POINTS:MODE?")
    print "Waveform points mode: %s" % qresult

    # Get the number of waveform points available.
    do_command(":WAVeform:POINTS 10240")
    qresult = do_query_string(":WAVeform:POINTS?")
    print "Waveform points available: %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 : "BYTE",
        1 : "WORD",
        4 : "ASCII",
    }

```

```

acq_type_dict = {
    0 : "NORMAl",
    1 : "PEAK",
    2 : "AVERage",
    3 : "HRESolution",
}
(
    wav_form,
    acq_type,
    wfmpts,
    avgcnt,
    x_increment,
    x_origin,
    x_reference,
    y_increment,
    y_origin,
    y_reference
) = do_query_numbers(":WAVEform:PREamble?")

print "Waveform format: %s" % wav_form_dict[wav_form]
print "Acquire type: %s" % acq_type_dict[acq_type]
print "Waveform points desired: %d" % wfmpts
print "Waveform average count: %d" % avgcnt
print "Waveform X increment: %1.12f" % x_increment
print "Waveform X origin: %1.9f" % x_origin
print "Waveform X reference: %d" % x_reference # Always 0.
print "Waveform Y increment: %f" % y_increment
print "Waveform Y origin: %f" % y_origin
print "Waveform Y reference: %d" % y_reference # Always 125.

# 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?")
y_reference = do_query_number(":WAVEform:YREFerence?")

# Get the waveform data.
data_bytes = do_query_ieee_block(":WAVEform:DATA?")
nLength = len(data_bytes)
print "Number of data values: %d" % nLength

# Open file for output.
strPath = "c:\scope\data\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_bytes[i] - y_reference) * y_increment + y_origin
    f.write("%E, %f\n" % (time_val, voltage))

# Close output file.
f.close()
print "Waveform format BYTE 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(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 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?", True)
        error_string = myScope.ReadString()
        if error_string:    # If there is an error string value.

            if error_string.find("+0," , 0, 3) == -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? should always return string.
            print "ERROR: :SYSTem:ERRor? 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("TCPIPO::a-mx3104a-90028.cos.is.keysight.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 837
- "VISA Example in Visual Basic" on page 846
- "VISA Example in C#" on page 856
- "VISA Example in Visual Basic .NET" on page 867
- "VISA Example in Python" on page 877

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.

```
/*
 * Keysight VISA Example in C
 * -----
 */
```

```

* This program illustrates a few commonly-used programming
* features of your Keysight oscilloscope.
*/

#include <stdio.h>           /* For printf(). */
#include <string.h>          /* For strcpy(), strcat(). */
#include <time.h>            /* For clock(). */
#include <visa.h>             /* Keysight VISA routines. */

#define VISA_ADDRESS "USB0::0x0957::0x17A6::US50210029::0::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 block. */
void check_instrument_errors();   /* Check for inst errors. */
void error_handler();            /* VISA error handler. */

/* Global variables */
ViSession defaultRM, vi;         /* Device session ID. */
ViStatus err;                   /* VISA function return value. */
char str_result[256] = {0};       /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE]; /* Result from
                                               do_query_ieeeblock(). */
double dbl_results[10];          /* Result from do_query_numbers(). */

/* Main Program
 * -----
void main(void)
{
    /* Open the default resource manager session. */
    err = viOpenDefaultRM(&defaultRM);
    if (err != VI_SUCCESS) error_handler();

    /* Open the session using the oscilloscope's VISA address. */
    err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL, VI_NULL, &vi);
    if (err != VI_SUCCESS) error_handler();

    /* Set the I/O timeout to fifteen seconds. */
    err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 15000);
    if (err != VI_SUCCESS) error_handler();

    /* Initialize - start from a known state. */
    initialize();

    /* Capture data. */
    capture();
}

```

```

/* Analyze the captured waveform. */
analyze();

/* Close the vi session and the resource manager session. */
viClose(vi);
viClose(defaultRM);
}

/* Initialize the oscilloscope to a known state.
 * -----
void initialize (void)
{
    /* Clear the interface. */
    err = viClear(vi);
    if (err != VI_SUCCESS) error_handler();

    /* Get and display the device's *IDN? string. */
    do_query_string("*IDN?");
    printf("Oscilloscope *IDN? string: %s\n", str_result);

    /* Clear status and load the default setup. */
    do_command("*CLS");
    do_command("*RST");
}

/* Capture the waveform.
 * -----
void capture (void)
{
    int num_bytes;
    FILE *fp;

    /* Use auto-scale to automatically configure oscilloscope. */
    do_command(":AUToscale");

    /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. */
    do_command(":TRIGger:MODE EDGE");
    do_query_string(":TRIGger:MODE?");
    printf("Trigger mode: %s\n", str_result);

    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURce CHANnel1");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);

    do_command(":TRIGger:EDGE:LEVel 1.5");
    do_query_string(":TRIGger:EDGE:LEVel?");
    printf("Trigger edge level: %s\n", str_result);

    do_command(":TRIGger:EDGE:SLOPe POSitive");
    do_query_string(":TRIGger:EDGE:SLOPe?");
    printf("Trigger edge slope: %s\n", str_result);

    /* Save oscilloscope configuration. */

    /* Read system setup. */
    num_bytes = do_query_ieeeblock(":SYSTem:SETUp?");
}

```

```

printf("Read setup string query (%d bytes).\\n", num_bytes);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\\n");

/* Change settings with individual commands:

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.05");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\\n", str_result);

/* Set horizontal scale and offset. */
do_command(":TIMEbase:SCALe 0.0002");
do_query_string(":TIMEbase:SCALe?");
printf("Timebase scale: %s\\n", str_result);

do_command(":TIMEbase:POSItion 0.0");
do_query_string(":TIMEbase:POSItion?");
printf("Timebase position: %s\\n", str_result);

/* Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution). */
/
do_command(":ACQuire:TYPE NORMal");
do_query_string(":ACQuire:TYPE?");
printf("Acquire type: %s\\n", str_result);

/* Or, configure by loading a previously saved setup. */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\\n");

/* Restore setup string. */
num_bytes = do_command_ieeeblock(":SYSTem:SETUp", num_bytes);
printf("Restored setup string (%d bytes).\\n", num_bytes);

/* Capture an acquisition using :DIGItize. */
do_command(":DIGItize CHANnel1");
}

/* Analyze the captured waveform.
* -----
void analyze (void)

```

```

{
    double wav_format;
    double acq_type;
    double wav_points;
    double avg_count;
    double x_increment;
    double x_origin;
    double x_reference;
    double y_increment;
    double y_origin;
    double y_reference;

    FILE *fp;
    int num_bytes; /* Number of bytes returned from instrument. */
    int i;

    /* Make a couple of measurements.
     * -----
     do_command(":MEASure:SOURce CHANnel1");
     do_query_string(":MEASure:SOURce?");
     printf("Measure source: %s\n", str_result);

     do_command(":MEASure:FREQuency");
     do_query_number(":MEASure:FREQuency?");
     printf("Frequency: %.4f kHz\n", num_result / 1000);

     do_command(":MEASure:VAMPplitude");
     do_query_number(":MEASure:VAMPplitude?");
     printf("Vertical amplitude: %.2f V\n", num_result);

    /* Download the screen image.
     * -----
     do_command(":HARDcopy:INKSaver OFF");

    /* Read screen image. */
    num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG, COLOR");
    printf("Screen image bytes: %d\n", num_bytes);

    /* Write screen image bytes to file. */
    fp = fopen ("c:\\scope\\data\\screen.png", "wb");
    num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
                      fp);
    fclose (fp);
    printf("Wrote screen image (%d bytes) to ", num_bytes);
    printf("c:\\scope\\data\\screen.png.\n");

    /* Download waveform data.
     * -----
     */

    /* Set the waveform points mode. */
    do_command(":WAVeform:POINTS:MODE RAW");
    do_query_string(":WAVeform:POINTS:MODE?");
    printf("Waveform points mode: %s\n", str_result);

    /* Get the number of waveform points available. */
    do_query_string(":WAVeform:POINTS?");
    printf("Waveform points available: %s\n", str_result);
}

```

```

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII) : */
do_command(":WAVeform:FORMAT BYTE");
do_query_string(":WAVeform:FORMAT?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
    printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
    printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
    printf("Waveform format: ASCII\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
    printf("Acquire type: NORMAL\n");
}
else if (acq_type == 1.0)
{
    printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
    printf("Acquire type: AVERage\n");
}
else if (acq_type == 3.0)
{
    printf("Acquire type: HRESolution\n");
}

wav_points = dbl_results[2];
printf("Waveform points: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);

x_origin = dbl_results[5];
printf("Waveform X origin: %e\n", x_origin);

```

```

x_reference = dbl_results[6];
printf("Waveform X reference: %e\n", x_reference);

y_increment = dbl_results[7];
printf("Waveform Y increment: %e\n", y_increment);

y_origin = dbl_results[8];
printf("Waveform Y origin: %e\n", y_origin);

y_reference = dbl_results[9];
printf("Waveform Y reference: %e\n", y_reference);

/* Read waveform data. */
num_bytes = do_query_ieeeblock(":WAVEform:DATA?");
printf("Number of data values: %d\n", num_bytes);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

/* Output waveform data in CSV format. */
for (i = 0; i < num_bytes - 1; i++)
{
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
            x_origin + ((float)i * x_increment),
            (((float)ieeeblock_data[i] - y_reference) * y_increment)
            + y_origin);
}

/* Close output file. */
fclose(fp);
printf("Waveform format BYTE data written to ");
printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * -----
void do_command(command)
char *command;
{
    char message[80];

    strcpy(message, command);
    strcat(message, "\n");
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * -----
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{

```

```

char message[80];
int data_length;

strcpy(message, command);
strcat(message, " #8%08d");
err = viPrintf(vi, message, num_bytes);
if (err != VI_SUCCESS) error_handler();

err = viBufWrite(vi, ieeeblock_data, num_bytes, &data_length);
if (err != VI_SUCCESS) error_handler();

check_instrument_errors();

return(data_length);
}

/* Query for a string result.
 * -----
void do_query_string(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%t", str_result);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for a number result.
 * -----
void do_query_number(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%lf", &num_result);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for numbers result.
 * -----
void do_query_numbers(query)

```

```

char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%,10lf\n", dbl_results);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * -----
int do_query_ieeeblock(query)
char *query;
{
    char message[80];
    int data_length;

    strcpy(message, query);
    strcat(message, "\n");
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    data_length = IEEEBLOCK_SPACE;
    err = viScanf(vi, "%#b\n", &data_length, ieeeblock_data);
    if (err != VI_SUCCESS) error_handler();

    if (data_length == IEEEBLOCK_SPACE )
    {
        printf("IEEE block buffer full: ");
        printf("May not have received all data.\n");
    }

    check_instrument_errors();

    return(data_length);
}

/* Check for instrument errors.
 * -----
void check_instrument_errors()
{
    char str_err_val[256] = {0};
    char str_out[800] = "";

    err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
    while(strncmp(str_err_val, "+0,No error", 3) != 0 )
    {
        strcat(str_out, ", ");
        strcat(str_out, str_err_val);
}

```

```

        err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
        if (err != VI_SUCCESS) error_handler();
    }

    if (strcmp(str_out, "") != 0)
    {
        printf("INST Error%s\n", str_out);
        err = viFlush(vi, VI_READ_BUF);
        if (err != VI_SUCCESS) error_handler();
        err = viFlush(vi, VI_WRITE_BUF);
        if (err != VI_SUCCESS) error_handler();
    }
}

/* Handle VISA errors.
 * -----
void error_handler()
{
    char err_msg[1024] = {0};

    viStatusDesc(vi, err, err_msg);
    printf("VISA Error: %s\n", err_msg);
    if (err < VI_SUCCESS)
    {
        exit(1);
    }
}

```

VISA Example in Visual Basic

To run this example in Visual Basic for Applications:

- 1** Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2** Press ALT+F11 to launch the Visual Basic editor.
- 3** Add the visa32.bas file to your project:
 - a** Choose **File > Import File....**
 - b** Navigate to the header file, visa32.bas (installed with Keysight 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.

```

'
' Keysight VISA Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming

```

```

' features of your Keysight oscilloscope.
' -----
Option Explicit

Public err As Long      ' Error returned by VISA function calls.
Public drm As Long      ' Session to Default Resource Manager.
Public vi As Long        ' Session to instrument.

' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long
Public Const DblArraySize = 20
Public dblArray(DblArraySize) As Double

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' -----
Sub Main()

    ' Open the default resource manager session.
    err = viOpenDefaultRM(drm)
    If (err <> VI_SUCCESS) Then HandleVISAError drm

    ' Open the session using the oscilloscope's VISA address.
    err = viOpen(drm, _
                "USB0::0x0957::0x17A6::US50210029::0::INSTR", 0, 15000, vi)
    If (err <> VI_SUCCESS) Then HandleVISAError drm

    ' Set the I/O timeout to ten seconds.
    err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 10000)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    ' Initialize - start from a known state.
    Initialize

    ' Capture data.
    Capture

    ' Analyze the captured waveform.
    Analyze

    ' Close the vi session and the resource manager session.
    err = viClose(vi)
    err = viClose(drm)

```

```

End Sub

'

' Initialize the oscilloscope to a known state.
' -----
Private Sub Initialize()

    ' Clear the interface.
    err = viClear(vi)
    If Not (err = VI_SUCCESS) Then HandleVISAError vi

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    MsgBox "*IDN? string: " + strQueryResult, vbOKOnly, "*IDN? Result"

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

End Sub

'

' Capture the waveform.
' -----
Private Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    ' -----
    DoCommand ":AUToscale"

    ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
    DoCommand ":TRIGger:MODE EDGE"
    Debug.Print "Trigger mode: " + _
        DoQueryString(":TRIGger:MODE?")

    ' Set EDGE trigger parameters.
    DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
    Debug.Print "Trigger edge source: " + _
        DoQueryString(":TRIGger:EDGE:SOURCe?")

    DoCommand ":TRIGger:EDGE:LEVel 1.5"
    Debug.Print "Trigger edge level: " + _
        DoQueryString(":TRIGger:EDGE:LEVel?")

    DoCommand ":TRIGger:EDGE:SLOPe POSitive"
    Debug.Print "Trigger edge slope: " + _
        DoQueryString(":TRIGger:EDGE:SLOPe?")

    ' Save oscilloscope configuration.
    ' -----
    Dim lngSetupStringSize As Long
    lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYStem:SETup?")
    Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

```

```

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
If Len(Dir(strPath)) Then
    Kill strPath      ' Remove file if it exists.
End If

' Open file for output.
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngSetupStringSize - 1
    Put hFile, , byteArray(lngI)      ' Write data.
Next lngI
Close hFile      ' Close file.

' Change settings with individual commands:
' -----
' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and position.
DoCommand ":TIMEbase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMEbase:SCALe?")

DoCommand ":TIMEbase:POSIon 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMEbase:POSIon?")

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
DoCommand ":ACQuire:TYPE NORMal"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQuire:TYPE?")

' Or, configure by loading a previously saved setup.
' -----
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile      ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile)      ' Length of file.
Get hFile, , byteArray      ' Read data.
Close hFile      ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup"
' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
Debug.Print "Setup bytes restored: " + CStr(lngRestored)

```

```

' Capture an acquisition using :DIGItize.
' -----
DoCommand ":DIGItize CHANnel1"

End Sub

'
' Analyze the captured waveform.
' -----

Private Sub Analyze()

' Make a couple of measurements.
' -----
DoCommand ":MEASure:SOURce CHANnel1"
Debug.Print "Measure source: " + _
    DoQueryString(":MEASure:SOURce?")

DoCommand ":MEASure:FREQuency"
dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(dblQueryResult / 1000, 4) + " kHz"

DoCommand ":MEASure:VAMplitude"
dblQueryResult = DoQueryNumber(":MEASure:VAMplitude?")
MsgBox "Vertical amplitude:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

'
' Download the screen image.
' -----
DoCommand ":HARDcopy:INKSaver OFF"

' Get screen image.
Dim lngBlockSize As Long
lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, COLOR")
Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

' Save screen image to a file:
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
    Kill strPath      ' Remove file if it exists.
End If
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)      ' Write data.
Next lngI
Close hFile      ' Close file.
MsgBox "Screen image written to " + strPath

'
' Download waveform data.
' -----

```

```

' Set the waveform points mode.
DoCommand ":WAVeform:POINTs:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINTs:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINTs?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMAT BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMAT?")

' Display the waveform settings:
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim lngYOrigin As Long
Dim lngYReference As Long
Dim strOutput As String

Dim lngNumNumbers As Long
lngNumNumbers = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
lngYOrigin = dblArray(8)
lngYReference = dblArray(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
    Debug.Print "Waveform format: ASCII"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMAL"

