Agilent InfiniiVision 7000A Series Oscilloscopes

Programmer's Guide
 Notices

© Agilent Technologies, Inc. 2005-2010

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

 Trademarks

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

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

 Manual Part Number

Version 06.10.0001

 Edition

June 30, 2010

Available in electronic format only
Agilent Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

 Warranty

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

 Technology Licenses

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

 Restricted Rights Legend

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

 Safety Notices

 CAUTION

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

 WARNING

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

This book is your guide to programming the 7000A Series oscilloscopes:

Table 1  InfiniiVision 7000A Series Oscilloscope Models

<table>
<thead>
<tr>
<th>Channels</th>
<th>Input Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 GHz</td>
</tr>
<tr>
<td>4 analog + 16 digital (mixed-signal)</td>
<td>MSO7104A</td>
</tr>
<tr>
<td>2 analog + 16 digital (mixed-signal)</td>
<td></td>
</tr>
<tr>
<td>4 analog</td>
<td>DSO7104A</td>
</tr>
<tr>
<td>2 analog</td>
<td></td>
</tr>
</tbody>
</table>

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

- Chapter 1, Chapter 1, “What's New,” starting on page 21, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, Chapter 2, “Setting Up,” starting on page 35, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, Chapter 3, “Getting Started,” starting on page 45, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- Chapter 4, Chapter 4, “Commands Quick Reference,” starting on page 59, is a brief listing of the 7000 Series oscilloscope commands and syntax.

The next chapters provide reference information:

- Chapter 5, Chapter 5, “Commands by Subsystem,” starting on page 111, describes the set of commands that belong to an individual subsystem and explains the function of each command. Command arguments and syntax are described. Some command descriptions have example code.
- Chapter 6, Chapter 6, “Commands A-Z,” starting on page 655, contains an alphabetical listing of all command elements.
- Chapter 7, Chapter 7, “Obsolete and Discontinued Commands,” starting on page 689, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
• Chapter 8, **Chapter 8**, “Error Messages,” starting on page 745, lists the instrument error messages that can occur while programming the oscilloscope.

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

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

• Chapter 9, **Chapter 9**, “Status Reporting,” starting on page 753, describes the oscilloscope's status registers and how to check the status of the instrument.

• Chapter 10, **Chapter 10**, “Synchronizing Acquisitions,” starting on page 777, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.

• Chapter 11, **Chapter 11**, “More About Oscilloscope Commands,” starting on page 787, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

• Chapter 12, **Chapter 12**, “Programming Examples,” starting on page 813.

**Mixed-Signal Oscilloscope Channel Differences**

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

**See Also**

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

• For information on controller PC interface configuration, see the documentation for the interface card used (for example, the Agilent 82350A GPIB interface).

• For information on oscilloscope front-panel operation, see the *User's Guide*.

• For detailed connectivity information, refer to the *Agilent Technologies USB/LAN/GPIB Connectivity Guide*. For a printable electronic copy of the *Connectivity Guide*, direct your Web browser to "www.agilent.com" and search for "Connectivity Guide".

• For the latest versions of this and other manuals, see: "http://www.agilent.com/find/7000manual"
Contents

In This Book 3

1 What's New
   What's New in Version 6.10 22
   What's New in Version 6.00 23
   What's New in Version 5.25 26
   What's New in Version 5.20 28
   What's New in Version 5.15 31
   What's New in Version 5.10 33
   Version 5.00 at Introduction 34

2 Setting Up
   Step 1. Install Agilent IO Libraries Suite software 36
   Step 2. Connect and set up the oscilloscope 37
       Using the USB (Device) Interface 37
       Using the LAN Interface 37
   Step 3. Verify the oscilloscope connection 39

3 Getting Started
   Basic Oscilloscope Program Structure 46
       Initializing 46
       Capturing Data 46
       Analyzing Captured Data 47
Programming the Oscilloscope 48
Referencing the IO Library 48
Opening the Oscilloscope Connection via the IO Library 49
Initializing the Interface and the Oscilloscope 49
Using :AUToscale to Automate Oscilloscope Setup 50
Using Other Oscilloscope Setup Commands 50
Capturing Data with the :DIGitize Command 51
Reading Query Responses from the Oscilloscope 53
Reading Query Results into String Variables 54
Reading Query Results into Numeric Variables 54
Reading Definite-Length Block Query Response Data 54
Sending Multiple Queries and Reading Results 55
Checking Instrument Status 56

Other Ways of Sending Commands 57
Telnet Sockets 57
Sending SCPI Commands Using Browser Web Control 57

4 Commands Quick Reference

Command Summary 60
Syntax Elements 108
Number Format 108
<NL> (Line Terminator) 108
[ ] (Optional Syntax Terms) 108
{} (Braces) 108
::= (Defined As) 108
<> (Angle Brackets) 109
... (Ellipsis) 109
n,...,p (Value Ranges) 109
d (Digits) 109
Quoted ASCII String 109
Definite-Length Block Response Data 109

5 Commands by Subsystem

Common (*) Commands 113
*CLS (Clear Status) 117
*ESE (Standard Event Status Enable) 118
*ESR (Standard Event Status Register) 120
*IDN (Identification Number) 122
*LRN (Learn Device Setup) 123
*OPC (Operation Complete) 124
*OPT (Option Identification) 125
*RCL (Recall) 127
*RST (Reset) 128
*SAV (Save) 131
*SRE (Service Request Enable) 132
*STB (Read Status Byte) 134
*TRG (Trigger) 136
*TST (Self Test) 137
*WAI (Wait To Continue) 138

Root (;) Commands 139
:ACTivity 142
:AER (Arm Event Register) 143
:AUToscale 144
:AUToscale:AMODE 146
:AUToscale:CHANnels 147
:BLANk 148
:CDISplay 149
:DIGitize 150
:HWEnable (Hardware Event Enable Register) 152
:HWERegister:CONDition (Hardware Event Condition Register) 154
:HWERegister[:EVENT] (Hardware Event Event Register) 156
:MERGe 158
:MTEenable (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