```

```

        ElseIf intType = 1 Then
            Debug.Print "Acquisition type: PEAK"
        ElseIf intType = 2 Then
            Debug.Print "Acquisition type: AVERage"
        ElseIf intType = 3 Then
            Debug.Print "Acquisition type: HRESolution"
        End If

        Debug.Print "Waveform points: " + _
                    FormatNumber(lngPoints, 0)

        Debug.Print "Waveform average count: " + _
                    FormatNumber(lngCount, 0)

        Debug.Print "Waveform X increment: " + _
                    Format(dblXIncrement, "Scientific")

        Debug.Print "Waveform X origin: " + _
                    Format(dblXOrigin, "Scientific")

        Debug.Print "Waveform X reference: " + _
                    FormatNumber(lngXReference, 0)

        Debug.Print "Waveform Y increment: " + _
                    Format(sngYIncrement, "Scientific")

        Debug.Print "Waveform Y origin: " + _
                    FormatNumber(lngYOrigin, 0)

        Debug.Print "Waveform Y reference: " + _
                    FormatNumber(lngYReference, 0)

        ' Get the waveform data
        Dim lngNumBytes As Long
        lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVEform:DATA?")
        Debug.Print "Number of data values: " + CStr(lngNumBytes)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"

        ' Open file for output.
        Open strPath For Output Access Write Lock Write As hFile

        ' Output waveform data in CSV format.
        Dim lngDataValue As Long

        For lngI = 0 To lngNumBytes - 1
            lngDataValue = CLng(byteArray(lngI))

            ' Write time value, voltage value.
            Print #hFile, _
                FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
                ", " + _
                FormatNumber(((lngDataValue - lngYReference) * sngYIncrement) + lngYOrigin)

        Next lngI
    End Sub
End Class

```

```

' Close output file.
Close hFile    ' Close file.
MsgBox "Waveform format BYTE data written to " + _
       "c:\scope\data\waveform_data.csv."

End Sub

Private Sub DoCommand(command As String)

    err = viVPrintf(vi, command + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    CheckInstrumentErrors

End Sub

Private Function DoCommandIEEEBlock(command As String, _
                                      lngBlockSize As Long)

    retCount = lngBlockSize

    Dim strCommandAndLength As String
    strCommandAndLength = command + " %#" + _
                           Format(lngBlockSize) + "b"

    err = viVPrintf(vi, strCommandAndLength + vbLf, paramsArray(1))
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    DoCommandIEEEBlock = retCount

    CheckInstrumentErrors

End Function

Private Function DoQueryString(query As String) As String

    Dim strResult As String * 200

    err = viVPrintf(vi, query + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strResult)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    DoQueryString = strResult

    CheckInstrumentErrors

End Function

Private Function DoQueryNumber(query As String) As Variant

    Dim dblResult As Double

    err = viVPrintf(vi, query + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

```

```

err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblResult))
If (err <> VI_SUCCESS) Then HandleVISAError vi

DoQueryNumber = dblResult

CheckInstrumentErrors

End Function

Private Function DoQueryNumbers(query As String) As Long

Dim dblResult As Double

' Send query.
err = viVPrintf(vi, query + vbLf, 0)
If (err <> VI_SUCCESS) Then HandleVISAError vi

' Set up paramsArray for multiple parameter query returning array.
paramsArray(0) = VarPtr(retCount)
paramsArray(1) = VarPtr(dblArray(0))

' Set retCount to max number of elements array can hold.
retCount = DblArraySize

' Read numbers.
err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
If (err <> VI_SUCCESS) Then HandleVISAError vi

' retCount is now actual number of values returned by query.
DoQueryNumbers = retCount

CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

' Send query.
err = viVPrintf(vi, query + vbLf, 0)
If (err <> VI_SUCCESS) Then HandleVISAError vi

' Set up paramsArray for multiple parameter query returning array.
paramsArray(0) = VarPtr(retCount)
paramsArray(1) = VarPtr(byteArray(0))

' Set retCount to max number of elements array can hold.
retCount = ByteArraySize

' Get unsigned integer bytes.
err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
If (err <> VI_SUCCESS) Then HandleVISAError vi

err = viFlush(vi, VI_READ_BUF)
If (err <> VI_SUCCESS) Then HandleVISAError vi

err = viFlush(vi, VI_WRITE_BUF)

```

```

If (err <> VI_SUCCESS) Then HandleVISAError vi

' retCount is now actual number of bytes returned by query.
DoQueryIEEEBlock_Bytes = retCount

CheckInstrumentErrors

End Function

Private Sub CheckInstrumentErrors()

On Error GoTo ErrorHandler

Dim strErrVal As String * 200
Dim strOut As String

err = viVPrintf(vi, ":SYSTem:ERRor?" + vbLf, 0)      ' Query any errors.
If (err <> VI_SUCCESS) Then HandleVISAError vi

err = viVScanf(vi, "%t", strErrVal)      ' Read: Errnum,"Error String".
If (err <> VI_SUCCESS) Then HandleVISAError vi

While Val(strErrVal) <> 0                  ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal

    err = viVPrintf(vi, ":SYSTem:ERRor?" + vbLf, 0)      ' Request error.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strErrVal)      ' Read error message.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

Wend

If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"

    err = viFlush(vi, VI_READ_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viFlush(vi, VI_WRITE_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

End If

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Sub HandleVISAError(session As Long)

Dim strVisaErr As String * 200
Call viStatusDesc(session, err, strVisaErr)

```

```

    MsgBox "**** VISA Error : " + strVisaErr, vbExclamation

    ' If the error is not a warning, close the session.
    If err < VI_SUCCESS Then
        If session <> 0 Then Call viClose(session)
        End
    End If

End Sub

```

VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

- 1** Open Visual Studio.
 - 2** Create a new Visual C#, Windows, Console Application project.
 - 3** Cut-and-paste the code that follows into the C# source file.
 - 4** Edit the program to use the VISA address of your oscilloscope.
 - 5** Add Keysight'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 Keysight 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 Keysight IO Libraries Suite 16.

```

/*
 * Keysight VISA Example in C#
 * -----
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * -----
 */

using System;
using System.IO;
using System.Text;

namespace InfiniiVision
{
    class VisaInstrumentApp

```

```

{
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
        try
        {
            myScope = new
                VisaInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR");
            myScope.SetTimeoutSeconds(10);

            // Initialize - start from a known state.
            Initialize();

            // Capture data.
            Capture();

            // Analyze the captured waveform.
            Analyze();
        }
        catch (System.ApplicationException err)
        {
            Console.WriteLine("*** VISA Error Message : " + err.Message);
        }
        catch (System.SystemException err)
        {
            Console.WriteLine("*** System Error Message : " + err.Message);
        }
        catch (System.Exception err)
        {
            System.Diagnostics.Debug.Fail("Unexpected Error");
            Console.WriteLine("*** Unexpected Error : " + err.Message);
        }
        finally
        {
            myScope.Close();
        }
    }

/*
 * Initialize the oscilloscope to a known state.
 * -----
 */
private static void Initialize()
{
    StringBuilder strResults;

    // Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?");
    Console.WriteLine("*IDN? result is: {0}", strResults);

    // Clear status and load the default setup.
    myScope.DoCommand("*CLS");
    myScope.DoCommand("*RST");
}

```

```

/*
 * Capture the waveform.
 * -----
 */
private static void Capture()
{
    // Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale");

    // Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE");
    Console.WriteLine("Trigger mode: {0}",
        myScope.DoQueryString(":TRIGger:MODE?"));

    // Set EDGE trigger parameters.
    myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
    Console.WriteLine("Trigger edge source: {0}",
        myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

    myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
    Console.WriteLine("Trigger edge level: {0}",
        myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

    myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
    Console.WriteLine("Trigger edge slope: {0}",
        myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

    // Save oscilloscope configuration.
    byte[] ResultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Query and read setup string.
    nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETUp?",
        out ResultsArray);

    // Write setup string to file.
    strPath = "c:\\scope\\config\\setup.stp";
    FileStream fStream = File.Open(strPath, FileMode.Create);
    fStream.Write(ResultsArray, 0, nLength);
    fStream.Close();
    Console.WriteLine("Setup bytes saved: {0}", nLength);

    // Change settings with individual commands:

    // Set vertical scale and offset.
    myScope.DoCommand(":CHANnel1:SCALe 0.05");
    Console.WriteLine("Channel 1 vertical scale: {0}",
        myScope.DoQueryString(":CHANnel1:SCALe?"));

    myScope.DoCommand(":CHANnel1:OFFSet -1.5");
    Console.WriteLine("Channel 1 vertical offset: {0}",
        myScope.DoQueryString(":CHANnel1:OFFSet?"));

    // Set horizontal scale and position.
    myScope.DoCommand(":TIMEbase:SCALe 0.0002");
    Console.WriteLine("Timebase scale: {0}",

```

```

    myScope.DoQueryString(":TIMEbase:SCALe?");

    myScope.DoCommand(":TIMEbase:POSIon 0.0");
    Console.WriteLine("Timebase position: {0}",
                      myScope.DoQueryString(":TIMEbase:POSIon?"));

    // Set the acquisition type (NORMAL, PEAK, AVERage, or HRESolution
    ).
    myScope.DoCommand(":ACQuire:TYPE NORMAL");
    Console.WriteLine("Acquire type: {0}",
                      myScope.DoQueryString(":ACQuire:TYPE?"));

    // Or, configure by loading a previously saved setup.
    byte[] dataArray;
    int nBytesWritten;

    // Read setup string from file.
    strPath = "c:\\scope\\config\\setup.stp";
    dataArray = File.ReadAllBytes(strPath);

    // Restore setup string.
    nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
                                              dataArray);
    Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

    // Capture an acquisition using :DIGItize.
    myScope.DoCommand(":DIGItize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * -----
 */
private static void Analyze()
{
    byte[] ResultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Make a couple of measurements.
    // -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1");
    Console.WriteLine("Measure source: {0}",
                      myScope.DoQueryString(":MEASure:SOURce?"));

    double fResult;
    myScope.DoCommand(":MEASure:FREQuency");
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

    myScope.DoCommand(":MEASure:VAMplitude");
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?");
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

    // Download the screen image.
    // -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF");
}

```

```

// Get the screen data.
nLength = myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, COLOR",
    out ResultsArray);

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ----

// Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTS:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"));

// Get the number of waveform points available.
myScope.DoCommand(":WAVEform:POINTS 10240");
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVEform:POINTS?"));

// Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVEform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMAT BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVEform:FORMAT?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
    Console.WriteLine("Waveform format: ASCII");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{

```

```

        Console.WriteLine("Acquire type: NORMal");
    }
    else if (fType == 1.0)
    {
        Console.WriteLine("Acquire type: PEAK");
    }
    else if (fType == 2.0)
    {
        Console.WriteLine("Acquire type: AVERage");
    }
    else if (fType == 3.0)
    {
        Console.WriteLine("Acquire type: HRESolution");
    }

    double fPoints = fResultsArray[2];
    Console.WriteLine("Waveform points: {0:e}", fPoints);

    double fCount = fResultsArray[3];
    Console.WriteLine("Waveform average count: {0:e}", fCount);

    double fXincrement = fResultsArray[4];
    Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

    double fXorigin = fResultsArray[5];
    Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

    double fXreference = fResultsArray[6];
    Console.WriteLine("Waveform X reference: {0:e}", fXreference);

    double fYincrement = fResultsArray[7];
    Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

    double fYorigin = fResultsArray[8];
    Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

    double fYreference = fResultsArray[9];
    Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

    // Read waveform data.
    nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?",
        out ResultsArray);
    Console.WriteLine("Number of data values: {0}", nLength);

    // Set up output file.
    strPath = "c:\\scope\\data\\waveform_data.csv";
    if (File.Exists(strPath)) File.Delete(strPath);

    // Open file for output.
    StreamWriter writer = File.CreateText(strPath);

    // Output waveform data in CSV format.
    for (int i = 0; i < nLength - 1; i++)
        writer.WriteLine("{0:f9}, {1:f6}",
            fXorigin + ((float)i * fXincrement),
            (((float)ResultsArray[i] - fYreference) *
            fYincrement) + fYorigin);
}

```

```

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format BYTE data written to {0}",
                          strPath);
    }
}

class VisaInstrument
{
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA resource manager.
        OpenResourceManager();

        // Open a VISA resource session.
        OpenSession();

        // Clear the interface.
        int nViStatus;
        nViStatus = visa32.viClear(m_nSession);
    }

    public void DoCommand(string strCommand)
    {
        // Send the command.
        VisaSendCommandOrQuery(strCommand);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public int DoCommandIEEEBlock(string strCommand,
                                 byte[] dataArray)
    {
        // Send the command to the device.
        string strCommandAndLength;
        int nViStatus, nLength, nBytesWritten;

        nLength = dataArray.Length;
        strCommandAndLength = String.Format("{0} #8%08d",
                                            strCommand);

        // Write first part of command to formatted I/O write buffer.
        nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,
                                    nLength);
        CheckVisaStatus(nViStatus);

        // Write the data to the formatted I/O write buffer.
    }
}

```

```

nViStatus = visa32.viBufWrite(m_nSession, dataArray, nLength,
    out nBytesWritten);
CheckVisaStatus(nViStatus);

// Check for inst errors.
CheckInstrumentErrors(strCommand);

return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    StringBuilder strResults = new StringBuilder(1000);
    strResults = VisaGetResultString();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
}

public double DoQueryNumber(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultNumber();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
}

public double[] DoQueryNumbers(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double[] fResultsArray;
    fResultsArray = VisaGetResultNumbers();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return fResultsArray;
}

```

```

public int DoQueryIEEEBlock(string strQuery,
    out byte[] ResultsArray)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    int length;    // Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(out ResultsArray);

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return length;
}

private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
    // Send command or query to the device.
    string strWithNewline;
    strWithNewline = String.Format("{0}\n", strCommandOrQuery);
    int nViStatus;
    nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
    CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetString()
{
    StringBuilder strResults = new StringBuilder(1000);

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
    CheckVisaStatus(nViStatus);

    return strResults;
}

private double VisaGetResultNumber()
{
    double fResults = 0;

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
    CheckVisaStatus(nViStatus);

    return fResults;
}

private double[] VisaGetResultNumbers()
{
    double[] fResultsArray;
    fResultsArray = new double[10];
}

```

```

// Read return value string from the device.
int nViStatus;
nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
                           fResultsArray);
CheckVisaStatus(nViStatus);

return fResultsArray;
}

private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
    // Results array, big enough to hold a PNG.
    ResultsArray = new byte[300000];
    int length; // Number of bytes returned from instrument.

    // Set the default number of bytes that will be contained in
    // the ResultsArray to 300,000 (300kB).
    length = 300000;

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
                               ResultsArray);
    CheckVisaStatus(nViStatus);

    // Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
    CheckVisaStatus(nViStatus);

    nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
    CheckVisaStatus(nViStatus);

    return length;
}

private void CheckInstrumentErrors(string strCommand)
{
    // Check for instrument errors.
    StringBuilder strInstrumentError = new StringBuilder(1000);
    bool bFirstError = true;

    do // While not "0,No error"
    {
        VisaSendCommandOrQuery(":SYSTem:ERRor?");
        strInstrumentError = VisaGetResultString();

        if (!strInstrumentError.ToString().StartsWith("+0,"))
        {
            if (bFirstError)
            {
                Console.WriteLine("ERROR(s) for command '{0}': {1}",
                                  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 Keysight'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 Keysight IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include directory), select it, but *do not click the Open button*.
 - d** Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

- e** Right-click the project again and choose **Properties**; then, select "InfiniiVision.VisalnstrumentApp" 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 Keysight IO Libraries Suite 16.

```
'-----'
' Keysight VISA Example in Visual Basic .NET
' -----
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' -----'

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
    Class VisaInstrumentApp
        Private Shared myScope As VisaInstrument

        Public Shared Sub Main(ByVal args As String())
            Try
                myScope = _
                    New VisaInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR")
            "
                myScope.SetTimeoutSeconds(10)
            "
        End Sub
    End Class
End Namespace
```

```

' Initialize - start from a known state.
Initialize()

' Capture data.
Capture()

' Analyze the captured waveform.
Analyze()

Catch err As System.ApplicationException
    Console.WriteLine("**** VISA Error Message : " + err.Message)
Catch err As System.SystemException
    Console.WriteLine("**** System Error Message : " + err.Message)
Catch err As System.Exception
    Debug.Fail("Unexpected Error")
    Console.WriteLine("**** Unexpected Error : " + err.Message)
End Try
End Sub

'
' Initialize the oscilloscope to a known state.
' -----
Private Shared Sub Initialize()
    Dim strResults As StringBuilder

    ' Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?")
    Console.WriteLine("*IDN? result is: {0}", strResults)

    ' Clear status and load the default setup.
    myScope.DoCommand("*CLS")
    myScope.DoCommand("*RST")

End Sub

'
' Capture the waveform.
' -----
Private Shared Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale")

    ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE")
    Console.WriteLine("Trigger mode: {0}", _
        myScope.DoQueryString(":TRIGger:MODE?"))

    ' Set EDGE trigger parameters.
    myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
    Console.WriteLine("Trigger edge source: {0}", _
        myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

    myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
    Console.WriteLine("Trigger edge level: {0}", _

```

```

myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()      ' Results array.
Dim nLength As Integer         ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETUp?", _
    ResultsArray)

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and position.
myScope.DoCommand(":TIMEbase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMEbase:SCALe?"))

myScope.DoCommand(":TIMEbase:POSItion 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMEbase:POSItion?"))

' Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESolution)

myScope.DoCommand(":ACQuire:TYPE NORMAL")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim DataArray As Byte()
Dim nBytesWritten As Integer

' Read setup string from file.
strPath = "c:\scope\config\setup.stp"
DataArray = File.ReadAllBytes(strPath)

```

```

' Restore setup string.
nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup", _
    dataArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGitize.
myScope.DoCommand(":DIGitize CHANnel1")

End Sub

'

' Analyze the captured waveform.
' -----
Private Shared Sub Analyze()

    Dim fResult As Double
    Dim ResultsArray As Byte()      ' Results array.
    Dim nLength As Integer        ' Number of bytes returned from inst.
    Dim strPath As String

    ' Make a couple of measurements.
    ' -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1")
    Console.WriteLine("Measure source: {0}", _
        myScope.DoQueryString(":MEASure:SOURce?"))

    myScope.DoCommand(":MEASure:FREQuency")
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    myScope.DoCommand(":MEASure:VAMplitude")
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?")
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

    ' Download the screen image.
    ' -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF")

    ' Get the screen data.
    nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLOR", _
        ResultsArray)

    ' Store the screen data to a file.
    strPath = "c:\scope\data\screen.png"
    Dim fStream As FileStream
    fStream = File.Open(strPath, FileMode.Create)
    fStream.Write(ResultsArray, 0, nLength)
    fStream.Close()
    Console.WriteLine("Screen image ({0} bytes) written to {1}", _
        nLength, strPath)

    ' Download waveform data.
    ' -----
    ' Set the waveform points mode.
    myScope.DoCommand(":WAVeform:POINTS:MODE RAW")

```

```

Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVEform:POINts:MODE?"))

' Get the number of waveform points available.
myScope.DoCommand(":WAVEform:POINts 10240")
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVEform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVEform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMAT BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVEform:FORMAT?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
    Console.WriteLine("Waveform format: ASCII")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
    Console.WriteLine("Acquire type: NORMAL")
ElseIf fType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
    Console.WriteLine("Acquire type: AVERAGE")
ElseIf fType = 3 Then
    Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)

```

```

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVEform:DATA?", _
    ResultsArray)
Console.WriteLine("Number of data values: {0}", nLength)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"
If File.Exists(strPath) Then
    File.Delete(strPath)
End If

' Open file for output.
Dim writer As StreamWriter = File.CreateText(strPath)

' Output waveform data in CSV format.
For index As Integer = 0 To nLength - 1
    ' Write time value, voltage value.
    writer.WriteLine("{0:f9}, {1:f6}", _
        fXorigin + (CSng(index) * fxincrement), _
        ((CSng(ResultsArray(index)) - fYreference) _ 
         * fyincrement) + fyorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", _
    strPath)

End Sub

End Class

Class VisaInstrument
    Private m_nResourceManager As Integer
    Private m_nSession As Integer
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)
        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA resource manager.
        OpenResourceManager()

        ' Open a VISA resource session.
        OpenSession()
    End Sub
End Class

```

```

' Clear the interface.
Dim nViStatus As Integer
nViStatus = visa32.viClear(m_nSession)
End Sub

Public Sub DoCommand(ByVal strCommand As String)
    ' Send the command.
    VisaSendCommandOrQuery(strCommand)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

End Sub

Public Function DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte()) As Integer

    ' Send the command to the device.
    Dim strCommandAndLength As String
    Dim nViStatus As Integer
    Dim nLength As Integer
    Dim nBytesWritten As Integer

    nLength = DataArray.Length
    strCommandAndLength = [String].Format("{0} #8{1:D8}", _
        strCommand, nLength)

    ' Write first part of command to formatted I/O write buffer.
    nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
    CheckVisaStatus(nViStatus)

    ' Write the data to the formatted I/O write buffer.
    nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength, _
        nBytesWritten)
    CheckVisaStatus(nViStatus)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

    Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) _
    As StringBuilder
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim strResults As New StringBuilder(1000)
    strResults = VisaGetResultString()

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return strResults
End Function

```

```

Public Function DoQueryNumber(ByVal strQuery As String) As Double
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim fResults As Double
    fResults = VisaGetResultNumber()

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return fResults
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) _
    As Double()
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim fResultsArray As Double()
    fResultsArray = VisaGetResultNumbers()

    ' Check for instrument errors (another command and result).
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String, _
    ByRef ResultsArray As Byte()) As Integer
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    System.Threading.Thread.Sleep(2000) ' Delay before reading data.
    Dim length As Integer
    ' Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(ResultsArray)

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return length
End Function

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _ 
    As String)
    ' Send command or query to the device.
    Dim strWithNewline As String
    strWithNewline = [String].Format("{0}" & Chr(10) & "", _
        strCommandOrQuery)
    Dim nViStatus As Integer

```

```

nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetString() As StringBuilder
    Dim strResults As New StringBuilder(1000)

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
    CheckVisaStatus(nViStatus)

    Return strResults
End Function

Private Function VisaGetResultNumber() As Double
    Dim fResults As Double = 0

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
    CheckVisaStatus(nViStatus)

    Return fResults
End Function

Private Function VisaGetResultNumbers() As Double()
    Dim fResultsArray As Double()
    fResultsArray = New Double(9) {}

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession,
        "%,10lf" & Chr(10) & "", fResultsArray)
    CheckVisaStatus(nViStatus)

    Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _
    As Byte()) As Integer
    ' Results array, big enough to hold a PNG.
    ResultsArray = New Byte(299999) {}
    Dim length As Integer
    ' Number of bytes returned from instrument.
    ' Set the default number of bytes that will be contained in
    ' the ResultsArray to 300,000 (300kB).
    length = 300000

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%#b", length, _
        ResultsArray)
    CheckVisaStatus(nViStatus)

    ' Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)

```

```

CheckVisaStatus(nViStatus)

nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
CheckVisaStatus(nViStatus)

    Return length
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
    ' Check for instrument errors.
    Dim strInstrumentError As New StringBuilder(1000)
    Dim bFirstError As Boolean = True
    Do      ' While not "0,No error"
        VisaSendCommandOrQuery(":SYSTem:ERRor?")
        strInstrumentError = VisaGetResultString()

        If Not strInstrumentError.ToString().StartsWith("+0,") Then
            If bFirstError Then
                Console.WriteLine("ERROR(s) for command '{0}': {1}, _",
                    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

```

```

        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 Keysight 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 Keysight oscilloscope.
# ****

# Import modules.
# -----
import visa
import string
import struct
import sys

# Global variables (booleans: 0 = False, 1 = True).
# -----
debug = 0

# =====
# 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():

    # 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:EDGE:LEVel 1.5")
    qresult = do_query_string(":TRIGger:EDGE:LEVel?")
    print "Trigger edge level: %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 0.05")
    qresult = do_query_values(":CHANnel1:SCALe?") [0]
    print "Channel 1 vertical scale: %f" % qresult

    do_command(":CHANnel1:OFFSet -1.5")
    qresult = do_query_values(":CHANnel1:OFFSet?") [0]
    print "Channel 1 offset: %f" % qresult

    # Set horizontal scale and offset.
    do_command(":TIMEbase:SCALe 0.0002")

```

```

qresult = do_query_string(":TIMEbase:SCALe?")
print "Timebase scale: %s" % qresult

do_command(":TIMEbase:POSIon 0.0")
qresult = do_query_string(":TIMEbase:POSIon?")
print "Timebase position: %s" % qresult

# Set the acquisition type.
do_command(":ACQuire:TYPE NORMal")
qresult = do_query_string(":ACQuire:TYPE?")
print "Acquire type: %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)

# Capture an acquisition using :DIGItize.
do_command(":DIGItize CHANnel1")

# =====
# 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:VAMPplitude")
    qresult = do_query_string(":MEASure:VAMPplitude?")
    print "Measured vertical amplitude on channel 1: %s" % qresult

    # Download the screen image.
    # -----
    do_command(":HARDcopy:INKSaver OFF")

    sDisplay = do_query_string(":DISPlay:DATA? PNG, COLOR")
    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.
# -----
#
# Set the waveform points mode.
do_command(":WAVeform:POINts:MODE RAW")
qresult = do_query_string(":WAVeform:POINts:MODE?")
print "Waveform points mode: %s" % qresult

# Get the number of waveform points available.
do_command(":WAVeform:POINts 10240")
qresult = do_query_string(":WAVeform:POINts?")
print "Waveform points available: %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 : "BYTE",
    1 : "WORD",
    4 : "ASCII",
}
acq_type_dict = {
    0 : "NORMal",
    1 : "PEAK",
    2 : "AVERage",
    3 : "HRESolution",
}

preamble_string = do_query_string(":WAVeform:PREamble?")
(
    wav_form, acq_type, wfmpnts, avgcnt, x_increment, x_origin,
    x_reference, y_increment, y_origin, y_reference
) = 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" % wfmpnts
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

# 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]

```

```

y_reference = do_query_values(":WAVeform:YREFerence?") [0]

# Get the waveform data.
sData = do_query_string(":WAVeform:DATA?")
sData = get_definite_length_block_data(sData)

# Unpack unsigned 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_reference) * 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

    InfiniiVision.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 = InfiniiVision.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 = InfiniiVision.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 = InfiniiVision.ask(":SYSTem:ERRor?\n")
        if error_string:    # If there is an error string value.

            if error_string.find("+0,", 0, 3) == -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? should always return string.
            print "ERROR: :SYSTem:ERRor? 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'." % pound
        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:

```

```
# =====
InfiniiVision = visa.instrument("TCPIP0::130.29.70.139::inst0::INSTR")
InfiniiVision.timeout = 15
InfiniiVision.term_chars = ""
InfiniiVision.clear()

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program."
```

SICL Examples

- "SICL Example in C" on page 884
- "SICL Example in Visual Basic" on page 893

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.

```
/*
 * Keysight SICL Example in C
 * -----
 * This program illustrates a few commonly-used programming
 * features of your Keysight oscilloscope.
 */
#include <stdio.h>           /* For printf() . */
```

```

#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>            /* For clock(). */
#include <sicl.h>             /* Keysight SICL routines. */

#define SICL_ADDRESS      "usb0[2391::6054::US50210029::0]"
#define TIMEOUT          5000
#define IEEEBLOCK_SPACE  100000

/* Function prototypes */
void initialize(void);           /* Initialize to known state. */
void capture(void);              /* Capture the waveform. */
void analyze(void);              /* Analyze the captured waveform. */

void do_command(char *command);   /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query); /* Query for string. */
void do_query_number(char *query); /* Query for number. */
void do_query_numbers(char *query); /* Query for numbers. */
int do_query_ieeeblock(char *query); /* Query for IEEE block. */
void check_instrument_errors();   /* Check for inst errors. */

/* Global variables */
INST id;                         /* Device session ID. */
char str_result[256] = {0};        /* Result from do_query_string(). */
double num_result;                /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE]; /* Result from
                                               do_query_ieeeblock(). */
double dbl_results[10];           /* Result from do_query_numbers(). */

/* Main Program
 * -----
void main(void)
{
    /* Install a default SICL error handler that logs an error message
     * and exits. On Windows 98SE or Windows Me, view messages with
     * the SICL Message Viewer. For Windows 2000 or XP, use the Event
     * Viewer.
    */
    ionerror(I_ERROR_EXIT);

    /* Open a device session using the SICL_ADDRESS */
    id = iopen(SICL_ADDRESS);

    if (id == 0)
    {
        printf ("Oscilloscope iopen failed!\n");
    }
    else
    {
        printf ("Oscilloscope session opened!\n");
    }

    /* Initialize - start from a known state. */
    initialize();

    /* Capture data. */
    capture();
}

```

```

/* Analyze the captured waveform. */
analyze();

/* Close the device session to the instrument. */
iclose(id);
printf ("Program execution is complete...\n");

/* For WIN16 programs, call _siclcleanup before exiting to release
 * resources allocated by SICL for this application. This call is
 * a no-op for WIN32 programs.
 */
_siclcleanup();
}

/* Initialize the oscilloscope to a known state.
 * -----
void initialize (void)
{
    /* Set the I/O timeout value for this session to 5 seconds. */
    itimeout(id, TIMEOUT);

    /* Clear the interface. */
    iclear(id);

    /* Get and display the device's *IDN? string. */
    do_query_string("*IDN?");
    printf("Oscilloscope *IDN? string: %s\n", str_result);

    /* Clear status and load the default setup. */
    do_command("*CLS");
    do_command("*RST");
}

/* Capture the waveform.
 * -----
void capture (void)
{
    int num_bytes;
    FILE *fp;

    /* Use auto-scale to automatically configure oscilloscope.
     * -----
    do_command(":AUToscale");

    /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. */
    do_command(":TRIGger:MODE EDGE");
    do_query_string(":TRIGger:MODE?");
    printf("Trigger mode: %s\n", str_result);

    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURCe CHANnel1");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);

    do_command(":TRIGger:EDGE:LEVel 1.5");
    do_query_string(":TRIGger:EDGE:LEVel?");
}

```

```

printf("Trigger edge level: %s\n", str_result);

do_command(":TRIGger:EDGE:SLOPe POSitive");
do_query_string(":TRIGger:EDGE:SLOPe?");
printf("Trigger edge slope: %s\n", str_result);

/* Save oscilloscope configuration.
 * -----
 */

/* Read system setup. */
num_bytes = do_query_ieeeblock(":SYSTem:SETUp?");
printf("Read setup string query (%d bytes).\n", num_bytes);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:
 * -----
 */

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.05");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and position. */
do_command(":TIMEbase:SCALe 0.0002");
do_query_string(":TIMEbase:SCALe?");
printf("Timebase scale: %s\n", str_result);

do_command(":TIMEbase:POSIon 0.0");
do_query_string(":TIMEbase:POSIon?");
printf("Timebase position: %s\n", str_result);

/* Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESolution). */
do_command(":ACQuire:TYPE NORMAL");
do_query_string(":ACQuire:TYPE?");
printf("Acquire type: %s\n", str_result);

/* Or, configure by loading a previously saved setup.
 * -----
 */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file ", num_bytes);

```

```

printf("c:\\\\scope\\\\config\\\\setup.stp.\n");

/* Restore setup string. */
num_bytes = do_command_ieeeblock(":SYSTem:SETUp", num_bytes);
printf("Restored setup string (%d bytes).\n", num_bytes);

/* Capture an acquisition using :DIGitize.
 * -----
do_command(":DIGitize CHANnel1");
}

/* Analyze the captured waveform.
 * -----
void analyze (void)
{
    double wav_format;
    double acq_type;
    double wav_points;
    double avg_count;
    double x_increment;
    double x_origin;
    double x_reference;
    double y_increment;
    double y_origin;
    double y_reference;

    FILE *fp;
    int num_bytes; /* Number of bytes returned from instrument. */
    int i;

    /* Make a couple of measurements.
     * -----
do_command(":MEASure:SOURce CHANnel1");
do_query_string(":MEASure:SOURce?");
printf("Measure source: %s\n", str_result);

do_command(":MEASure:FREQuency");
do_query_number(":MEASure:FREQuency?");
printf("Frequency: %.4f kHz\n", num_result / 1000);

do_command(":MEASure:VAMPplitude");
do_query_number(":MEASure:VAMPplitude?");
printf("Vertical amplitude: %.2f V\n", num_result);

/* Download the screen image.
 * -----
do_command(":HARDcopy:INKSaver OFF");

/* Read screen image. */
num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG, COLOR");
printf("Screen image bytes: %d\n", num_bytes);

/* Write screen image bytes to file. */
fp = fopen ("c:\\\\scope\\\\data\\\\screen.png", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
                  fp);
fclose (fp);
}

```

```

printf("Wrote screen image (%d bytes) to ", num_bytes);
printf("c:\\scope\\data\\screen.png.\n");

/* Download waveform data.
 * -----
 */

/* Set the waveform points mode. */
do_command(":WAVeform:POINTs:MODE RAW");
do_query_string(":WAVeform:POINTs:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Get the number of waveform points available. */
do_command(":WAVeform:POINTs 10240");
do_query_string(":WAVeform:POINTs?");
printf("Waveform points available: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURCE?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMAT BYTE");
do_query_string(":WAVeform:FORMAT?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
    printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
    printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
    printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
    printf("Acquire type: NORMAl\n");
}
else if (acq_type == 1.0)
{
    printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
    printf("Acquire type: AVERage\n");
}
else if (acq_type == 3.0)

```

```

{
    printf("Acquire type: HRESolution\n");
}

wav_points = dbl_results[2];
printf("Waveform points: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);

x_origin = dbl_results[5];
printf("Waveform X origin: %e\n", x_origin);

x_reference = dbl_results[6];
printf("Waveform X reference: %e\n", x_reference);

y_increment = dbl_results[7];
printf("Waveform Y increment: %e\n", y_increment);

y_origin = dbl_results[8];
printf("Waveform Y origin: %e\n", y_origin);

y_reference = dbl_results[9];
printf("Waveform Y reference: %e\n", y_reference);

/* Read waveform data. */
num_bytes = do_query_ieeeblock(":WAVEFORM:DATA?");
printf("Number of data values: %d\n", num_bytes);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

/* Output waveform data in CSV format. */
for (i = 0; i < num_bytes - 1; i++)
{
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
            x_origin + ((float)i * x_increment),
            (((float)ieeeblock_data[i] - y_reference) * y_increment)
            + y_origin);
}

/* Close output file. */
fclose(fp);
printf("Waveform format BYTE data written to ");
printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * -----
void do_command(command)
char *command;
{
    char message[80];

```

```

    strcpy(message, command);
    strcat(message, "\n");
    iprintf(id, message);

    check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * -----
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
    char message[80];
    int data_length;

    strcpy(message, command);
    strcat(message, "#8%08d");
    iprintf(id, message, num_bytes);
    ifwrite(id, ieeeblock_data, num_bytes, 1, &data_length);

    check_instrument_errors();

    return(data_length);
}

/* Query for a string result.
 * -----
void do_query_string(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%t\n", str_result);

    check_instrument_errors();
}

/* Query for a number result.
 * -----
void do_query_number(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%lf", &num_result);

    check_instrument_errors();
}

```

```

    }

/* Query for numbers result.
 * -----
void do_query_numbers(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    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);
}

/* Check for instrument errors.
 * -----
void check_instrument_errors()
{
    char str_err_val[256] = {0};
    char str_out[800] = "";

    ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
    while(strncmp(str_err_val, "+0,No error", 3) != 0 )
    {
        strcat(str_out, ", ");
        strcat(str_out, str_err_val);
        ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
    }
}

```

```

    }

    if (strcmp(str_out, "") != 0)
    {
        printf("INST Error%s\n", str_out);
        iflush(id, I_BUF_READ | I_BUF_WRITE);
    }
}

```

SICL Example in Visual Basic

To run this example in Visual Basic for Applications:

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Add the sicl32.bas file to your project:
 - a Choose **File > Import File....**
 - b Navigate to the header file, sicl32.bas (installed with Keysight 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.