:ACQuire Commands 181
:ACQuire:AALias 183
:ACQuire:COMPLETE 184
:ACQuire:COUNT 185
<table>
<thead>
<tr>
<th>Command Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ACQuire:DAALias</td>
<td>186</td>
</tr>
<tr>
<td>:ACQuire:MODE</td>
<td>187</td>
</tr>
<tr>
<td>:ACQuire:POINts</td>
<td>188</td>
</tr>
<tr>
<td>:ACQuire:RSGignal</td>
<td>189</td>
</tr>
<tr>
<td>:ACQuire:SEGmented:ANALyze</td>
<td>190</td>
</tr>
<tr>
<td>:ACQuire:SEGmented:COUNt</td>
<td>191</td>
</tr>
<tr>
<td>:ACQuire:SEGmented:INDex</td>
<td>192</td>
</tr>
<tr>
<td>:ACQuire:SRATe</td>
<td>195</td>
</tr>
<tr>
<td>:ACQuire:TYPE</td>
<td>196</td>
</tr>
<tr>
<td>:BUS&lt;n&gt; Commands</td>
<td>198</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:BIT&lt;m&gt;</td>
<td>200</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:BITs</td>
<td>201</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:CLEar</td>
<td>203</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:DISPlay</td>
<td>204</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:LABel</td>
<td>205</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:MASK</td>
<td>206</td>
</tr>
<tr>
<td>:CALibrate Commands</td>
<td>207</td>
</tr>
<tr>
<td>:CALibrate:DATE</td>
<td>209</td>
</tr>
<tr>
<td>:CALibrate:LABel</td>
<td>210</td>
</tr>
<tr>
<td>:CALibrate:OUTPut</td>
<td>211</td>
</tr>
<tr>
<td>:CALibrate:STARt</td>
<td>212</td>
</tr>
<tr>
<td>:CALibrate:STATus</td>
<td>213</td>
</tr>
<tr>
<td>:CALibrate:SWITch</td>
<td>214</td>
</tr>
<tr>
<td>:CALibrate:TEMPerature</td>
<td>215</td>
</tr>
<tr>
<td>:CALibrate:TIME</td>
<td>216</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt; Commands</td>
<td>217</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:BWLimit</td>
<td>220</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:COUPling</td>
<td>221</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:DISPlay</td>
<td>222</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:IMPedance</td>
<td>223</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:INVert</td>
<td>224</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:LABel</td>
<td>225</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:OFFSet</td>
<td>226</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe</td>
<td>227</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:HEAD[:TYPE]</td>
<td>228</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:ID</td>
<td>229</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:SKEW</td>
<td>230</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:STYPe</td>
<td>231</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROTection</td>
<td>232</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:RANGE</td>
<td>233</td>
</tr>
</tbody>
</table>
:FUNCTION:WINDow 280

:HARDcopy Commands 281
  :HARDcopy:AREA 283
  :HARDcopy:APRinter 284
  :HARDcopy:FACTors 285
  :HARDcopy:FFEed 286
  :HARDcopy:INKSaver 287
  :HARDcopy:LAYout 288
  :HARDcopy:PALETTE 289
  :HARDcopy:PRINTER:LIST 290
  :HARDcopy:START 291

:LISTer Commands 292
  :LISTer:DATA 293
  :LISTer:DISPlay 294

:MARKer Commands 295
  :MARKer:MODE 297
  :MARKer:X1Position 298
  :MARKer:X1Y1source 299
  :MARKer:X2Position 300
  :MARKer:X2Y2source 301
  :MARKer:XDELta 302
  :MARKer:Y1Position 303
  :MARKer:Y2Position 304
  :MARKer:YDELta 305

:MEASure Commands 306
  :MEASure:CLEar 314
  :MEASure:COUNter 315
  :MEASure:DEFine 316
  :MEASure:DELay 319
  :MEASure:DUTYcycle 321
  :MEASure:FALLtime 322
  :MEASure:FREQuency 323
  :MEASure:NWIDth 324
  :MEASure:OVERshoot 325
  :MEASure:PERiod 327
  :MEASure:PHASE 328
  :MEASure:PRESHoot 329
  :MEASure:PWIDth 330
  :MEASure:RESults 331
  :MEASure:RISetime 334
:MEASure:SDEViation 335
:MEASure:SHOW 336
:MEASure:SOURce 337
:MEASure:STATistics 339
:MEASure:STATistics:INCRement 340
:MEASure:STATistics:RESet 341
:MEASure:TEDGE 342
:MEASure:TVALue 344
:MEASure:VAMPplitude 346
:MEASure:VAverage 347
:MEASure:VBASE 348
:MEASure:VMAX 349
:MEASure:VMIN 350
:MEASure:VPP 351
:MEASure:VRATIO 352
:MEASure:VRMS 353
:MEASure:VTIME 354
:MEASure:VTOP 355
:MEASure:WINDow 356
:MEASure:XMAX 357
:MEASure:XMIN 358

:MTES:t:AMAS:k:CREate 364
:MTES:t:AMAS:k:SOURce 365
:MTES:t:AMAS:k:UNIt 366
:MTES:t:AMAS:k:XDELta 367
:MTES:t:AMAS:k:YDELta 368
:MTES:t:COUNT:FWAVEforms 369
:MTES:t:COUNT:RESet 370
:MTES:t:COUNT:TIME 371
:MTES:t:COUNT:WAVEforms 372
:MTES:t:DATA 373
:MTES:t:DELETE 374
:MTES:t:ENABLe 375
:MTES:t:LOCK 376
:MTES:t:OUTPut 377
:MTES:t:RMODe 378
:MTES:t:RMODe:FACTion:MEASure 379
:MTES:t:RMODe:FACTion:PRInt 380
:MTES:t:RMODe:FACTion:SAVE 381
:MTES:t:RMODe:FACTion:STOP 382
:MTES:rMoDe:SIGMa 383
:MTES:rMoDe:TIME 384
:MTES:rMoDe:WAVEforms 385
:MTES:SCALe:BIND 386
:MTES:SCALe:X1 387
:MTES:SCALe:XDELta 388
:MTES:SCALe:Y1 389
:MTES:SCALe:Y2 390
:MTES:SOURce 391
:MTES:TITLe 392