```

'
' Keysight SICL Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' -----


Option Explicit

Public id As Integer      ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/ifread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

' For Sleep subroutine.

```

```

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

'
' Main Program
' -----
'

Sub Main()

    On Error GoTo ErrorHandler

    ' Open a device session using the SICL_ADDRESS.
    id = iopen("usb0[2391::6054::US50210029::0]")
    Call itimeout(id, 5000)

    ' Initialize - start from a known state.
    Initialize

    ' Capture data.
    Capture

    ' Analyze the captured waveform.
    Analyze

    ' Close the vi session and the resource manager session.
    Call iclose(id)

    Exit Sub

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
    End

End Sub

'
' Initialize the oscilloscope to a known state.
' -----
'

Private Sub Initialize()

    On Error GoTo ErrorHandler

    ' Clear the interface.
    Call iclear(id)

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

    Exit Sub

ErrorHandler:

```

```

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

'

' Capture the waveform.
' -----
Private Sub Capture()

On Error GoTo ErrorHandler

' Use auto-scale to automatically configure oscilloscope.
' -----
DoCommand ":AUToscale"

' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
DoQueryString(":TRIGger:MODE?")

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
Debug.Print "Trigger edge source: " + _
DoQueryString(":TRIGger:EDGE:SOURce?")

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
DoQueryString(":TRIGger:EDGE:LEVel?")

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
DoQueryString(":TRIGger:EDGE:SLOPe?")

' Save oscilloscope configuration.
' -----
Dim lngSetupStringSize As Long
lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
If Len(Dir(strPath)) Then
    Kill strPath      ' Remove file if it exists.
End If

' Open file for output.
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngSetupStringSize - 1
    Put hFile, , byteArray(lngI)      ' Write data.
Next lngI

```

```

        Close hFile      ' Close file.

        ' Change settings with individual commands:
        ' -----
        ' Set vertical scale and offset.
        DoCommand ":CHANnel1:SCALe 0.05"
        Debug.Print "Channel 1 vertical scale: " + _
                    DoQueryString(":CHANnel1:SCALe?")

        DoCommand ":CHANnel1:OFFSet -1.5"
        Debug.Print "Channel 1 vertical offset: " + _
                    DoQueryString(":CHANnel1:OFFSet?")

        ' Set horizontal scale and position.
        DoCommand ":TIMEbase:SCALe 0.0002"
        Debug.Print "Timebase scale: " + _
                    DoQueryString(":TIMEbase:SCALe?")

        DoCommand ":TIMEbase:POSIon 0.0"
        Debug.Print "Timebase position: " + _
                    DoQueryString(":TIMEbase:POSIon?")

        ' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
        DoCommand ":ACQuire:TYPE NORMal"
        Debug.Print "Acquire type: " + _
                    DoQueryString(":ACQuire:TYPE?")

        ' Or, configure by loading a previously saved setup.
        ' -----
        strPath = "c:\scope\config\setup.dat"
        Open strPath For Binary Access Read As hFile      ' Open file for input.
        Dim lngSetupFileSize As Long
        lngSetupFileSize = LOF(hFile)      ' Length of file.
        Get hFile, , byteArray      ' Read data.
        Close hFile      ' Close file.
        ' Write setup string back to oscilloscope using ":SYSTem:SETup"
        ' command:
        Dim lngRestored As Long
        lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
        Debug.Print "Setup bytes restored: " + CStr(lngRestored)

        ' Capture an acquisition using :DIGitize.
        ' -----
        DoCommand ":DIGitize CHANnel1"

        Exit Sub

ErrorHandler:
        MsgBox "*** Error : " + Error, vbExclamation
        End

End Sub

        '
        ' Analyze the captured waveform.

```

```

' -----
Private Sub Analyze()

    On Error GoTo ErrorHandler

    ' Make a couple of measurements.
    ' -----
    DoCommand ":MEASure:SOURce CHANnel1"
    Debug.Print "Measure source: " + _
        DoQueryString(":MEASure:SOURce?")

    DoCommand ":MEASure:FREQuency"
    dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
    MsgBox "Frequency:" + vbCrLf + _
        FormatNumber(dblQueryResult / 1000, 4) + " kHz"

    DoCommand ":MEASure:VAMPplitude"
    dblQueryResult = DoQueryNumber(":MEASure:VAMPplitude?")
    MsgBox "Vertical amplitude:" + vbCrLf + _
        FormatNumber(dblQueryResult, 4) + " V"

    ' Download the screen image.
    ' -----
    DoCommand ":HARDcopy:INKSaver OFF"

    ' Get screen image.
    Dim lngBlockSize As Long
    lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPLAY:DATA? PNG, COLOR")
    Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

    ' Save screen image to a file:
    Dim strPath As String
    strPath = "c:\scope\data\screen.png"
    If Len(Dir(strPath)) Then
        Kill strPath      ' Remove file if it exists.
    End If
    Dim hFile As Long
    hFile = FreeFile
    Open strPath For Binary Access Write Lock Write As hFile
    Dim lngI As Long
    ' Skip past header.
    For lngI = CInt(Chr(byteArray(1))) + 2 To lngBlockSize - 1
        Put hFile, , byteArray(lngI)      ' Write data.
    Next lngI
    Close hFile      ' Close file.
    MsgBox "Screen image written to " + strPath

    ' Download waveform data.
    ' -----
    ' Set the waveform points mode.
    DoCommand ":WAVEform:POINTS:MODE RAW"
    Debug.Print "Waveform points mode: " + _
        DoQueryString(":WAVEform:POINTS:MODE?")

    ' Get the number of waveform points available.

```

```

DoCommand ":WAVeform:POINTs 10240"
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINTs?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMAT BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMAT?")

' Display the waveform settings:
Dim Preamble() As Double
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
    Debug.Print "Waveform format: ASCII"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMAL"
ElseIf intType = 1 Then
    Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
    Debug.Print "Acquisition type: AVERAGE"
ElseIf intType = 3 Then
    Debug.Print "Acquisition type: HRESolution"
End If

```

```

Debug.Print "Waveform points: " + _
FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
FormatNumber(lngYReference, 0)

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVEFORM:DATA?")
Debug.Print "Number of data values: " + _
CStr(lngNumBytes - CInt(Chr(byteArray(1))) - 2)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long

' Skip past header.
For lngI = CInt(Chr(byteArray(1))) + 2 To lngNumBytes - 2
    lngDataValue = CLng(byteArray(lngI))

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngDataValue - lngYReference) * _
        sngYIncrement) + sngYOrigin)

Next lngI

' Close output file.
Close hFile    ' Close file.
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 + vbCrLf)

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.
Sleep 2000      ' Delay before reading data.
Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)

' Get number of block length digits.
Dim intLengthDigits As Integer
intLengthDigits = CInt(Chr(byteArray(1)))

' Get block length from those digits.
Dim strBlockLength As String
strBlockLength = ""
Dim i As Integer
For i = 2 To intLengthDigits + 1
    strBlockLength = strBlockLength + Chr(byteArray(i))
Next

' Return number of bytes in block plus header.
DoQueryIEEEBlock_Bytes = CLng(strBlockLength) + intLengthDigits + 2

CheckInstrumentErrors

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Sub CheckInstrumentErrors()

On Error GoTo ErrorHandler

Dim strErrVal As String * 200
Dim strOut As String

Call ivprintf(id, ":SYSTem:ERRor?" + vbLf)      ' Query any errors data.
Call ivscanf(id, "%200t", strErrVal)      ' Read: Errnum, "Error String".
While Val(strErrVal) <> 0                  ' End if find: +0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    Call ivprintf(id, ":SYSTem:ERRor?" + vbLf)      ' Request error message
    Call ivscanf(id, "%200t", strErrVal)      ' Read error message.
Wend

If Not strOut = "" Then

```

```
MsgBox strOut, vbExclamation, "INST Error Messages"
Call iflush(id, I_BUF_READ Or I_BUF_WRITE)

End If

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub
```

SCPI.NET Examples

These programming examples show how to use the SCPI.NET drivers that come with Keysight'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, Keysight VEE, and Keysight SystemVue.

For more information on Keysight Command Expert, and to download the software, see: <http://www.keysight.com/find/commandexpert>

- ["SCPI.NET Example in C#" on page 904](#)
- ["SCPI.NET Example in Visual Basic .NET" on page 910](#)
- ["SCPI.NET Example in IronPython" on page 916](#)

SCPI.NET Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Install the Keysight 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 **AgInfiniiVision2000X_01_20.dll**; then, click **OK**.

7 Build and run the program.

For more information, see the SCPI.NET driver help that comes with Keysight Command Expert.

```
/*
 * Keysight SCPI.NET Example in C#
 * -----
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * -----
 */

using System;
using System.IO;
using System.Text;
using Agilent.CommandExpert.ScpiNet.AgInfiniiVision2000X_01_20;

namespace InfiniiVision
{
    class ScpiNetInstrumentApp
    {
        private static AgInfiniiVision2000X myScope;

        static void Main(string[] args)
        {
            try
            {
                string strScopeAddress;
                //strScopeAddress = "a-mx2024a-10029.cos.is.keysight.com";
                strScopeAddress =
                    "TCPIPO:a-mx2024a-10029.cos.is.keysight.com::inst0::INSTR";
                Console.WriteLine("Connecting to oscilloscope...");
                Console.WriteLine();
                myScope = new AgInfiniiVision2000X(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;

    // Use auto-scale to automatically configure oscilloscope.
    myScope.SCPI.AUToscale.Command(null, null, null, null, null);

    // 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("CHANnel1");
    myScope.SCPI.TRIGger.EDGE.SOURce.Query(out strResults);
}

```

```

Console.WriteLine("Trigger edge source: {0}", strResults);

myScope.SCPI.TRIGger.EDGE.LEVEL.Command(1.5, "CHANnel1");
myScope.SCPI.TRIGger.EDGE.LEVEL.Query("CHANnel1", out fResult);
Console.WriteLine("Trigger edge level: {0:F2}", fResult);

myScope.SCPI.TRIGger.EDGE.SLOPe.Command("POSitive");
myScope.SCPI.TRIGger.EDGE.SLOPe.Query(out strResults);
Console.WriteLine("Trigger edge slope: {0}", strResults);

// Save oscilloscope configuration.
string[] strResultsArray; // Results array.
int nLength; // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
myScope.SCPI.SYSTem.SETup.Query(out strResultsArray);
nLength = strResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
File.WriteAllLines(strPath, strResultsArray);
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.SCPI.CHANnel.SCALE.Command(1, 0.05);
myScope.SCPI.CHANnel.SCALE.Query(1, out fResult);
Console.WriteLine("Channel 1 vertical scale: {0:F4}", fResult);

myScope.SCPI.CHANnel.OFFSet.Command(1, -1.5);
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 type.
myScope.SCPI.ACQuire.TYPE.Command("NORMAL");
myScope.SCPI.ACQuire.TYPE.Query(out strResults);
Console.WriteLine("Acquire type: {0}", strResults);

// Or, configure by loading a previously saved setup.
int nBytesWritten;

strPath = "c:\\scope\\config\\setup.stp";
strResultsArray = File.ReadAllLines(strPath);
nBytesWritten = strResultsArray.Length;

```

```

// Restore setup string.
myScope.SCPI.SYSTem.SETup.Command(strResultsArray);
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

// Capture an acquisition using :DIGItize.
myScope.SCPI.DIGItize.Command("CHANnel1", null, null, null, null);
}

/*
 * Analyze the captured waveform.
 * -----
 */
private static void Analyze()
{
    string strResults, source1, source2;
    double fResult;

    // Make a couple of measurements.
    // -----
    myScope.SCPI.MEASure.SOURce.Command("CHANnel1", null);
    myScope.SCPI.MEASure.SOURce.Query(out source1, out source2);
    Console.WriteLine("Measure source: {0}", source1);

    myScope.SCPI.MEASure.FREQuency.Command("CHANnel1");
    myScope.SCPI.MEASure.FREQuency.Query("CHANnel1", 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.
    // -----
    myScope.SCPI.HARDcopy.INKSaver.Command(false);

    // Get the screen data.
    byte[] byteResultsArray;    // Results array.
    myScope.SCPI.DISPlay.DATA.Query("PNG", "COLOR",
        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.
    // -----
}

// Set the waveform points mode.

```

```

myScope.SCPI.WAVeform.POINts.MODE.Command("RAW");
myScope.SCPI.WAVeform.POINts.MODE.Query(out strResults);
Console.WriteLine("Waveform points mode: {0}", strResults);

// Get the number of waveform points available.
myScope.SCPI.WAVeform.POINts.CommandPoints(10240);
int nPointsAvail;
myScope.SCPI.WAVeform.POINts.Query1(out nPointsAvail);
Console.WriteLine("Waveform points available: {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 (WORD, BYTE, ASCII):
myScope.SCPI.WAVeform.FORMat.Command("BYTE");
myScope.SCPI.WAVeform.FORMat.Query(out strResults);
Console.WriteLine("Waveform format: {0}", strResults);

// Display the waveform settings:
int nFormat, nType, nPoints, nCount, nXreference, nYreference;
double dblXincrement, dblXorigin, dblYincrement, dblYorigin;
myScope.SCPI.WAVeform.PREamble.Query(
    out nFormat,
    out nType,
    out nPoints,
    out nCount,
    out dblXincrement,
    out dblXorigin,
    out nXreference,
    out dblYincrement,
    out dblYorigin,
    out nYreference);

if (nFormat == 0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (nFormat == 1)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (nFormat == 2)
{
    Console.WriteLine("Waveform format: ASCII");
}

if (nType == 0)
{
    Console.WriteLine("Acquire type: NORMAL");
}
else if (nType == 1)
{
    Console.WriteLine("Acquire type: PEAK");
}
else if (nType == 2)

```

```

        {
            Console.WriteLine("Acquire type: AVERage");
        }
    else if (nType == 3)
    {
        Console.WriteLine("Acquire type: HRESolution");
    }

    Console.WriteLine("Waveform points: {0:e}", nPoints);
    Console.WriteLine("Waveform average count: {0:e}", nCount);
    Console.WriteLine("Waveform X increment: {0:e}", dblXincrement);
    Console.WriteLine("Waveform X origin: {0:e}", dblXorigin);
    Console.WriteLine("Waveform X reference: {0:e}", nXreference);
    Console.WriteLine("Waveform Y increment: {0:e}", dblYincrement);
    Console.WriteLine("Waveform Y origin: {0:e}", dblYorigin);
    Console.WriteLine("Waveform Y reference: {0:e}", nYreference);

    // Read waveform data.
    myScope.SCPI.WAVeform.DATA.QueryBYTE(out byteResultsArray);
    nLength = byteResultsArray.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}",
                        dblXorigin + ((float)i * dblXincrement),
                        (((float)byteResultsArray[i] - nYreference)
                         * dblYincrement) + dblYorigin);

    // Close output file.
    writer.Close();
    Console.WriteLine("Waveform format BYTE 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 Keysight 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 **AgInfiniiVision2000X_01_20.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 "InfiniiVision.ScpiNetInstrumentApp" as the **Startup object**.
- 7** Build and run the program.

For more information, see the SCPI.NET driver help that comes with Keysight Command Expert.

```
'-----'
' Keysight SCPI.NET Example in Visual Basic .NET
' -----
' This program illustrates a few commonly used programming
' features of your Keysight oscilloscope.
' -----'

Imports System
Imports System.IO
Imports System.Text
Imports Agilent.CommandExpert.ScpiNet.AgInfiniiVision2000X_01_20

Namespace InfiniiVision
    Class ScpiNetInstrumentApp
        Private Shared myScope As AgInfiniiVision2000X

        Public Shared Sub Main(ByVal args As String())
            Try
                Dim strScopeAddress As String
                'strScopeAddress = "a-mx2024a-10029.cos.is.keysight.com";
                strScopeAddress =
                    "TCPIP0::a-mx2024a-10029.cos.is.keysight.com::inst0::INSTR"
                Console.WriteLine("Connecting to oscilloscope...")
                Console.WriteLine()
                myScope = New AgInfiniiVision2000X(strScopeAddress)
                myScope.Transport.DefaultTimeout.[Set](10000)

                ' Initialize - start from a known state.
                Initialize()
            End Try
        End Sub
    End Class
End Namespace
```

```

' 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

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.SCPI.AUToscale.Command(Nothing, Nothing, Nothing, _
        Nothing, Nothing)

    ' 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("CHANnel1")
    myScope.SCPI.TRIGger.EDGE.SOURce.Query(strResults)
    Console.WriteLine("Trigger edge source: {0}", strResults)

```

```

myScope.SCPI.TRIGger.EDGE.LEVEL.Command(1.5, "CHANnel1")
myScope.SCPI.TRIGger.EDGE.LEVEL.Query("CHANnel1", fResult)
Console.WriteLine("Trigger edge level: {0:F2}", fResult)

myScope.SCPI.TRIGger.EDGE.SLOPe.Command("POSitive")
myScope.SCPI.TRIGger.EDGE.SLOPe.Query(strResults)
Console.WriteLine("Trigger edge slope: {0}", strResults)

' Save oscilloscope configuration.
Dim strResultsArray As String()
' 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(strResultsArray)
nLength = strResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
File.WriteAllLines(strPath, strResultsArray)
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.SCPI.CHANnel.SCALE.Command(1, 0.05)
myScope.SCPI.CHANnel.SCALE.Query(1, fResult)
Console.WriteLine("Channel 1 vertical scale: {0:F4}", fResult)

myScope.SCPI.CHANnel.OFFSet.Command(1, -1.5)
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 type.
myScope.SCPI.ACQuire.TYPE.Command("NORMAL")
myScope.SCPI.ACQuire.TYPE.Query(strResults)
Console.WriteLine("Acquire type: {0}", strResults)

' Or, configure by loading a previously saved setup.
Dim nBytesWritten As Integer

strPath = "c:\scope\config\setup.stp"
strResultsArray = File.ReadAllLines(strPath)
nBytesWritten = strResultsArray.Length

```

```

' Restore setup string.
myScope.SCPI.SYSTem.SETup.Command(strResultsArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGItize.
myScope.SCPI.DIGItize.Command("CHANnel1", Nothing, Nothing, _
                               Nothing, Nothing)
End Sub

' Analyze the captured waveform.
' -----
Private Shared Sub Analyze()
    Dim strResults As String, source1 As String, source2 As String
    Dim fResult As Double

    ' Make a couple of measurements.
    ' -----
    myScope.SCPI.MEASure.SOURce.Command("CHANnel1", Nothing)
    myScope.SCPI.MEASure.SOURce.Query(source1, source2)
    Console.WriteLine("Measure source: {0}", source1)

    myScope.SCPI.MEASure.FREQuency.Command("CHANnel1")
    myScope.SCPI.MEASure.FREQuency.Query("CHANnel1", 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.
    ' -----
    myScope.SCPI.HARDcopy.INKSaver.Command(False)

    ' Get the screen data.
    Dim byteResultsArray As Byte()
    ' Results array.
    myScope.SCPI.DISPlay.DATA.Query("PNG", "COLOR", 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.
    ' -----

```

```

' Set the waveform points mode.
myScope.SCPI.WAVEform.POINTs.MODE.Command("RAW")
myScope.SCPI.WAVEform.POINTs.MODE.Query(strResults)
Console.WriteLine("Waveform points mode: {0}", strResults)

' Get the number of waveform points available.
myScope.SCPI.WAVEform.POINTs.CommandPoints(10240)
Dim nPointsAvail As Integer
myScope.SCPI.WAVEform.POINTs.Query1(nPointsAvail)
Console.WriteLine("Waveform points available: {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 (WORD, BYTE, ASCII):
myScope.SCPI.WAVEform.FORMAT.Command("BYTE")
myScope.SCPI.WAVEform.FORMAT.Query(strResults)
Console.WriteLine("Waveform format: {0}", strResults)

' Display the waveform settings:
Dim nFormat As Integer, nType As Integer, nPoints As Integer, _
    nCount As Integer, nXreference As Integer, _
    nYreference As Integer
Dim dblXincrement As Double, dblXorigin As Double, _
    dblYincrement As Double, dblYorigin As Double
myScope.SCPI.WAVEform.PREAMble.Query(nFormat, nType, nPoints, _
    nCount, dblXincrement, dblXorigin, nXreference, _
    dblYincrement, dblYorigin, nYreference)

If nFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf nFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf nFormat = 2 Then
    Console.WriteLine("Waveform format: ASCII")
End If

If nType = 0 Then
    Console.WriteLine("Acquire type: NORMAL")
ElseIf nType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf nType = 2 Then
    Console.WriteLine("Acquire type: AVERAGE")
ElseIf nType = 3 Then
    Console.WriteLine("Acquire type: HRESOLUTION")
End If

Console.WriteLine("Waveform points: {0:e}", nPoints)
Console.WriteLine("Waveform average count: {0:e}", nCount)
Console.WriteLine("Waveform X increment: {0:e}", dblXincrement)
Console.WriteLine("Waveform X origin: {0:e}", dblXorigin)
Console.WriteLine("Waveform X reference: {0:e}", nXreference)
Console.WriteLine("Waveform Y increment: {0:e}", dblYincrement)
Console.WriteLine("Waveform Y origin: {0:e}", dblYorigin)

```

```

Console.WriteLine("Waveform Y reference: {0:e}", nYreference)

' Read waveform data.
myScope.SCPI.WAVEform.DATA.QueryBYTE(byteResultsArray)
nLength = byteResultsArray.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}", _
        dblXorigin + (CSng(i) * dblXincrement), _
        ((CSng(byteResultsArray(i)) - nYreference) * _
        dblYincrement) + dblYorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", _
    strPath)
End Sub
End Class
End Namespace

```

SCPI.NET Example in IronPython

You can also control Keysight 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 Keysight 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

#
# Keysight SCPI.NET Example in IronPython

```

```

# ****
# This program illustrates a few commonly used programming
# features of your Keysight oscilloscope.
# ****

# Import Python modules.
# -----
import sys
sys.path.append("C:\Python27\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("AgInfiniiVision2000X_01_20")
from Agilent.CommandExpert.ScpiNet.AgInfiniiVision2000X_01_20 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():

    # Use auto-scale to automatically set up oscilloscope.
    print "Autoscale."
    scope.SCPI.AUToscale.Command(None, None, None, None, None)

    # 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("CHANnel1")
    qresult = scope.SCPI.TRIGger.EDGE.SOURce.Query()
    print "Trigger edge source: %s" % qresult

    scope.SCPI.TRIGger.EDGE.LEVel.Command(1.5, "CHANnel1")

```

```

qresult = scope.SCPI.TRIGger.EDGE.LEVel.Query("CHANnel1")
print "Trigger edge level: %s" % qresult

scope.SCPI.TRIGger.EDGE.SLOPe.Command("POSitive")
qresult = scope.SCPI.TRIGger.EDGE.SLOPe.Query()
print "Trigger edge slope: %s" % qresult

# Save oscilloscope setup.
setup_lines = scope.SCPI.SYSTem.SETup.Query()
nLength = len(setup_lines)
File.WriteAllLines("setup.stp", setup_lines)
print "Setup lines saved: %d" % nLength

# Change oscilloscope settings with individual commands:

# Set vertical scale and offset.
scope.SCPI.CHANnel.SCALE.Command(1, 0.05)
qresult = scope.SCPI.CHANnel.SCALE.Query(1)
print "Channel 1 vertical scale: %f" % qresult

scope.SCPI.CHANnel.OFFSet.Command(1, -1.5)
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 type.
scope.SCPI.ACQuire.TYPE.Command("NORMal")
qresult = scope.SCPI.ACQuire.TYPE.Query()
print "Acquire type: %s" % qresult

# Or, set up oscilloscope by loading a previously saved setup.
setup_lines = File.ReadAllLines("setup.stp")
scope.SCPI.SYSTem.SETup.Command(setup_lines)
print "Setup lines restored: %d" % len(setup_lines)

# Capture an acquisition using :DIGitize.
scope.SCPI.DIGitize.Command("CHANnel1", None, None, None, None)

# =====
# Analyze:
# =====
def analyze():

    # Make measurements.
    # -----
    scope.SCPI.MEASure.SOURce.Command("CHANnel1", None)
    (source1, source2) = scope.SCPI.MEASure.SOURce.Query()
    print "Measure source: %s" % source1

```

```

scope.SCPI.MEASure.FREQuency.Command("CHANnel1")
qresult = scope.SCPI.MEASure.FREQuency.Query("CHANnel1")
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.
# -----
scope.SCPI.HARDcopy.INKSaver.Command(False)

image_bytes = scope.SCPI.DISPlay.DATA.Query("PNG", "COLor")
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.
# ----

# Set the waveform points mode.
scope.SCPI.WAVEform.POINTs.MODE.Command("RAW")
qresult = scope.SCPI.WAVEform.POINTs.MODE.Query()
print "Waveform points mode: %s" % qresult

# Get the number of waveform points available.
scope.SCPI.WAVEform.POINTs.CommandPoints(10240)
qresult = scope.SCPI.WAVEform.POINTs.Query1()
print "Waveform points available: %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("BYTE")
qresult = scope.SCPI.WAVEform.FORMAT.Query()
print "Waveform format: %s" % qresult

# Display the waveform settings from preamble:
wav_form_dict = {
    0 : "BYTE",
    1 : "WORD",
    4 : "ASCII",
}
acq_type_dict = {
    0 : "NORMAL",
    1 : "PEAK",
    2 : "AVERage",
    3 : "HRESolution",
}

```

```

(
    wav_form, acq_type, wfmpnts, avgcnt, x_increment, x_origin,
    x_reference, y_increment, y_origin, y_reference
) = scope.SCPI.WAVEform.PREamble.Query()

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" % wfmpnts
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

# 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()
y_reference = scope.SCPI.WAVEform.YREFerence.Query()

# Get the waveform data.
data_bytes = scope.SCPI.WAVEform.DATA.QueryBYTE()
nLength = len(data_bytes)
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_bytes[i] - y_reference) * y_increment + y_origin
    writer.WriteLine("%E, %f" % (time_val, voltage))