:POD Commands 393
:POD<n>:DISPlay 394
:POD<n>:SIZE 395
:POD<n>:THReshold 396

:RECall Commands 398
:RECall:FILename 399
:RECall:IMAGe[:STARt] 400
:RECall:MASK[:STARt] 401
:RECall:PWD 402
:RECall:SETup[:STARt] 403

:SAVE Commands 404
:SAVE:FILename 406
:SAVE:IMAGe[:STARt] 407
:SAVE:IMAGe:AREA 408
:SAVE:IMAGe:FACTors 409
:SAVE:IMAGe:FORMat 410
:SAVE:IMAGe:INKSaver 411
:SAVE:IMAGe:PALette 412
:SAVE:LISTer[:STARt] 413
:SAVE:MASK[:STARt] 414
:SAVE:PWD 415
:SAVE:SETup[:STARt] 416
:SAVE:WAVEform[:STARt] 417
:SAVE:WAVEform:FORMat 418
:SAVE:WAVEform:LENGTH 419
:SAVE:WAVEform:SEGMenTed 420

:SBUS Commands 421
:SBUS:CAN:COUNt:ERRor 423
:SBUS:CAN:COUNt:OVERload 424
:SBUS:CAN:COUNt:RESet 425
<table>
<thead>
<tr>
<th><strong>:TRIGger Commands</strong></th>
<th>468</th>
</tr>
</thead>
<tbody>
<tr>
<td>General :TRIGger Commands</td>
<td>471</td>
</tr>
<tr>
<td>:TRIGger:HFReject</td>
<td>472</td>
</tr>
<tr>
<td>:TRIGger:HOLDoff</td>
<td>473</td>
</tr>
<tr>
<td>:TRIGger:LFIfy</td>
<td>474</td>
</tr>
<tr>
<td>:TRIGger:MODE</td>
<td>475</td>
</tr>
<tr>
<td>:TRIGger:NREJect</td>
<td>476</td>
</tr>
<tr>
<td>:TRIGger:PATTer</td>
<td>477</td>
</tr>
<tr>
<td>:TRIGger:SWEep</td>
<td>479</td>
</tr>
<tr>
<td>:TRIGger:CAN Commands</td>
<td>480</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTer:DATA</td>
<td>482</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTer:DATA:LENGth</td>
<td>483</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTer:ID</td>
<td>484</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTer:ID:MODE</td>
<td>485</td>
</tr>
<tr>
<td>:TRIGger:CAN:SAmpLe</td>
<td>486</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:BAUDrate</td>
<td>487</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:DEFinition</td>
<td>488</td>
</tr>
<tr>
<td>:TRIGger:CAN:SOUrce</td>
<td>489</td>
</tr>
<tr>
<td>:TRIGger:CAN:TRIGger</td>
<td>490</td>
</tr>
<tr>
<td>:TRIGger:DURation Commands</td>
<td>492</td>
</tr>
<tr>
<td>:TRIGger:DURation:GREaterthan</td>
<td>493</td>
</tr>
<tr>
<td>:TRIGger:DURation:LESSthan</td>
<td>494</td>
</tr>
<tr>
<td>:TRIGger:DURation:PATTer</td>
<td>495</td>
</tr>
<tr>
<td>:TRIGger:DURation:QUALifier</td>
<td>496</td>
</tr>
<tr>
<td>:TRIGger:DURation:RANGe</td>
<td>497</td>
</tr>
<tr>
<td>:TRIGger:EBURst Commands</td>
<td>498</td>
</tr>
<tr>
<td>:TRIGger:EBURst:COUNT</td>
<td>499</td>
</tr>
<tr>
<td>:TRIGger:EBURst:IDLE</td>
<td>500</td>
</tr>
<tr>
<td>:TRIGger:EBURst:SLOPe</td>
<td>501</td>
</tr>
<tr>
<td>:TRIGger:[EDGE] Commands</td>
<td>502</td>
</tr>
<tr>
<td>:TRIGger:[EDGE]:COUPling</td>
<td>503</td>
</tr>
<tr>
<td>:TRIGger:[EDGE]:LEVel</td>
<td>504</td>
</tr>
<tr>
<td>:TRIGger:[EDGE]:REJect</td>
<td>505</td>
</tr>
<tr>
<td>:TRIGger:[EDGE]:SLOPe</td>
<td>506</td>
</tr>
<tr>
<td>:TRIGger:[EDGE]:SOUrce</td>
<td>507</td>
</tr>
<tr>
<td>:TRIGger:FLEXray Commands</td>
<td>508</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:AUTOsetup</td>
<td>509</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:BAUDrate</td>
<td>510</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:CHANnel</td>
<td>511</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:ERRor:TYPE</td>
<td>512</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:TYPE</td>
<td>513</td>
</tr>
</tbody>
</table>
:TRIGger:LIN:SIGNal:BAUDrate  564
:TRIGger:LIN:SOURce  565
:TRIGger:LIN:STANdard  566
:TRIGger:LIN:SYNCbreak  567
:TRIGger:LIN:TRIGger  568
:TRIGger:M1553 Commands  569
:TRIGger:M1553:AUTosetup  570
:TRIGger:M1553:PATTern:DATA  571
:TRIGger:M1553:RTA  572
:TRIGger:M1553:SOURce:LOWer  573
:TRIGger:M1553:SOURce:UPPer  574
:TRIGger:M1553:TYPE  575
:TRIGger:SEQuence Commands  576
:TRIGger:SEQuence:COUNt  577
:TRIGger:SEQuence:EDGE  578
:TRIGger:SEQuence:FIND  579
:TRIGger:SEQuence:PATTern  580
:TRIGger:SEQuence:RESet  581
:TRIGger:SEQuence:TIMer  582
:TRIGger:SEQuence:TRIGger  583
:TRIGger:SPI Commands  584
:TRIGger:SPI:CLOCk:SLOPe  585
:TRIGger:SPI:CLOCk:TIMeout  586
:TRIGger:SPI:FRAMing  587
:TRIGger:SPI:PATTern:DATA  588
:TRIGger:SPI:PATTern:WIDTH  589
:TRIGger:SPI:SOURce:CLOCk  590
:TRIGger:SPI:SOURce:DATA  591
:TRIGger:SPI:SOURce:FRAMe  592
:TRIGger:TV Commands  593
:TRIGger:TV:LINE  594
:TRIGger:TV:MODE  595
:TRIGger:TV:POLarity  596
:TRIGger:TV:SOURce  597
:TRIGger:TV:STANdard  598
:TRIGger:UART Commands  599
:TRIGger:UART:BASE  601
:TRIGger:UART:BAUDrate  602
:TRIGger:UART:BITorder  603
:TRIGger:UART:BURSt  604
:TRIGger:UART:DATA  605
:TRIGger:UART:IDLE  606
6 Commands A-Z