# Close output file.
writer.Close()
print "Waveform format BYTE data written to %s." % strPath

# =====
# Main program:
# =====
#addr = "a-mx2024a-10029.cos.is.keysight.com"
addr = "TCPIP0::a-mx2024a-10029.cos.is.keysight.com::inst0::INSTR"
scope = AgInfiniiVision2000X(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)
```


Index

Symbols

+9.9E+37, infinity representation, 801
+9.9E+37, measurement error, 343

Numerics

0 (zero) values in waveform data, 639
1 (one) values in waveform data, 639
7000B Series oscilloscopes, command differences from, 37
82350A GPIB interface, 6

A

AC coupling, trigger edge, 604
AC input coupling for specified channel, 220
AC RMS measured on waveform, 375
accumulate activity, 149
ACQuire commands, 181
acquire data, 157, 193
acquire mode on autoscale, 153
acquire reset conditions, 134, 572
acquire sample rate, 192
ACQuire subsystem, 55
acquired data points, 186
acquisition count, 184
acquisition mode, 181, 185, 656
acquisition type, 181, 193
acquisition types, 631
active edges, 149
active printer, 300
activity logic levels, 149
activity on digital channels, 149
add function, 651
add math function, 291
add math function as g(t) source, 287
address field size, IIC serial decode, 472
address of network printer, 305
address, IIC trigger pattern, 475
Addresses softkey, 44
AER (Arm Event Register), 150, 165, 167, 777
ALB waveform data format, 29
all (snapshot) measurement, 344
ALL segments waveform save option, 445
AM demo signal, 238
AM depth, waveform generator modulation, 674

AM modulation type, waveform generator, 684
amplitude, vertical, 369
amplitude, waveform generator, 690
analog channel coupling, 220
analog channel display, 221
analog channel impedance, 222
analog channel input, 714
analog channel inversion, 223
analog channel labels, 224, 261
analog channel offset, 225
analog channel protection lock, 575
analog channel range, 232
analog channel scale, 233
analog channel source for glitch, 617
analog channel units, 234
analog channels only oscilloscopes, 5
analog probe attenuation, 226
analog probe head type, 227
analog probe sensing, 715
analog probe skew, 229, 713
analyzing captured data, 51
angle brackets, 117
annotate channels, 224
annotation background, display, 254
annotation color, display, 255
annotation text, display, 256
annotation, display, 253
apply network printer connection settings, 306
area for hardcopy print, 299
area for saved image, 748
Arm Event Register (AER), 150, 165, 167, 777
arrange waveforms, 717
ASCII format, 641
ASCII format for data transfer, 634
ASCII string, quoted, 117
ASCIixy waveform data format, 442
assign channel names, 224
attenuation factor (external trigger) probe, 275
attenuation for oscilloscope probe, 226
AUT option for probe sense, 715, 719
Auto Range capability for DVM, 266
auto set up, trigger level, 597
auto trigger sweep mode, 591
automask create, 385
automask source, 386
automask units, 387
automatic measurements constants, 226
automatic probe type detection, 715, 719
autoscale, 151

autoscale acquire mode, 153
autoscale channels, 154
AUToscale command, 54
average value measurement, 370
averaging acquisition type, 182, 633
averaging, synchronizing with, 790

B

bandwidth filter limits, 218, 274
band width filter limits to 20 MHz, 219
base value measurement, 371
base, UART trigger, 525
basic instrument functions, 121
baud rate, 462, 485, 514
begin acquisition, 157, 174, 176
BHARris window for minimal spectral leakage, 286
binary block data, 117, 258, 576, 639
BINary waveform data format, 442
bind levels for masks, 406
bit order, 515
bit order, SPI decode, 497
bit selection command, bus, 197
bit weights, 126
bitmap display, 258
bits in Service Request Enable Register, 139
bits in Standard Event Status Enable Register, 125
bits in Status Byte Register, 141
bits selection command, bus, 198
blank, 156
block data, 117, 129, 576
block response data, 58
blocking synchronization, 785
blocking wait, 784
BMP format screen image data, 258
braces, 116
built-in measurements, 51
burst data demo signal, 238
bus bit selection command, 197
bus bits selection commands, 198
bus clear command, 200
bus commands, 196
BUS data format, 636
bus display, 201
bus label command, 202
bus mask command, 203
BUS<n> commands, 195
button disable, 570
button, calibration protect, 210

byte format for data transfer, 635, 641
 BYTeorder, 637

C

C, SICL library example, 884
 C, VISA library example, 837
 C#, SCPI.NET example, 904
 C#, VISA COM example, 813
 C#, VISA example, 856
 CAL PROTECT button, 210
 CAL PROTECT switch, 206
 calculating preshoot of waveform, 359
 calculating the waveform overshoot, 355
 calibrate, 207, 208, 210, 214
 CALibrate commands, 205
 calibrate date, 207
 calibrate introduction, 206
 calibrate label, 208
 calibrate output, 209
 calibrate start, 211
 calibrate status, 212
 calibrate switch, 210
 calibrate temperature, 213
 calibrate time, 214
 CAN acknowledge, 461
 CAN baud rate, 462
 CAN frame counters, reset, 458
 CAN SEARch commands, 538
 CAN serial bus commands, 454
 CAN serial search, data, 540
 CAN serial search, data length, 541
 CAN serial search, ID, 542
 CAN serial search, ID mode, 543
 CAN serial search, mode, 539
 CAN signal definition, 463
 CAN source, 464
 CAN trigger, 465, 468
 CAN trigger data pattern, 467
 CAN trigger ID pattern, 469
 CAN trigger pattern id mode, 470
 CAN triggering, 449
 capture data, 157
 capturing data, 50
 center frequency set, 280, 283
 center of screen, 664
 center reference, 584
 center screen, vertical value at, 290, 293
 channel, 180, 224, 710, 712
 channel coupling, 220
 channel display, 221
 channel input impedance, 222
 channel inversion, 223
 channel label, 224, 711
 channel labels, 260, 261
 channel numbers, 717
 channel overload, 231
 channel protection, 231
 channel reset conditions, 134, 572
 channel selected to produce trigger, 617, 627

channel signal type, 230
 channel skew for oscilloscope probe, 229, 713
 channel status, 177, 717
 channel threshold, 712
 channel vernier, 235
 channel, stop displaying, 156
 CHANnel<n> commands, 215, 217
 channels to autoscale, 154
 channels, how autoscale affects, 151
 characters to display, 568
 classes of input signals, 286
 classifications, command, 794
 clear, 257
 clear bus command, 200
 clear cumulative edge variables, 710
 clear markers, 345, 728
 clear measurement, 345, 728
 clear message queue, 123
 Clear method, 53
 clear reference waveforms, 697
 clear screen, 718
 clear status, 123
 clear waveform area, 252
 clipped high waveform data value, 639
 clipped low waveform data value, 639
 clock, 473, 498, 501
 clock timeout, SPI, 499
 clock with infrequent glitch demo signal, 238
 CLS (Clear Status), 123
 CME (Command Error) status bit, 125, 127
 CMOS threshold voltage for digital channels, 249, 712
 CMOS trigger threshold voltage, 750
 code, :ACQuire:COMplete, 183
 code, :ACQuire:SEGmented, 189
 code, :ACQuire:TYPE, 194
 code, :AUToscale, 152
 code, :CHANnel<n>:LAbel, 224
 code, :CHANnel<n>:PROBe, 226
 code, :CHANnel<n>:RANGe, 232
 code, :DIGItize, 157
 code, :DISPlay:DATA, 258
 code, :DISPlay:LABEL, 260
 code, :DISPlay:ORDER, 717
 code, :MEASure:PERiod, 364
 code, :MEASure:TEDGE, 366
 code, :MTESt, 381
 code, :POD<n>:THRESHold, 417
 code, :RUN/:STOP, 174
 code, :SYSTem:SETup, 576
 code, :TIMEbase:DElay, 749
 code, :TIMEbase:MODE, 581
 code, :TIMEbase:RANGE, 583
 code, :TIMEbase:REFERENCE, 584
 code, :TRIGger:MODE, 600
 code, :TRIGger:SLOPe, 607
 code, :TRIGger:SOURce, 608
 code, :VIEW and :BLANK, 180
 code, :WAVEform, 652
 code, :WAVEform:DATA, 639
 code, :WAVEform:POINts, 643
 code, :WAVEform:PREamble, 647
 code, :WAVEform:SEGmented, 189
 code, *RST, 136
 code, SCPI.NET library example in C#, 904
 code, SCPI.NET library example in IronPython, 916
 code, SCPI.NET library example in Visual Basic .NET, 910
 code, SICL library example in C, 884
 code, SICL library example in Visual Basic, 893
 code, VISA COM library example in C#, 813
 code, VISA COM library example in Python, 830
 code, VISA COM library example in Visual Basic, 804
 code, VISA COM library example in Visual Basic .NET, 822
 code, VISA library example in C, 837
 code, VISA library example in C#, 856
 code, VISA library example in Python, 877
 code, VISA library example in Visual Basic, 846
 code, VISA library example in Visual Basic .NET, 867
 colon, root commands prefixed by, 148
 color palette for hardcopy, 311
 color palette for image, 435
 Comma Separated Values (CSV) waveform data format, 442
 command classifications, 794
 command differences from 7000B Series oscilloscopes, 37
 command errors detected in Standard Event Status, 127
 Command Expert, 904
 command header, 795
 command headers, common, 797
 command headers, compound, 797
 command headers, simple, 797
 command strings, valid, 795
 commands quick reference, 63
 commands sent over interface, 121
 commands, more about, 793
 commands, obsolete and discontinued, 705
 common (*) commands, 3, 119, 121
 common command headers, 797
 completion criteria for an acquisition, 183, 184
 compound command headers, 797
 compound header, 799
 computer control examples, 803
 conditions for external trigger, 273
 conditions, reset, 134, 572
 Config softkey, 44
 configurations, oscilloscope, 129, 133, 137, 576
 Configure softkey, 44
 connect oscilloscope, 43
 connect sampled data points, 716

Connection Expert, 45
 constants for making automatic measurements, 226
 constants for scaling display factors, 226
 constants for setting trigger levels, 226
 controller initialization, 50
 copy display, 173
 copyright, 2
 core commands, 794
 count, 638
 count values, 184
 coupling, 604
 coupling for channels, 220
 create automask, 385
 CSV (Comma Separated Values) waveform data format, 442
 cumulative edge activity, 710
 current logic levels on digital channels, 149
 current oscilloscope configuration, 129, 133, 137, 576
 current probe, 234, 277
 CURRent segment waveform save option, 445
 cursor mode, 321
 cursor position, 322, 324, 326, 329, 331
 cursor readout, 729, 731, 732
 cursor reset conditions, 134, 572
 cursor source, 323, 325
 cursor time, 729, 731, 732
 cursor units, X, 327, 328
 cursor units, Y, 332, 333
 cursors track measurements, 362
 cursors, how autoscale affects, 151
 cursors, X1, X2, Y1, Y2, 320
 cycle measured, 351
 cycle time, 357

D

data, 474, 476, 639
 data (waveform) maximum length, 444
 data 2, 477
 data acquisition types, 631
 data conversion, 633
 data format for transfer, 634
 data output order, 637
 data pattern length, 468, 493
 data pattern, CAN trigger, 467
 data point index, 661
 data points, 186
 data record, measurement, 644
 data record, raw acquisition, 644
 data required to fill time buckets, 183
 data structures, status reporting, 763
 data, saving and recalling, 252
 date, calibration, 207
 date, system, 567
 dB versus frequency, 280
 DC coupling for edge trigger, 604

DC input coupling for specified channel, 220
 DC RMS measured on waveform, 375
 DC waveform generator output, 670
 DDE (Device Dependent Error) status bit, 125, 127
 decision chart, status reporting, 781
 default conditions, 134, 572
 define channel labels, 224
 define glitch trigger, 615
 define logic thresholds, 712
 define measurement, 347
 define measurement source, 363
 define trigger, 616
 defined as, 116
 definite-length block query response, 58
 definite-length block response data, 117
 delay measured to calculate phase, 358
 delay measurement, 347
 delay measurements, 365
 delay parameters for measurement, 349
 delay, how autoscale affects, 151
 delayed time base, 581
 delayed window horizontal scale, 589
 delete mask, 395
 delta time, 729
 delta voltage measurement, 735
 delta X cursor, 320
 delta Y cursor, 320
 demo, 237
 DEMO commands, 237
 demo signal function, 238
 demo signal phase angle, 240
 demo signals output control, 241
 detecting probe types, 715, 719
 device-defined error queue clear, 123
 differences from 7000B Series oscilloscope commands, 37
 differential probe heads, 227
 differential signal type, 230
 digital channel commands, 244, 245, 246, 247, 249
 digital channel data, 635
 digital channel labels, 261
 digital channel order, 717
 digital channel source for glitch trigger, 617
 digital channels, 5
 digital channels, activity and logic levels on, 149
 digital channels, groups of, 413, 415, 417
 digital pod, stop displaying, 156
 digital reset conditions, 135, 573
 DIGItal<d> commands, 243
 digitize channels, 157
 DIGItize command, 50, 55, 632
 digits, 117
 disable front panel, 570
 disable function, 721
 disabling calibration, 210
 disabling channel display, 221
 disabling status register bits, 124, 138

discontinued and obsolete commands, 705
 display annotation, 253
 display annotation background, 254
 display annotation color, 255
 display annotation text, 256
 display channel labels, 260
 display clear, 257
 DISPlay commands, 251
 display commands introduction, 252
 display connect, 716
 display date, 567
 display factors scaling, 226
 display for channels, 221
 display frequency span, 284
 display measurements, 342, 362
 display order, 717
 display persistence, 262
 display reference, 582, 584
 display reference waveforms, 698
 display reset conditions, 135, 573
 display serial number, 175
 display vectors, 263
 display wave position, 717
 display, lister, 317
 display, oscilloscope, 245, 262, 282, 415, 568
 display, serial decode bus, 452
 displaying a baseline, 602
 displaying unsynchronized signal, 602
 DNS IP, 44
 domain, 44
 domain, network printer, 307
 driver, printer, 726
 DSO models, 5
 duplicate mnemonics, 799
 duration for glitch trigger, 611, 612, 616
 duration triggering, 591
 duty cycle measurement, 51, 342, 351
 DVM commands, 265
 DVM displayed value, 267
 DVM enable/disable, 268
 DVM frequency value, 269
 DVM input source, 271
 DVM mode, 270

E

ECL channel threshold, 712
 ECL threshold voltage for digital channels, 249
 ECL trigger threshold voltage, 750
 edge activity, 710
 edge coupling, 604
 edge fall time, 352
 edge parameter for delay measurement, 349
 edge preshoot measured, 359
 edge rise time, 361
 edge slope, 607
 edge source, 608
 EDGE trigger commands, 603

edge triggering, 591
edges (activity) on digital channels, 149
edges in measurement, 347
elapsed time in mask test, 392
ellipsis, 117
enable channel labels, 260
enabling calibration, 210
enabling channel display, 221
enabling status register bits, 124, 138
end of string (EOS) terminator, 796
end of text (EOT) terminator, 796
end or identify (EOI), 796
EOI (end or identify), 796
EOS (end of string) terminator, 796
EOT (end of text) terminator, 796
erase data, 257
erase measurements, 728
erase screen, 718
error frame count (CAN), 456
error frame count (UART), 516
error messages, 569, 753
error number, 569
error queue, 569, 774
error, measurement, 342
ESB (Event Status Bit), 139, 141
ESE (Standard Event Status Enable Register), 124, 773
ESR (Standard Event Status Register), 126, 772
event status conditions occurred, 141
Event Status Enable Register (ESE), 124, 773
Event Status Register (ESR), 126, 179, 772
example code, :ACQuire:COMPLETE, 183
example code, :ACQuire:SEGmented, 189
example code, :ACQuire:TYPE, 194
example code, :AUToscale, 152
example code, :CHANnel<n>:LABEL, 224
example code, :CHANnel<n>:PROBe, 226
example code, :CHANnel<n>:RANGe, 232
example code, :DIGitize, 157
example code, :DISPLAY:DATA, 258
example code, :DISPLAY:LABEL, 260
example code, :DISPLAY:ORDer, 717
example code, :MEASure:PERiod, 364
example code, :MEASure:TEDGE, 366
example code, :MTEST, 381
example code, :POD<n>:THRESHold, 417
example code, :RUN:/STOP, 174
example code, :SYSTem:SETup, 576
example code, :TIMEbase:DELay, 749
example code, :TIMEbase:MODE, 581
example code, :TIMEbase:RANGE, 583
example code, :TIMEbase:REFERENCE, 584
example code, :TRIGger:MODE, 600
example code, :TRIGger:SLOPe, 607
example code, :TRIGger:SOURce, 608
example code, :VIEW and :BLANK, 180
example code, :WAVEform, 652
example code, :WAVEform:DATA, 639
example code, :WAVEform:POINTS, 643
example code, :WAVEform:PREamble, 647

example code,
 :WAVEform:SEGmented, 189
example code, *RST, 136
example programs, 5, 803
EXE (Execution Error) status bit, 125, 127
execution error detected in Standard Event Status, 127
exponential notation, 116
external glitch trigger source, 617
external range, 276
external trigger, 273, 275, 608
EXternal trigger commands, 273
EXternal trigger level, 605
external trigger probe attenuation factor, 275
external trigger probe sensing, 719
EXternal trigger source, 608
external trigger units, 277

F

failed waveforms in mask test, 390
failure, self test, 143
fall time measurement, 342, 352
Fast Fourier Transform (FFT)
 functions, 280, 283, 284, 286, 296, 720
FF values in waveform data, 639
FFT (Fast Fourier Transform) functions, 280, 283, 284, 286, 296, 720
FFT (Fast Fourier Transform)
 operation, 291, 651
FFT vertical units, 285
fifty ohm impedance, disable setting, 575
filename for hardcopy, 723
filename for recall, 421, 668
filename for save, 430
filter for frequency reject, 606
filter for high frequency reject, 595
filter for noise reject, 601
filter used to limit bandwidth, 219, 274
filters to Fast Fourier Transforms, 286
fine horizontal adjustment (vernier), 586
fine vertical adjustment (vernier), 235
finish pending device operations, 130
first point displayed, 661
FLATtop window for amplitude measurements, 286
FM burst demo signal, 239
FM modulation type, waveform generator, 684
force trigger, 594
format, 641, 646
format for block data, 129
format for hardcopy, 722
format for image, 433
format for waveform data, 442
FormattedIO488 object, 53
formfeed for hardcopy, 298, 302
formulas for data conversion, 633
frame, 502

frame counters (CAN), error, 456
frame counters (CAN), overload, 457
frame counters (CAN), reset, 458
frame counters (CAN), total, 459
frame counters (UART), error, 516
frame counters (UART), reset, 517
frame counters (UART), Rx frames, 518
frame counters (UART), Tx frames, 519
framing, 500
frequency deviation, waveform generator FM modulation, 676
frequency measurement, 51, 342, 353
frequency measurements with X cursors, 327
frequency resolution, 286
frequency span of display, 284
frequency versus dB, 280
front panel mode, 602
front panel Single key, 176
front panel Stop key, 178
front-panel lock, 570
FSK modulation type, waveform generator, 684
FSK rate, waveform generator modulation, 679
full-scale horizontal time, 583, 588
full-scale vertical axis defined, 292
function, 180, 282, 283, 284, 286, 290, 291, 292, 293, 294, 720, 721
FUNCTION commands, 279
function memory, 177
function turned on or off, 721
function, demo signal, 238
function, waveform generator, 669
functions, 651

G

g(t) source, first input channel, 288
g(t) source, math operation, 287
g(t) source, second input channel, 289
gateway IP, 44
general SBUS<n> commands, 451
general SEARch commands, 534
general trigger commands, 593
glitch demo signal, 238
glitch duration, 616
glitch qualifier, 615
glitch source, 617
GLITCH trigger commands, 609
glitch trigger duration, 611
glitch trigger polarity, 614
glitch trigger source, 611
GPIB interface, 43, 44
graticule area for hardcopy print, 299