7 Obsolete and Discontinued Commands
8 Error Messages

9 Status Reporting

Status Reporting Data Structures 756
Status Byte Register (STB) 759
Service Request Enable Register (SRE) 761
Trigger Event Register (TER) 762
Output Queue 763
Message Queue 764
(Standard) Event Status Register (ESR) 765
(Standard) Event Status Enable Register (ESE) 766
Error Queue 767
Operation Status Event Register (:OPERegister[:EVENt]) 768
Operation Status Condition Register (:OPERegister:CONDition) 769
Arm Event Register (AER) 770
Overload Event Register (:OVLRegister) 771
Hardware Event Event Register (:HWERegister[:EVENt]) 772
Hardware Event Condition Register (:HWERegister:CONDition) 773
Mask Test Event Event Register (:MTERegister[:EVENt]) 774
Clearing Registers and Queues 775
Status Reporting Decision Chart 776

10 Synchronizing Acquisitions

Synchronization in the Programming Flow 778
   Set Up the Oscilloscope 778
   Acquire a Waveform 778
   Retrieve Results 778
Blocking Synchronization 779
Polling Synchronization With Timeout 780
Synchronizing with a Single-Shot Device Under Test (DUT) 782
Synchronization with an Averaging Acquisition 784
11 More About Oscilloscope Commands

Command Classifications 788
  Core Commands 788
  Non-Core Commands 788
  Obsolete Commands 788

Valid Command/Query Strings 789
  Program Message Syntax 789
  Command Tree 793
  Duplicate Mnemonics 807
  Tree Traversal Rules and Multiple Commands 808

Query Return Values 810

All Oscilloscope Commands Are Sequential 811

12 Programming Examples

VISA COM Examples 814
  VISA COM Example in Visual Basic 814
  VISA COM Example in C# 824
  VISA COM Example in Visual Basic .NET 836

VISA Examples 847
  VISA Example in C 847
  VISA Example in Visual Basic 856
  VISA Example in C# 866
  VISA Example in Visual Basic .NET 879

SICL Examples 893
  SICL Example in C 893
  SICL Example in Visual Basic 902

Index
1

What's New

What’s New in Version 6.10  22
What’s New in Version 6.00  23
What’s New in Version 5.25  26
What's New in Version 5.20  28
What’s New in Version 5.15  31
What's New in Version 5.10  33
Version 5.00 at Introduction  34
## What's New in Version 6.10

New features in version 6.10 of the InfiniiVision 7000 Series oscilloscope software are:

- When the zoomed time base mode is on, you can select whether the Main window or the Zoom window is used as the measurement window.
- An interval specification for the V average and dc RMS measurements has been added.
- A 50% trigger level command.

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

### New Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:WINDow (see page 356)</td>
<td>When the zoomed time base mode is on, specifies whether the Main window or the Zoom window is used as the measurement window.</td>
</tr>
<tr>
<td>:TRIGger:LFIFty (see page 474)</td>
<td>Sets the trigger level of a displayed analog channel trigger source to the waveform’s 50% value.</td>
</tr>
</tbody>
</table>

### Changed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:VAVerage (see page 347)</td>
<td>There is now an option for specifying the interval.</td>
</tr>
<tr>
<td>:MEASure:VRMS (see page 353)</td>
<td>There is now an option for specifying the interval.</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:DEFinition (see page 488)</td>
<td>There are now DIFH (differential H-L) and DIFL (differential L-H) options. The DIFL option is the same as the existing DIFFerential option. Also, this command is no longer classified as obsolete.</td>
</tr>
</tbody>
</table>
What's New in Version 6.00

New features in version 6.00 of the InfiniiVision 7000 Series oscilloscope software are:

- The ability to perform measurements and math functions on a 128K-point (maximum) precision analysis data record.
- Support for the new N5469A MIL-STD 1553 triggering and decode option (Option 553).
- Support for the new N5432C FlexRay triggering and decode option (Option FLX) which replaces previous FlexRay triggering and serial decode options.

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

### New Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SBUS:MODE (see page 437)</td>
<td>Determines the base to use for the MIL-STD 1553 decode display.</td>
</tr>
<tr>
<td>:SBUS:M1553:BASE (see page 436)</td>
<td>Selects the bit order used when displaying data in the SPI serial decode waveform and in the Lister.</td>
</tr>
<tr>
<td>:SYSTem:PRECision (see page 451)</td>
<td>Allows measurements and math functions to be performed on a precision analysis record (at the expense of waveform update rate).</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:BAUDrate (see page 510)</td>
<td>Specifies the baud rate of the FlexRay signal.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:CHANnel (see page 511)</td>
<td>Specifies whether the FlexRay input signal is for bus type A or B.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:SOURce (see page 518)</td>
<td>Specifies the input source channel probing the FlexRay signal.</td>
</tr>
<tr>
<td>:TRIGger:M1553 Commands (see page 569)</td>
<td>Commands for triggering on MIL-STD 1553 signals.</td>
</tr>
</tbody>
</table>

### Changed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SBUS:MODE (see page 437)</td>
<td>You can now select the M1553 serial bus decode mode.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:ERRor:TYPE (see page 512)</td>
<td>Now, only the FCRC, HCRC, or ALL error types can be selected.</td>
</tr>
</tbody>
</table>
Discontinued Commands

Most of the following commands have been discontinued because they are not supported by the new N5432C FlexRay triggering and decode option (Option FLX) which replaces previous FlexRay triggering and serial decode options.

<table>
<thead>
<tr>
<th>Discontinued Command</th>
<th>Current Command Equivalent</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DISPLAY:FREEze</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>:SBUS:BUSDoctor:ADDRESS</td>
<td>none</td>
<td>The VPT1000 (BusDoctor) vehicle protocol tester module is not used with the new FLX option.</td>
</tr>
<tr>
<td>:SBUS:BUSDoctor:BAUDrate</td>
<td>none</td>
<td>The VPT1000 (BusDoctor) vehicle protocol tester module is not used with the new FLX option. You now specify the baud rate using the :TRIGGER:FLEXRay:BAUDrate (see page 510) command.</td>
</tr>
<tr>
<td>:SBUS:BUSDoctor:CHANnel</td>
<td>none</td>
<td>The VPT1000 (BusDoctor) vehicle protocol tester module is not used with the new FLX option. You now specify bus A or B using the :TRIGGER:FLEXRay:CHANnel (see page 511) command.</td>
</tr>
<tr>
<td>Discontinued Command</td>
<td>Current Command Equivalent</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>:SBUS:BUSDoctor:MODE</td>
<td>none</td>
<td>The VPT1000 (BusDoctor) vehicle protocol tester module is not used with the new FLX option.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TIME:CBASE</td>
<td>none</td>
<td>Time triggering not supported by new FLX option.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TIME:CREPe</td>
<td>none</td>
<td>Time triggering not supported by new FLX option.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TIME:SEGMe</td>
<td>none</td>
<td>Time triggering not supported by new FLX option.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TIME:SLOT</td>
<td>none</td>
<td>Time triggering not supported by new FLX option.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:LEVel</td>
<td>none</td>
<td>The :TRIGger[:EDGE]:LEVel (see page 504) command is used instead, as with other trigger modes.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:SOURce</td>
<td>none</td>
<td>The input source channels are now specified using the :TRIGger:FLEXray:SOURce (see page 518) command.</td>
</tr>
</tbody>
</table>
What's New in Version 5.25

New features in version 5.25 of the InfiniiVision 7000 Series oscilloscope software are:

- The Lister display for showing decoded serial data in tabular format.
- The ability to trigger on and decode I2S serial bus data with a four-channel oscilloscope that includes the Option SND license.
- FlexRay event triggering.

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

### New Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:HEAD[:TYPE] (see page 228)</td>
<td>Sets an analog channel probe head type and dB value.</td>
</tr>
<tr>
<td>:DISPLAY:FREeze</td>
<td>Freezes the display without stopping currently running acquisitions.</td>
</tr>
<tr>
<td>:LISTer Commands (see page 292)</td>
<td>Commands for turning the Lister display on/off and for returning the Lister data.</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACtion:MEASure (see page 379)</td>
<td>Lets you enable or disable measurements on mask test failures.</td>
</tr>
<tr>
<td>:SAVE:LISTer[:STARt] (see page 413)</td>
<td>Saves the Lister display data to a file.</td>
</tr>
<tr>
<td>:SBUS:I2S:BASE (see page 433)</td>
<td>Determines the base to use for the I2S decode display.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:AUToSet</td>
<td>Automatically changes oscilloscope settings for the selected FlexRay event trigger type.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:LEVel</td>
<td>Lets you fine-tune the voltage level for the FlexRay event trigger.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:SOURce</td>
<td>Specifies the FlexRay event trigger source.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:TYPE (see page 513)</td>
<td>Specifies the FlexRay event type to trigger on.</td>
</tr>
<tr>
<td>:TRIGger:I2S Commands (see page 529)</td>
<td>Commands for triggering on I2S signals.</td>
</tr>
<tr>
<td>:TRIGger:LIN:PATTern:DATA (see page 559)</td>
<td>Sets the data value when triggering on a LIN frame ID and data.</td>
</tr>
<tr>
<td>:TRIGger:LIN:PATTern:DATA:LENGth (see page 561)</td>
<td>Sets the byte length of the LIN data string.</td>
</tr>
<tr>
<td>:TRIGger:LIN:PATTern:FORMat (see page 562)</td>
<td>Sets the entry (and query) number base used by the :TRIGger:LIN:PATTern:DATA command.</td>
</tr>
</tbody>
</table>
### Changed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SBUS:MODE (see page 437)</td>
<td>You can now select the I2S serial bus decode mode.</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TRIGger (see page 519)</td>
<td>You can now select FlexRay EVENt triggers.</td>
</tr>
<tr>
<td>:TRIGger:LIN:TRIGger (see page 568)</td>
<td>You can now select the DATA option for triggering on a LIN frame ID and data.</td>
</tr>
<tr>
<td>:TRIGger:MODE (see page 475)</td>
<td>You can now select the I2S trigger mode.</td>
</tr>
<tr>
<td>:TRIGger:TV:STANdard (see page 598)</td>
<td>The P1080L50HZ and P1080L60HZ standards have been added.</td>
</tr>
</tbody>
</table>
What’s New in Version 5.20

New features in version 5.20 of the InfiniiVision 7000 Series oscilloscope software are:

- Mask testing, enabled with Option LMT.
- Tracking cursors (markers) have been added.
- Measurement statistics have been added.
- Labels can now be up to 10 characters.