graticule colors, invert for hardcopy, 303, 725
graticule colors, invert for image, 434
grayscale palette for hardcopy, 311
grayscale palette for image, 435
grayscale on hardcopy, 724

greater than qualifier, 615
 greater than time, 611, 616
 groups of digital channels, 413, 415, 417, 712

H

HANNing window for frequency resolution, 286
 hardcopy, 173, 298
 HARDcopy commands, 297
 hardcopy factors, 301, 432
 hardcopy filename, 723
 hardcopy format, 722
 hardcopy formfeed, 302
 hardcopy grayscale, 724
 hardcopy invert graticule colors, 303, 725
 hardcopy layout, 304
 hardcopy palette, 311
 hardcopy print, area, 299
 hardcopy printer driver, 726
 head type, probe, 227
 header, 795
 high resolution acquisition type, 633
 high trigger level, 598
 high-frequency reject filter, 595, 606
 high-level voltage, waveform generator, 691
 high-resolution acquisition type, 182
 hold until operation complete, 130
 holdoff time, 596
 holes in waveform data, 639
 hop frequency, waveform generator FSK modulation, 678
 horizontal adjustment, fine (vernier), 586
 horizontal position, 587
 horizontal scale, 585, 589
 horizontal scaling, 646
 horizontal time, 583, 588, 729
 Host name softkey, 44
 hostname, 44

I

id mode, 470
 ID pattern, CAN trigger, 469
 identification number, 128
 identification of options, 131
 identifier, LIN, 490
 idle until operation complete, 130
 IDN (Identification Number), 128
 IEEE 488.2 standard, 121
 IIC address, 475
 IIC clock, 473
 IIC data, 474, 476
 IIC data 2, 477
 IIC SEARch commands, 544
 IIC serial decode address field size, 472
 IIC serial search, address, 547
 IIC serial search, data, 548

IIC serial search, data2, 549
 IIC serial search, mode, 545
 IIC serial search, qualifier, 550
 IIC trigger commands, 471
 IIC trigger qualifier, 478
 IIC trigger type, 479
 IIC triggering, 449
 image format, 433
 image invert graticule colors, 434
 image memory, 177
 image palette, 435
 image, save, 431
 image, save with inksaver, 434
 impedance, 222
 infinity representation, 801
 initialization, 50, 53
 initialize, 134, 572
 initialize label list, 261
 initiate acquisition, 157
 inksaver, save image with, 434
 input coupling for channels, 220
 input impedance for channels, 222, 714
 input inversion for specified channel, 223
 insert label, 224
 installed options identified, 131
 instruction header, 795
 instrument number, 128
 instrument options identified, 131
 instrument requests service, 141
 instrument serial number, 175
 instrument settings, 298
 instrument status, 60
 instrument type, 128
 intensity, waveform, 259
 internal low-pass filter, 218, 219, 274
 introduction to :ACQuire commands, 181
 introduction to :BUS<n> commands, 196
 introduction to :CALibrate commands, 206
 introduction to :CHANnel<n> commands, 217
 introduction to :DEMO commands, 237
 introduction to :DIGItal<d> commands, 244
 introduction to :DISPlay commands, 252
 introduction to :EXTernal commands, 273
 introduction to :FUNCtion commands, 280
 introduction to :HARDcopy commands, 298
 introduction to :LISTer commands, 315
 introduction to :MARKer commands, 320
 introduction to :MEASure commands, 342
 introduction to :POD<n> commands, 413
 introduction to :RECall commands, 419
 introduction to :SAVE commands, 429
 introduction to :SBUS commands, 449
 introduction to :SYSTem commands, 566
 introduction to :TIMEbase commands, 580
 introduction to :TRIGger commands, 591
 introduction to :WAVEform commands, 631
 introduction to :WGEN commands, 667

introduction to :WMEMory<r> commands, 695
 introduction to common (*) commands, 121
 introduction to root () commands, 148
 invert graticule colors for hardcopy, 303, 725
 invert graticule colors for image, 434
 inverted masks, bind levels, 406
 inverting input for channels, 223
 IO library, referencing, 52
 IP address, 44
 IronPython, SCPI.NET example, 916

K

key disable, 570
 key press detected in Standard Event Status Register, 127
 Keysight Interactive IO application, 47
 Keysight IO Control icon, 45
 Keysight IO Libraries Suite, 6, 41, 52, 54
 Keysight IO Libraries Suite, installing, 42
 knob disable, 570
 known state, 134, 572

L

label, 711
 label command, bus, 202
 label list, 224, 261
 label reference waveforms, 699
 label, digital channel, 246
 labels, 224, 260, 261
 labels to store calibration information, 208
 labels, specifying, 252
 LAN instrument, 46
 LAN interface, 43, 45
 LAN Settings softkey, 44
 landscape layout for hardcopy, 304
 language for program examples, 49
 layout for hardcopy, 304
 leakage into peak spectrum, 286
 learn string, 129, 576
 least significant byte first, 637
 left reference, 584
 legal values for channel offset, 225
 legal values for frequency span, 284
 legal values for offset, 290, 293
 length for waveform data, 443
 less than qualifier, 615
 less than time, 612, 616
 level for trigger voltage, 605, 613
 LF coupling, 604
 license information, 131
 limit bandwidth, 218
 limits for line number, 624
 LIN acknowledge, 484
 LIN baud rate, 485
 LIN identifier, 490

LIN pattern data, 491
 LIN pattern format, 494
 LIN SEARch commands, 551
 LIN serial decode bus parity bits, 483
 LIN serial search, data, 554
 LIN serial search, data format, 556
 LIN serial search, data length, 555
 LIN serial search, frame ID, 552
 LIN serial search, mode, 553
 LIN source, 486
 LIN standard, 487
 LIN sync break, 488
 LIN trigger, 489, 493
 LIN trigger commands, 481
 LIN triggering, 449
 line glitch trigger source, 617
 line number for TV trigger, 624
 line terminator, 116
 LINE trigger level, 605
 LINE trigger source, 608
 list of channel labels, 261
 LISTer commands, 315
 lister display, 317
 lister time reference, 318
 load utilization (CAN), 460
 local lockout, 570
 lock, 570
 lock mask to signal, 397
 lock, analog channel protection, 575
 lockout message, 570
 logic level activity, 710
 long form, 796
 low frequency sine with glitch demo signal, 239
 low trigger level, 599
 lower threshold, 357
 lower threshold voltage for measurement, 727
 lowercase characters in commands, 795
 low-frequency reject filter, 606
 low-level voltage, waveform generator, 692
 low-pass filter used to limit bandwidth, 218, 219, 274
 LRN (Learn Device Setup), 129
 lsbfirst, 637

M

magnitude of occurrence, 367
 main sweep range, 587
 main time base, 749
 main time base mode, 581
 making measurements, 342
 MAN option for probe sense, 715, 719
 manual cursor mode, 321
 MARKer commands, 319
 marker mode, 329
 marker position, 330
 marker readout, 731, 732

marker set for voltage measurement, 736, 737
 marker sets start time, 730
 marker time, 729
 markers for delta voltage measurement, 735
 markers track measurements, 362
 markers, command overview, 320
 markers, mode, 321
 markers, time at start, 732
 markers, time at stop, 731
 markers, X delta, 326
 markers, X1 position, 322
 markers, X1Y1 source, 323
 markers, X2 position, 324
 markers, X2Y2 source, 325
 markers, Y delta, 331
 markers, Y1 position, 329
 markers, Y2 position, 330
 mask, 124, 138
 mask command, bus, 203
 mask statistics, reset, 391
 mask test commands, 379
 Mask Test Event Enable Register (MTEenable), 159
 mask test event event register, 161
 Mask Test Event Event Register (:MTERegister[:EVENTn]), 161, 779
 mask test run mode, 398
 mask test termination conditions, 398
 mask test, all channels, 384
 mask test, enable/disable, 396
 mask, delete, 395
 mask, get as binary block data, 394
 mask, load from binary block data, 394
 mask, lock to signal, 397
 mask, recall, 422
 mask, save, 436, 437
 masks, bind levels, 406
 master summary status bit, 141
 math function, stop displaying, 156
 math operations, 280
 MAV (Message Available), 123, 139, 141
 maximum duration, 612
 maximum position, 582
 maximum range for zoomed window, 588
 maximum scale for zoomed window, 589
 maximum vertical value measurement, 372
 maximum waveform data length, 444
 MEASure commands, 335
 measure mask test failures, 399
 measure overshoot, 355
 measure period, 357
 measure phase between channels, 358
 measure preshoot, 359
 measure start voltage, 736
 measure stop voltage, 737
 measure value at a specified time, 376
 measure value at top of waveform, 377
 measurement error, 342
 measurement record, 644
 measurement setup, 342, 363

measurement source, 363
 measurement window, 378
 measurements, AC RMS, 375
 measurements, average value, 370
 measurements, base value, 371
 measurements, built-in, 51
 measurements, clear, 345, 728
 measurements, command overview, 342
 measurements, DC RMS, 375
 measurements, definition setup, 347
 measurements, delay, 349
 measurements, duty cycle, 351
 measurements, fall time, 352
 measurements, frequency, 353
 measurements, how autoscale affects, 151
 measurements, lower threshold level, 727
 measurements, maximum vertical value, 372
 measurements, minimum vertical value, 373
 measurements, overshoot, 355
 measurements, period, 357
 measurements, phase, 358
 measurements, preshoot, 359
 measurements, pulse width, negative, 354
 measurements, pulse width, positive, 360
 measurements, rise time, 361
 measurements, show, 362
 measurements, snapshot all, 344
 measurements, source channel, 363
 measurements, start marker time, 731
 measurements, stop marker time, 732
 measurements, thresholds, 730
 measurements, time between start and stop markers, 729
 measurements, time between trigger and edge, 365
 measurements, time between trigger and vertical value, 367
 measurements, time between trigger and voltage level, 733
 measurements, upper threshold value, 734
 measurements, vertical amplitude, 369
 measurements, vertical peak-to-peak, 374
 measurements, voltage difference, 735
 memory setup, 137, 576
 menu, system, 571
 message available bit, 141
 message available bit clear, 123
 message displayed, 141
 message error, 753
 message queue, 771
 messages ready, 141
 midpoint of thresholds, 357
 minimum duration, 611
 minimum vertical value measurement, 373
 MISO data pattern width, 506
 MISO data pattern, SPI trigger, 505
 MISO data source, SPI trigger, 503
 MISO data, SPI, 655
 mixed-signal demo signals, 239
 mixed-signal oscilloscopes, 5

mnemonics, duplicate, 799
mode, 321, 581
mode, serial decode, 453
model number, 128
models, oscilloscope, 3
modes for triggering, 600
Modify softkey, 44
modulating signal frequency, waveform generator, 675, 677
modulation (waveform generator), enabling/disabling, 683
modulation type, waveform generator, 684
MOSI data pattern width, 508
MOSI data pattern, SPI trigger, 507
MOSI data source, SPI trigger, 504
most significant byte first, 637
move cursors, 731, 732
msbfirst, 637
MSG (Message), 139, 141
MSO models, 5
MSS (Master Summary Status), 141
MTEnable (Mask Test Event Enable Register), 159
MTERegister[:EVENT] (Mask Test Event Event Register), 161, 779
MTEST commands, 379
multi-channel waveform data, save, 438
multiple commands, 799
multiple queries, 59
multiply math function, 280, 291, 651
multiply math function as g(t) source, 287

N

N8900A InfiniiView oscilloscope analysis software, 438
name channels, 224
name list, 261
negative glitch trigger polarity, 614
negative pulse width, 354
negative pulse width measurement, 51
negative slope, 498, 607
negative TV trigger polarity, 626
network domain password, 308
network domain user name, 310
network printer address, 305
network printer domain, 307
network printer slot, 309
network printer, apply connection settings, 306
new line (NL) terminator, 116, 796
NL (new line) terminator, 116, 796
noise reject filter, 601
noise waveform generator output, 670
noise, adding to waveform generator output, 682
noisy sine waveform demo signal, 238
non-core commands, 794
non-volatile memory, label list, 202, 246, 261
normal acquisition type, 181, 632

normal trigger sweep mode, 591
notices, 2
NR1 number format, 116
NR3 number format, 116
NTSC, 624, 628
NULL string, 568
number format, 116
number of points, 186, 642, 644
number of time buckets, 642, 644
numeric variables, 58
numeric variables, reading query results into multiple, 60
nwidth, 354

O

obsolete and discontinued commands, 705
obsolete commands, 794
occurrence reported by magnitude, 733
offset value for channel voltage, 225
offset value for selected function, 290, 293
offset, waveform generator, 693
one values in waveform data, 639
OPC (Operation Complete) command, 130
OPC (Operation Complete) status bit, 125, 127
OPEE (Operation Status Enable Register), 163
Open method, 53
operating configuration, 129, 576
operating state, 137
operation complete, 130
operation status condition register, 165
Operation Status Condition Register (:OPERRegister:CONDITION), 165, 776
operation status conditions occurred, 141
Operation Status Enable Register (OPEE), 163
operation status event register, 167
Operation Status Event Register (:OPERRegister[:EVENT]), 167, 775
operation, math, 280
operations for function, 291
OPERRegister:CONDITION (Operation Status Condition Register), 165, 776
OPERRegister[:EVENT] (Operation Status Event Register), 167, 775
OPT (Option Identification), 131
optional syntax terms, 116
options, 131
order of digital channels on display, 717
order of output, 637
oscilloscope connection, opening, 53
oscilloscope connection, verifying, 45
oscilloscope external trigger, 273
oscilloscope models, 3
oscilloscope rate, 192
oscilloscope, connecting, 43
oscilloscope, initialization, 50
oscilloscope, operation, 6
oscilloscope, program structure, 50

oscilloscope, setting up, 43
oscilloscope, setup, 54
output control, demo signals, 241
output control, waveform generator, 686
output load impedance, waveform generator, 687
output messages ready, 141
output queue, 130, 770
output queue clear, 123
output sequence, 637
overlapped commands, 802
overload, 231
Overload Event Enable Register (OVL), 169
Overload Event Register (:OVLRegister), 778
Overload Event Register (OVLR), 171
overload frame count (CAN), 457
overload protection, 169, 171
overshoot of waveform, 355
overvoltage, 231
OVL (Overload Event Enable Register), 169
OVLR (Overload Event Register), 171
OVL bit, 165, 167
OVLRegister (Overload Event Register), 778

P

PAL, 624, 628
palette for hardcopy, 311
palette for image, 435
PAL-M, 624, 628
parameters for delay measurement, 349
parametric measurements, 342
parity, 521
parity bits, LIN serial decode bus, 483
parser, 148, 799
pass, self test, 143
password, network domain, 308
path information, recall, 423
path information, save, 439
pattern, 475, 476, 477
pattern data, LIN, 491
pattern duration, 611, 612
pattern for pattern trigger, 619
pattern format, LIN, 494
pattern length, 468, 493
PATTern trigger commands, 618
pattern trigger format, 621
pattern trigger qualifier, 622
pattern triggering, 591
pattern width, 506, 508
peak data, 633
peak detect, 193
peak detect acquisition type, 182, 633
peak-to-peak vertical value measurement, 374
pending operations, 130
percent of waveform overshoot, 355
percent thresholds, 347
period measured to calculate phase, 358
period measurement, 51, 342, 357

period, waveform generator, 688
 persistence, waveform, 252, 262
 phase angle, demo signals, 240
 phase measured between channels, 358
 phase measurements, 365
 phase measurements with X cursors, 327
 phase shifted demo signals, 238
 PNG format screen image data, 258
 pod, 413, 415, 416, 417, 651, 712
 POD commands, 413
 POD data format, 635
 pod, stop displaying, 156
 points, 186, 642, 644
 points in waveform data, 632
 polarity, 522, 626
 polarity for glitch trigger, 614
 polling synchronization with timeout, 786
 polling wait, 784
 PON (Power On) status bit, 125, 127
 portrait layout for hardcopy, 304
 position, 247, 324, 582, 587
 position cursors, 731, 732
 position in zoomed view, 587
 position waveforms, 717
 positive glitch trigger polarity, 614
 positive pulse width, 360
 positive pulse width measurement, 51
 positive slope, 498, 607
 positive TV trigger polarity, 626
 positive width, 360
 preamble data, 646
 preamble metadata, 631
 predefined logic threshold, 712
 predefined threshold voltages, 750
 present working directory, recall
 operations, 423
 present working directory, save
 operations, 439
 preset conditions, 572
 preshoot measured on waveform, 359
 previously stored configuration, 133
 print command, 173
 print job, start, 313
 print mask test failures, 400
 print query, 746
 printer driver for hardcopy, 726
 printer, active, 300
 printing, 298
 printing in grayscale, 724
 probe, 605
 probe attenuation affects channel voltage
 range, 232
 probe attenuation factor (external
 trigger), 275
 probe attenuation factor for selected
 channel, 226
 probe head type, 227
 probe ID, 228
 probe sense for oscilloscope, 715, 719
 probe skew value, 229, 713
 process sigma, mask test run, 403
 program data, 796

program data syntax rules, 798
 program initialization, 50
 program message, 53, 121
 program message syntax, 795
 program message terminator, 796
 program structure, 50
 programming examples, 5, 803
 protecting against calibration, 210
 protection, 169, 171, 231
 protection lock, 575
 pulse waveform generator output, 670
 pulse width, 354, 360
 pulse width duration trigger, 611, 612,
 616
 pulse width measurement, 51, 342
 pulse width trigger, 601
 pulse width trigger level, 613
 pulse width triggering, 591
 pulse width, waveform generator, 671
 pwidth, 360
 Python, VISA COM example, 830
 Python, VISA example, 877

Q

qualifier, 616
 qualifier, trigger pattern, 622
 queries, multiple, 59
 query error detected in Standard Event
 Status, 127
 query responses, block data, 58
 query responses, reading, 57
 query results, reading into numeric
 variables, 58
 query results, reading into string
 variables, 58
 query return values, 801
 query setup, 298, 320, 342, 576
 query subsystem, 196, 244
 querying setup, 217
 querying the subsystem, 592
 queues, clearing, 780
 quick reference, commands, 63
 quoted ASCII string, 117
 QYE (Query Error) status bit, 125, 127

R

ramp symmetry, waveform generator, 672
 ramp symmetry, waveform generator
 modulating signal, 681
 ramp waveform generator output, 669
 range, 588
 range for channels, 232
 range for external trigger, 276
 range for full-scale vertical axis, 292
 range for glitch trigger, 616
 range for time base, 583
 range of offset values, 225
 range qualifier, 615
 ranges, value, 117
 rate, 192
 ratio measurements with X cursors, 327
 ratio measurements with Y cursors, 332
 raw acquisition record, 644
 RCL (Recall), 133
 read configuration, 129
 ReadIEEEBlock method, 53, 57, 59
 ReadList method, 53, 57
 ReadNumber method, 53, 57
 readout, 729
 ReadString method, 53, 57
 real-time acquisition mode, 185
 recall, 133, 419, 576
 RECall commands, 419
 recall filename, 421, 668
 recall mask, 422
 recall path information, 423
 recall reference waveform, 425
 recall setup, 424
 recalling and saving data, 252
 RECTangular window for transient
 signals, 286
 reference, 584
 reference for time base, 749
 reference waveform save source, 446
 reference waveform, recall, 425
 reference waveform, save, 447
 reference waveforms, clear, 697
 reference waveforms, display, 698
 reference waveforms, label, 699
 reference waveforms, save to, 700
 reference waveforms, skew, 701
 reference waveforms, Y offset, 702
 reference waveforms, Y range, 703
 reference waveforms, Y scale, 704
 reference, lister, 318
 registers, 126, 133, 137, 150, 159, 161,
 163, 165, 167, 169, 171
 registers, clearing, 780
 reject filter, 606
 reject high frequency, 595
 reject noise, 601
 remote control examples, 803
 remove cursor information, 321
 remove labels, 260
 remove message from display, 568
 reorder channels, 151
 repetitive acquisitions, 174
 report errors, 569
 report transition, 365, 367
 reporting status, 761
 reporting the setup, 592
 request service, 141
 Request-for-OPC flag clear, 123
 reset, 134
 reset conditions, 134
 reset defaults, waveform generator, 689
 reset mask statistics, 391
 reset measurements, 257
 resolution of printed copy, 724
 resource session object, 53

ResourceManager object, 53
 restore configurations, 129, 133, 137, 576
 restore labels, 260
 restore setup, 133
 return values, query, 801
 returning acquisition type, 193
 returning number of data points, 186
 RF burst demo signal, 239
 right reference, 584
 ringing pulse demo signal, 238
 rise time measurement, 342
 rise time of positive edge, 361
 RMS value measurement, 375
 roll time base mode, 581
 root () commands, 145, 148
 root level commands, 3
 RQL (Request Control) status bit, 125, 127
 RQS (Request Service), 141
 RS-232/UART triggering, 450
 RST (Reset), 134
 rules, tree traversal, 799
 rules, truncation, 796
 run, 142, 174
 Run bit, 165, 167
 run mode, mask test, 398
 running configuration, 137, 576
 Rx frame count (UART), 518
 Rx source, 523

S

sample rate, 192
 sampled data, 716
 sampled data points, 639
 SAV (Save), 137
 save, 137, 429
 SAVE commands, 427
 save filename, 430
 save image, 431
 save image with inksaver, 434
 save mask, 436, 437
 save mask test failures, 401
 save path information, 439
 save reference waveform, 447
 save setup, 440
 save to reference waveform location, 700
 save waveform data, 441
 saved image, area, 748
 saving and recalling data, 252
 SBUS CAN commands, 454
 SBUS commands, 449
 SBUS<n> commands, general, 451
 scale, 294, 585, 589
 scale factors output on hardcopy, 301, 432
 scale for channels, 233
 scale units for channels, 234
 scale units for external trigger, 277
 scaling display factors, 226
 SCPI commands, 61
 SCPI.NET example in C#, 904
 SCPI.NET example in IronPython, 916

SCPI.NET example in Visual Basic .NET, 910
 SCPI.NET examples, 904
 scratch measurements, 728
 screen area for hardcopy print, 299
 screen area for saved image, 748
 screen image data, 258
 SEARch commands, 533
 SEARch commands, CAN, 538
 SEARch commands, general, 534
 SEARch commands, IIC, 544
 SEARch commands, LIN, 551
 SEARch commands, SPI, 557
 SEARch commands, UART, 561
 search mode, 536
 search state, 537
 SECAM, 624, 628
 seconds per division, 585
 segmented waveform save option, 445
 segments, analyze, 187
 segments, count of waveform, 649
 segments, setting number of memory, 188
 segments, setting the index, 189
 segments, time tag, 650
 select measurement channel, 363
 self-test, 143
 sensing a channel probe, 715
 sensing a external trigger probe, 719
 sensitivity of oscilloscope input, 226
 sequential commands, 802
 serial clock, 473, 501
 serial data, 474
 serial decode bus, 449
 serial decode bus display, 452
 serial decode mode, 453
 serial frame, 502
 serial number, 175
 service request, 141
 Service Request Enable Register (SRE), 139, 768
 set center frequency, 283
 set cursors, 731, 732
 set date, 567
 set time, 578
 set up oscilloscope, 43
 setting digital display, 245
 setting digital label, 202, 246
 setting digital position, 247
 setting digital threshold, 249
 setting display, 282
 setting external trigger level, 273
 setting impedance for channels, 222
 setting inversion for channels, 223
 setting pod display, 415
 setting pod size, 416
 setting pod threshold, 417
 settings, 133, 137
 settings, instrument, 298
 setup, 182, 196, 217, 244, 252, 298, 576
 setup configuration, 133, 137, 576
 setup defaults, 134, 572

setup memory, 133
 setup reported, 592
 setup, recall, 424
 setup, save, 440
 shape of modulation signal, waveform generator, 680
 short form, 5, 796
 show channel labels, 260
 show measurements, 342, 362
 SICL example in C, 884
 SICL example in Visual Basic, 893
 SICL examples, 884
 sigma, mask test run, 403
 signal type, 230
 signed data, 634
 simple command headers, 797
 sine waveform demo signal, 238
 sine waveform generator output, 669
 single acquisition, 176
 single-ended probe heads, 227
 single-ended signal type, 230
 single-shot demo signal, 238
 single-shot DUT, synchronizing with, 788
 size, 416
 size, digital channels, 248
 skew, 229, 713
 skew reference waveform, 701
 slope, 498, 607
 slope (direction) of waveform, 733
 slope not valid in TV trigger mode, 607
 slope parameter for delay measurement, 349
 slot, network printer, 309
 smoothing acquisition type, 633
 snapshot all measurement, 344
 software version, 128
 source, 363, 464, 486
 source for function, 295, 296, 720
 source for trigger, 608
 source for TV trigger, 627
 source, automask, 386
 source, mask test, 411
 source, save reference waveform, 446
 source, waveform, 651
 span, 280
 span of frequency on display, 284
 specify measurement, 363
 SPI, 498
 SPI clock timeout, 499
 SPI decode bit order, 497
 SPI decode word width, 510
 SPI MISO data, 655
 SPI SEARch commands, 557
 SPI serial search, data, 559
 SPI serial search, data width, 560
 SPI serial search, mode, 558
 SPI trigger, 500, 506, 508
 SPI trigger clock, 501
 SPI trigger commands, 495
 SPI trigger frame, 502
 SPI trigger MISO data pattern, 505
 SPI trigger MOSI data pattern, 507

- SPI trigger type, 509
 SPI trigger, MISO data source, 503
 SPI trigger, MOSI data source, 504
 SPI triggering, 450
 square wave duty cycle, waveform generator, 673
 square waveform generator output, 669
 SRE (Service Request Enable Register), 139, 768
 SRQ (Service Request interrupt), 159, 163
 Standard Event Status Enable Register (ESE), 124, 773
 Standard Event Status Register (ESR), 126, 772
 standard for video, 628
 standard, LIN, 487
 start acquisition, 142, 157, 174, 176
 start and stop edges, 347
 start cursor, 731
 start measurement, 342
 start print job, 313
 start time, 616, 731
 start time marker, 730
 state memory, 137
 state of instrument, 129, 576
 status, 140, 177, 179
 Status Byte Register (STB), 138, 140, 141, 766
 status data structure clear, 123
 status registers, 60
 status reporting, 761
 STB (Status Byte Register), 138, 140, 141, 766
 step size for frequency span, 284
 stop, 157, 178
 stop acquisition, 178
 stop cursor, 732
 stop displaying channel, 156
 stop displaying math function, 156
 stop displaying pod, 156
 stop on mask test failure, 402
 stop time, 616, 732
 storage, 137
 store instrument setup, 129, 137
 store setup, 137
 storing calibration information, 208
 string variables, 58
 string variables, reading multiple query results into, 59
 string variables, reading query results into multiple, 59
 string, quoted ASCII, 117
 subnet mask, 44
 subsource, waveform source, 655
 subsystem commands, 3, 799
 subtract math function, 280, 291, 651
 subtract math function as g(t) source, 287
 sweep mode, trigger, 591, 602
 sweep speed set to fast to measure fall time, 352
 sweep speed set to fast to measure rise time, 361
 switch disable, 570
 sync break, LIN, 488
 syntax elements, 116
 syntax rules, program data, 798
 syntax, optional terms, 116
 syntax, program message, 795
 SYStem commands, 565
 system commands, 567, 568, 569, 570, 576, 578
 system commands introduction, 566
- T**
- tdelta, 729
 tedge, 365
 telnet ports 5024 and 5025, 639
 Telnet sockets, 61
 temporary message, 568
 TER (Trigger Event Register), 179, 769
 termination conditions, mask test, 398
 test sigma, mask test run, 403
 test, self, 143
 text, writing to display, 568
 threshold, 249, 417, 712, 750
 threshold voltage (lower) for measurement, 727
 threshold voltage (upper) for measurement, 734
 thresholds, 347, 730
 thresholds used to measure period, 357
 thresholds, how autoscale affects, 151
 time base, 581, 582, 583, 584, 585, 749
 time base commands introduction, 580
 time base reset conditions, 135, 573
 time base window, 587, 588, 589
 time between points, 729
 time buckets, 183, 184
 time delay, 749
 time delta, 729
 time difference between data points, 659
 time duration, 616
 time holdoff for trigger, 596
 time interval, 365, 367, 729
 time interval between trigger and occurrence, 733
 time marker sets start time, 730
 time measurements with X cursors, 327
 time per division, 583
 time record, 286
 time reference, lister, 318
 time specified, 376
 time, calibration, 214
 time, mask test run, 404
 time, start marker, 731
 time, stop marker, 732
 time, system, 578
 time/div, how autoscale affects, 151
 TIMEbase commands, 579
 timebase vernier, 586
 TIMEbase:MODE, 56
 time-ordered label list, 261
- timeout, SPI clock, 499
 timing measurement, 342
 title channels, 224
 title, mask test, 412
 tolerance, automask, 388, 389
 top of waveform value measured, 377
 total frame count (CAN), 459
 total waveforms in mask test, 393
 trace memory, 177
 track measurements, 362
 transfer instrument state, 129, 576
 tree traversal rules, 799
 TRG (Trigger), 139, 141, 142
 TRIG OUT BNC, 209
 trigger armed event register, 165, 167
 trigger burst, UART, 526
 trigger channel source, 617, 627
 TRIGger commands, 591
 TRIGger commands, general, 593
 trigger data, UART, 527
 TRIGger EDGE commands, 603
 trigger edge coupling, 604
 trigger edge slope, 607
 trigger event bit, 179
 Trigger Event Register (TER), 769
 TRIGger GLITCH commands, 609
 trigger holdoff, 596
 trigger idle, UART, 528
 TRIGger IIC commands, 471
 trigger level auto set up, 597
 trigger level constants, 226
 trigger level voltage, 605
 trigger level, high, 598
 trigger level, low, 599
 TRIGger LIN commands, 481
 trigger occurred, 141
 TRIGger PATTern commands, 618
 trigger pattern qualifier, 622
 trigger qualifier, UART, 529
 trigger reset conditions, 135, 573
 trigger SPI clock slope, 498
 TRIGger SPI commands, 495
 trigger status bit, 179
 trigger sweep mode, 591
 TRIGger TV commands, 623
 trigger type, SPI, 509
 trigger type, UART, 530
 TRIGger UART commands, 511
 trigger, CAN, 465
 trigger, CAN pattern data length, 468
 trigger, CAN pattern ID mode, 470
 trigger, CAN sample point, 461
 trigger, CAN signal baudrate, 462
 trigger, CAN signal definition, 463
 trigger, CAN source, 464
 trigger, edge coupling, 604
 trigger, edge level, 605
 trigger, edge reject, 606
 trigger, edge slope, 607
 trigger, edge source, 608
 trigger, force a, 594
 trigger, glitch greater than, 611

trigger, glitch less than, 612
 trigger, glitch level, 613
 trigger, glitch polarity, 614
 trigger, glitch qualifier, 615
 trigger, glitch range, 616
 trigger, glitch source, 617
 trigger, high frequency reject filter, 595
 trigger, holdoff, 596
 trigger, IIC clock source, 473
 trigger, IIC data source, 474
 trigger, IIC pattern address, 475
 trigger, IIC pattern data, 476
 trigger, IIC pattern data 2, 477
 trigger, IIC qualifier, 478
 trigger, IIC signal baudrate, 485
 trigger, IIC type, 479
 trigger, LIN, 489
 trigger, LIN pattern data, 491
 trigger, LIN pattern data length, 493
 trigger, LIN pattern format, 494
 trigger, LIN sample point, 484
 trigger, LIN source, 486
 trigger, mode, 600
 trigger, noise reject filter, 601
 trigger, SPI clock slope, 498
 trigger, SPI clock source, 501
 trigger, SPI clock timeout, 499
 trigger, SPI frame source, 502
 trigger, SPI framing, 500
 trigger, SPI pattern MISO width, 506
 trigger, SPI pattern MOSI width, 508
 trigger, sweep, 602
 trigger, threshold, 750
 trigger, TV line, 624
 trigger, TV mode, 625, 751
 trigger, TV polarity, 626
 trigger, TV source, 627
 trigger, TV standard, 628
 trigger, UART base, 525
 trigger, UART baudrate, 514
 trigger, UART bit order, 515
 trigger, UART parity, 521
 trigger, UART polarity, 522
 trigger, UART Rx source, 523
 trigger, UART Tx source, 524
 trigger, UART width, 531
 truncation rules, 796
 TST (Self Test), 143
 tstart, 731
 tstop, 732
 TTL threshold voltage for digital channels, 249, 712
 TTL trigger threshold voltage, 750
 turn function on or off, 721
 turn off channel, 156
 turn off channel labels, 260
 turn off digital pod, 156
 turn off math function, 156
 turn on channel labels, 260
 turn on channel number display, 717
 turning channel display on and off, 221
 turning off/on function calculation, 282

turning vectors on or off, 716
 TV mode, 625, 751
 TV trigger commands, 623
 TV trigger line number setting, 624
 TV trigger mode, 627
 TV trigger polarity, 626
 TV trigger standard setting, 628
 TV triggering, 592
 tvmode, 751
 Tx data, UART, 655
 Tx frame count (UART), 519
 Tx source, 524
 type, 656

U

UART base, 525
 UART baud rate, 514
 UART bit order, 515
 UART frame counters, reset, 517
 UART parity, 521
 UART polarity, 522
 UART Rx source, 523
 UART SEARch commands, 561
 UART serial search, data, 562
 UART serial search, data qualifier, 564
 UART serial search, mode, 563
 UART trigger burst, 526
 UART trigger commands, 511
 UART trigger data, 527
 UART trigger idle, 528
 UART trigger qualifier, 529
 UART trigger type, 530
 UART Tx data, 655
 UART Tx source, 524
 UART width, 531
 UART/RS-232 triggering, 450
 units (vertical) for FFT, 285
 units per division, 233, 234, 277, 585
 units per division (vertical) for function, 233, 294
 units, automask, 387
 units, X cursor, 327, 328
 units, Y cursor, 332, 333
 unsigned data, 634
 unsigned mode, 657
 upper threshold, 357
 upper threshold voltage for measurement, 734
 uppercase characters in commands, 795
 URQ (User Request) status bit, 125, 127
 USB (Device) interface, 43
 user defined channel labels, 224
 user defined threshold, 712
 user event conditions occurred, 141
 user name, network domain, 310
 User's Guide, 6
 user-defined threshold voltage for digital channels, 249
 user-defined trigger threshold, 750
 USR (User Event bit), 139, 141

utilization, CAN bus, 460

V

valid command strings, 795
 value, 367
 value measured at base of waveform, 371
 value measured at specified time, 376
 value measured at top of waveform, 377
 value ranges, 117
 values required to fill time buckets, 184
 VBA, 52, 804
 vectors turned on or off, 716
 vectors, display, 263
 vectors, turning on or off, 252
 vernier, channel, 235
 vernier, horizontal, 586
 vertical adjustment, fine (vernier), 235
 vertical amplitude measurement, 369
 vertical axis defined by RANGE, 292
 vertical axis range for channels, 232
 vertical offset for channels, 225
 vertical peak-to-peak measured on waveform, 374
 vertical scale, 233, 294
 vertical scaling, 646
 vertical threshold, 712
 vertical units for FFT, 285
 vertical value at center screen, 290, 293
 vertical value maximum measured on waveform, 372
 vertical value measurements to calculate overshoot, 355
 vertical value minimum measured on waveform, 373
 video line to trigger on, 624
 video standard selection, 628
 view, 180, 280, 658, 717
 view turns function on or off, 721
 VISA COM example in C#, 813
 VISA COM example in Python, 830
 VISA COM example in Visual Basic, 804
 VISA COM example in Visual Basic .NET, 822
 VISA example in C, 837
 VISA example in C#, 856
 VISA example in Python, 877
 VISA example in Visual Basic, 846
 VISA example in Visual Basic .NET, 867
 VISA examples, 804, 837
 Visual Basic .NET, SCPI.NET example, 910
 Visual Basic .NET, VISA COM example, 822
 Visual Basic .NET, VISA example, 867
 Visual Basic 6.0, 53
 Visual Basic for Applications, 52, 804
 Visual Basic, SCL library example, 893
 Visual Basic, VISA COM example, 804
 Visual Basic, VISA example, 846
 voltage crossing reported or not found, 733
 voltage difference between data points, 662

voltage difference measured, 735
 voltage level for active trigger, 605
 voltage marker used to measure waveform, 736, 737
 voltage offset value for channels, 225
 voltage probe, 234, 277
 voltage ranges for channels, 232
 voltage ranges for external trigger, 276
 voltage threshold, 347

W

WAI (Wait To Continue), 144
 wait, 144
 wait for operation complete, 130
 Wait Trig bit, 165, 167
 warranty, 2
 waveform base value measured, 371
 WAVerform command, 51
 WAVerform commands, 629
 waveform data, 631
 waveform data format, 442
 waveform data length, 443
 waveform data length, maximum, 444
 waveform data, save, 441
 waveform generator, 667
 waveform generator amplitude, 690
 waveform generator function, 669
 waveform generator high-level voltage, 691
 waveform generator low-level voltage, 692
 waveform generator offset, 693
 waveform generator output control, 686
 waveform generator output load impedance, 687
 waveform generator period, 688
 waveform generator pulse width, 671
 waveform generator ramp symmetry, 672
 waveform generator reset defaults, 689
 waveform generator square wave duty cycle, 673
 waveform introduction, 631
 waveform maximum vertical value measured, 372
 waveform minimum vertical value measured, 373
 waveform must cross voltage level to be an occurrence, 733
 WAVerform parameters, 56
 waveform peak-to-peak vertical value measured, 374
 waveform period, 357
 waveform persistence, 252
 waveform RMS value measured, 375
 waveform save option for segments, 445
 waveform source, 651
 waveform source subsource, 655
 waveform vertical amplitude, 369
 waveform voltage measured at marker, 736, 737
 waveform, byte order, 637

waveform, count, 638
 waveform, data, 639
 waveform, format, 641
 waveform, points, 642, 644
 waveform, preamble, 646
 waveform, type, 656
 waveform, unsigned, 657
 waveform, view, 658
 waveform, X increment, 659
 waveform, X origin, 660
 waveform, X reference, 661
 waveform, Y increment, 662
 waveform, Y origin, 663
 waveform, Y reference, 664
 WAVeform:FORMat, 56
 waveforms, mask test run, 405
 Web control, 61
 WGEN commands, 665
 WGEN trigger source, 608
 what's new, 25
 width, 531, 616
 window, 587, 588, 589
 window time, 583
 window time base mode, 581
 window, measurement, 378
 windows, 286
 windows as filters to Fast Fourier Transforms, 286
 windows for Fast Fourier Transform functions, 286
 WMEMory commands, 695
 word format, 641
 word format for data transfer, 634
 word width, SPI decode, 510
 write text to display, 568
 WriteIEEEBlock method, 53, 59
 WriteList method, 53
 WriteNumber method, 53
 WriteString method, 53

X

X axis markers, 320
 X cursor units, 327, 328
 X delta, 326
 X delta, mask scaling, 408
 X1 and X2 cursor value difference, 326
 X1 cursor, 320, 322, 323
 X1, mask scaling, 407
 X2 cursor, 320, 324, 325
 X-axis functions, 580
 X-increment, 659
 X-origin, 660
 X-reference, 661
 X-Y mode, 580, 581

Y

Y axis markers, 320
 Y cursor units, 332, 333

Y offset, reference waveform, 702
 Y range, reference waveform, 703
 Y scale, reference waveform, 704
 Y1 and Y2 cursor value difference, 331
 Y1 cursor, 320, 323, 329, 331
 Y1, mask scaling, 409
 Y2 cursor, 320, 325, 330, 331
 Y2, mask scaling, 410
 Y-axis value, 663
 Y-increment, 662
 Y-origin, 663, 664
 Y-reference, 664

Z

zero values in waveform data, 639
 zoomed time base, 581
 zoomed time base measurement window, 378
 zoomed time base mode, how autoscale affects, 151
 zoomed window horizontal scale, 589