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

New Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ACQuire:SEGMented:ANALyze (see page 190)</td>
<td>Calculates measurement statistics and/or infinite persistence over all segments that have been acquired.</td>
</tr>
<tr>
<td>:CALibrate:OUTPut (see page 211)</td>
<td>Selects the signal output on the rear panel TRIG OUT BNC.</td>
</tr>
<tr>
<td>:HARDcopy:LAYout (see page 288)</td>
<td>Sets the hardcopy layout mode.</td>
</tr>
<tr>
<td>:MEASure:RESults (see page 331)</td>
<td>Returns measurement statistics values.</td>
</tr>
<tr>
<td>:MEASure:STATistics (see page 339)</td>
<td>Sets the type of measurement statistics to return.</td>
</tr>
<tr>
<td>:MEASure:STATistics:INCRement (see page 340)</td>
<td>Updates the statistics once (incrementing the count by one) using the current measurement values.</td>
</tr>
<tr>
<td>:MEASure:STATistics:RESet (see page 341)</td>
<td>Resets the measurement statistics values.</td>
</tr>
<tr>
<td>:MTEnable (Mask Test Event Enable Register) (see page 159)</td>
<td>Sets a mask in the Mask Test Event Enable register.</td>
</tr>
<tr>
<td>:MTERegister[:EVENt] (Mask Test Event Event Register) (see page 161)</td>
<td>Returns the integer value contained in the Mask Test Event Event Register and clears the register.</td>
</tr>
<tr>
<td>:MTESt Commands (see page 359)</td>
<td>Commands and queries to control the mask test (Option LMT) features.</td>
</tr>
<tr>
<td>:RECall:MASK[:STARt] (see page 414)</td>
<td>Recalls a mask.</td>
</tr>
<tr>
<td>:SAVE:MASK[:STARt] (see page 414)</td>
<td>Saves the current mask.</td>
</tr>
<tr>
<td>:SAVE:WAWeform:SEGMented (see page 420)</td>
<td>Specifies which segments are included when the waveform is saved.</td>
</tr>
<tr>
<td>:TRIGger:UART:BASE (see page 601)</td>
<td>Selects the front panel UART/RS232 trigger setup data selection option from HEX or BINary.</td>
</tr>
</tbody>
</table>
### Changed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:BUS&lt;n&gt;:LABel (see page 205)</td>
<td>Labels can now be up to 10 characters.</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:LABel (see page 225)</td>
<td>Labels can now be up to 10 characters.</td>
</tr>
<tr>
<td>:DIGital&lt;n&gt;:LABel (see page 240)</td>
<td>Labels can now be up to 10 characters.</td>
</tr>
<tr>
<td>:DISPlay:LABList (see page 250)</td>
<td>Labels can now be up to 10 characters.</td>
</tr>
<tr>
<td>:MARKer:MODE (see page 297)</td>
<td>You can now select the WAVEform tracking cursors mode.</td>
</tr>
<tr>
<td>:RECall:PWD (see page 402)</td>
<td>You can set the present working directory in addition to querying for this information.</td>
</tr>
<tr>
<td>:SAVE:IMAGe[:START] (see page 407)</td>
<td>The file extension specified will change the :SAVE:IMAGe:FORMat setting if it is a valid image file extension.</td>
</tr>
<tr>
<td>:SAVE:PWD (see page 415)</td>
<td>You can set the present working directory in addition to querying for this information.</td>
</tr>
<tr>
<td>:SAVE:WAVeform[:START] (see page 407)</td>
<td>The file extension specified will change the :SAVE:WAVeform:FORMat setting if it is a valid waveform file extension.</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:BAUDrate (see page 487)</td>
<td>The baud rate value can now be set in 100 b/s increments.</td>
</tr>
<tr>
<td>:TRIGger:LIN:SIGNal:BAUDrate (see page 564)</td>
<td>The baud rate value can now be set in 100 b/s increments.</td>
</tr>
<tr>
<td>:TRIGger:UART:BAUDrate (see page 602)</td>
<td>The baud rate value can now be set in 100 b/s increments and the maximum baud rate is now 3 M b/s.</td>
</tr>
<tr>
<td>:TRIGger:UART:DATA (see page 605)</td>
<td>You can now specify the data value using a quoted ASCII character.</td>
</tr>
</tbody>
</table>

### Obsolete Commands

<table>
<thead>
<tr>
<th>Obsolete Command</th>
<th>Current Command Equivalent</th>
<th>Behavior Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MTESt:AMASK:{SAVE</td>
<td>STORE} (see page 729)</td>
<td>:SAVE:MASK[:START] (see page 414)</td>
</tr>
<tr>
<td>:MTESt:AVERage (see page 730)</td>
<td>:ACQuire:TYPE AVERage (see page 196)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:AVERage:COUNt (see page 731)</td>
<td>:ACQuire:COUNt (see page 185)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:LOAD (see page 732)</td>
<td>:RECall:MASK[:START] (see page 401)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:RUMode (see page 733)</td>
<td>:MTESt:RMODE (see page 378)</td>
<td></td>
</tr>
</tbody>
</table>
### What's New

<table>
<thead>
<tr>
<th>Obsolete Command</th>
<th>Current Command Equivalent</th>
<th>Behavior Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>.MTEST:RUMode:SOFailure (see page 734)</td>
<td>.MTEST:RMODE:FACTion:STOP (see page 382)</td>
<td></td>
</tr>
<tr>
<td>.MTEST:(START</td>
<td>STOP) (see page 735)</td>
<td>.RUN (see page 174) or .STOP (see page 178)</td>
</tr>
<tr>
<td>.MTEST:TRIGger:SOURce (see page 736)</td>
<td>.TRIGger Commands (see page 468)</td>
<td></td>
</tr>
</tbody>
</table>
What's New in Version 5.15

New features in version 5.15 of the InfiniiVision 7000 Series oscilloscope software are:

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

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:FUNCtion:GOFT:OPERation (see page 269)</td>
<td>Selects the math operation for the internal g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, and SQRT functions.</td>
</tr>
<tr>
<td>:FUNCtion:GOFT:SOURce1 (see page 270)</td>
<td>Selects the first input channel for the g(t) source.</td>
</tr>
<tr>
<td>:FUNCtion:GOFT:SOURce2 (see page 271)</td>
<td>Selects the second input channel for the g(t) source.</td>
</tr>
<tr>
<td>:FUNCtion:SOURce1 (see page 277)</td>
<td>Selects the first source for the ADD, SUBTract, and MULTiply arithmetic operations or the single source for the FFT, INTegrate, DIFFerentiate, and SQRT functions.</td>
</tr>
<tr>
<td>:FUNCtion:SOURce2 (see page 278)</td>
<td>Selects the second input channel for the ADD, SUBTract, and MULTiply arithmetic operations.</td>
</tr>
<tr>
<td>:MEASure:VRATio (see page 352)</td>
<td>Measures and returns the ratio of AC RMS values of the specified sources expressed in dB.</td>
</tr>
<tr>
<td>:SYSTem:PROTection:LOCK (see page 452)</td>
<td>Disables/enables the fifty ohm input impedance setting.</td>
</tr>
</tbody>
</table>
## What's New

### Changed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ACQuire:COUNt (see page 185)</td>
<td>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.</td>
</tr>
<tr>
<td>:FUNCTION:OPERation (see page 273)</td>
<td>The ADD parameter is new, and now that waveform math can be performed using channels 3 and 4, this command selects the operation only.</td>
</tr>
<tr>
<td>:FUNCTION:WINDow (see page 280)</td>
<td>You can now select the Blackman-Harris FFT window.</td>
</tr>
</tbody>
</table>

### Obsolete Commands

<table>
<thead>
<tr>
<th>Obsolete Command</th>
<th>Current Command Equivalent</th>
<th>Behavior Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:FUNCTION:SOURce (see page 706)</td>
<td>:FUNCTION:SOURce1 (see page 277)</td>
<td>Obsolete command has ADD, SUBTract, and MULTIply parameters; current command has GOFT parameter.</td>
</tr>
</tbody>
</table>
What's New in Version 5.10

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

- Segmented memory acquisition mode, enabled with Option SGM.

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

### New Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>::ACQuire::SEGMenTed::COUNt (see page 191)</td>
<td>Sets the number of memory segments.</td>
</tr>
<tr>
<td>::ACQuire::SEGMenTed::INDex (see page 192)</td>
<td>Selects the segmented memory index.</td>
</tr>
<tr>
<td>::WAVeform::SEGMenTed::COUNt (see page 639)</td>
<td>Returns the number of segments in the currently acquired waveform data.</td>
</tr>
<tr>
<td>::WAVeform::SEGMenTed::TTAG (see page 640)</td>
<td>Returns the time tag for the selected segmented memory index.</td>
</tr>
</tbody>
</table>

### Changed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>::ACQuire::MODE (see page 187)</td>
<td>You can now select the SEGMenTed memory mode.</td>
</tr>
</tbody>
</table>
1 What's New

Version 5.00 at Introduction

The Agilent InfiniiVision 7000 Series oscilloscopes were introduced with version 5.00 of oscilloscope operating software. The command set is based on the 6000 Series oscilloscopes (and the 54620/54640 Series oscilloscopes before them).
2

Setting Up

Step 1. Install Agilent IO Libraries Suite software  36
Step 2. Connect and set up the oscilloscope  37
Step 3. Verify the oscilloscope connection  39

This chapter explains how to install the Agilent IO Libraries Suite software, connect the oscilloscope to the controller PC, set up the oscilloscope, and verify the oscilloscope connection.
Step 1. Install Agilent IO Libraries Suite software

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

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

- "http://www.agilent.com/find/iolib"
Step 2. Connect and set up the oscilloscope

The 7000 Series oscilloscope has two different interfaces you can use for programming: USB (device) or LAN.

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

![Control Connectors on Rear Panel](image)

Using the USB (Device) Interface

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

   This is a USB 2.0 high-speed port.

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

Using the LAN Interface

1. If the controller PC isn't already connected to the local area network (LAN), do that first.

2. Get the oscilloscope's network parameters (hostname, domain, IP address, subnet mask, gateway IP, DNS IP, etc.) from your network administrator.

3. Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port on the back of the oscilloscope.
4 On the oscilloscope, verify that the controller interface is enabled:
   a Press the **Utility** button.
   b Using the softkeys, press **I/O** and **Control**.
   c Ensure the box next to **LAN** is selected ( ). If not ( ), use the Entry knob to select **LAN**; then, press the **Control** softkey again.

5 Configure the oscilloscope's LAN interface:
   a Press the **Configure** softkey until "LAN" is selected.
   b Press the **LAN Settings** softkey.
   c Press the **Addresses** softkey. Use the **IP Options** softkey and the Entry knob to select DHCP, AutoIP, or netBIOS. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the IP Address, Subnet Mask, Gateway IP, and DNS IP values. When you are done, press the return (up arrow) softkey.
   d Press the **Domain** softkey. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the Host name and the Domain name. When you are done, press the return (up arrow) softkey.
Step 3. Verify the oscilloscope connection

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

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

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

b If the oscilloscope is on the same subnet, select it, and click **OK**.
Otherwise, if the instrument is not on the same subnet, click **Add Address**.

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

ii Click **Test Connection**.
iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.
3 Test some commands on the instrument:
   a Right-click on the instrument and choose **Send Commands To This Instrument** from the popup menu.

   ![Agilent InfiniiVision 7000A Series Oscilloscopes Programmer's Guide](image1.png)

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

   ![Agilent Connection Expert](image2.png)

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

4 In the Agilent Connection Expert application, choose **File>Exit** from the menu to exit the application.
Setting Up
3 Getting Started

Basic Oscilloscope Program Structure 46
Programming the Oscilloscope 48
Other Ways of Sending Commands 57

This chapter gives you an overview of programming the 7000 Series oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

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

NOTE Language for Program Examples

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

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

```
  Initialize
    ↓
  Capture
    ↓
  Analyze
```

**Initializing**

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

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

**Capturing Data**

Once you initialize the oscilloscope, you can begin capturing data for analysis. Remember that while the oscilloscope is responding to commands from the controller, it is not performing acquisitions. Also, when you change the oscilloscope configuration, any data already captured will most likely be rendered.
To collect data, you use the :DIGitize command. This command clears the waveform buffers and starts the acquisition process. Acquisition continues until acquisition memory is full, then stops. The acquired data is displayed by the oscilloscope, and the captured data can be measured, stored in trace memory in the oscilloscope, or transferred to the controller for further analysis. Any additional commands sent while :DIGitize is working are buffered until :DIGitize is complete.

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

**Analyzing Captured Data**

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

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

- "Referencing the IO Library" on page 48
- "Opening the Oscilloscope Connection via the IO Library" on page 49
- "Using :AUToscale to Automate Oscilloscope Setup" on page 50
- "Using Other Oscilloscope Setup Commands" on page 50
- "Capturing Data with the :DIGitize Command" on page 51
- "Reading Query Responses from the Oscilloscope" on page 53
- "Reading Query Results into String Variables" on page 54
- "Reading Query Results into Numeric Variables" on page 54
- "Reading Definite-Length Block Query Response Data" on page 54
- "Sending Multiple Queries and Reading Results" on page 55
- "Checking Instrument Status" on page 56

Referencing the IO Library

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

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

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

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

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

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

Opening the Oscilloscope Connection via the IO Library

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

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

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

```vb
Dim myMgr As VisaComLib.ResourceManager
Dim myScope As VisaComLib.FormattedIO488
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

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

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

The ":DISPLAY:LABEL ON" in the above example is called a program message. Program messages are explained in more detail in "Program Message Syntax" on page 789.

Initializing the Interface and the Oscilloscope

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. When using the Agilent VISA COM library, you can use the resource session object's Clear method to clear the interface buffer:
Dim myMgr As VisaComLib.ResourceManager
Dim myScope As VisaComLib.FormattedIO488

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

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

' Clear the interface buffer.
myScope.IO.Clear

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

After clearing the interface, initialize the instrument to a preset state:
myScope.WriteString "*RST"

NOTE

Information for Initializing the Instrument
The actual commands and syntax for initializing the instrument are discussed in "Common (*) Commands" on page 113.
Refer to the Agilent IO Libraries Suite documentation for information on initializing the interface.

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

The syntax for the autoscale command is:
myScope.WriteString ":AUToscale"

Using Other Oscilloscope Setup Commands
A typical oscilloscope setup would set the vertical range and offset voltage, 
the horizontal range, delay time, delay reference, trigger mode, trigger 
level, and slope. An example of the commands that might be sent to the 
oscilloscope are:
myScope.WriteString ":CHANnel1:PROBe 10*
myScope.WriteString ":CHANnel1:RANGe 16*
myScope.WriteString ":CHANnel1:OFFSet 1.00*
myScope.WriteString ":TIMebase:MODE MAIN*
myScope.WriteString ":TIMebase:RANGe 1E-3*
myScope.WriteString ":TIMebase:DELay 100E-6*
Vertical is set to 16 V full-scale (2 V/div) with center of screen at 1 V and probe attenuation set to 10. This example sets the time base at 1 ms full-scale (100 ms/div) with a delay of 100 µs.

**Example Oscilloscope Setup Code**

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

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

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

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

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

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

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

**Capturing Data with the :DIGitize Command**

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

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:

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

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

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

Digitized waveforms are passed from the instrument to the controller by sending a numerical representation of each digitized point. The format of the numerical representation is controlled with the :WAVeform:FORMat command and may be selected as BYTE, WORD, or ASCii.
The easiest method of transferring a digitized waveform depends on data structures, formatting available and I/O capabilities. You must scale the integers to determine the voltage value of each point. These integers are passed starting with the left most point on the instrument's display.

For more information, see the waveform subsystem commands and corresponding program code examples in ":WAVEform Commands" on page 619.

**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:

```vbnet
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.
Getting Started

Reading Query Results into String Variables

The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific command descriptions in Chapter 5, “Commands by Subsystem,” starting on page 111 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:

```vbnet
myScope.WriteString "::CHAnell: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:

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

After running this program, the controller displays:

Range (variant): 40

Reading Definite-Length Block Query Response Data

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

For example, for transmitting 1000 bytes of data, the syntax would be:
The "8" states the number of digits that follow, and "00001000" states the number of bytes to be transmitted.

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

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

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

**Sending Multiple Queries and Reading Results**

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

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

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

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

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

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

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

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

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

**Checking Instrument Status**

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

For more information, see Chapter 9, “Status Reporting,” starting on page 753 which explains how to check the status of the instrument.
Other Ways of Sending Commands

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

Telnet Sockets

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

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

telnet <hostname> 5024

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

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

telnet <hostname> 5025

Sending SCPI Commands Using Browser Web Control

To send SCPI commands using the Browser Web Control feature, establish a connection to the oscilloscope via LAN as described in the 7000 Series Oscilloscopes User's Guide. When you make the connection to the oscilloscope via LAN and the instrument's welcome page is displayed, select the Browser Web Control tab, then select the Remote Programming link.
3  Getting Started
4

Commands Quick Reference

Command Summary  60
Syntax Elements  108
# Command Summary

## Table 2  Common (*) Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CLS (see page 117)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
| *ESE <mask> (see page 118) | *ESE? (see page 119) | <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 120) | <status> ::= 0 to 255; an integer in NR1 format  
| n/a             | *IDN? (see page 120) | 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 123) | <learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format  
| *OPC (see page 124) | *OPC? (see page 124) | ASCII "1" is placed in the output queue when all pending device operations have completed. |
## Table 2  Common (*) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>*OPT? (see page 125)</td>
<td>&lt;return_value&gt; ::= 0,0,&lt;license info&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;license info&gt; ::= &lt;All field&gt;, &lt;reserved&gt;, &lt;Factory MSO&gt;, &lt;Upgraded MSO&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Xilinx FPGA Probe&gt;, &lt;Memory&gt;, &lt;Low Speed Serial&gt;, &lt;Automotive Serial&gt;, &lt;reserved&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Secure&gt;, &lt;Battery&gt;, &lt;Altera FPGA Probe&gt;, &lt;FlexRay Serial&gt;, &lt;Power Measurements&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;RS-232/UART Serial&gt;, &lt;reserved&gt;, &lt;Segmented Memory&gt;, &lt;Mask Test&gt;, &lt;reserved&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;reserved&gt;, &lt;FlexRay Conformance&gt;, &lt;reserved&gt;, &lt;I2S Serial&gt;, &lt;FlexRay Trigger/Decode&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;reserved&gt;, &lt;reserved&gt;, &lt;MIL-STD 1553 Trigger/Decode&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;All field&gt; ::= (0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;reserved&gt; ::= 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Factory MSO&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Upgraded MSO&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Xilinx FPGA Probe&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Memory&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Low Speed Serial&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Automotive Serial&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Secure&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Battery&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Altera FPGA Probe&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;FlexRay Serial&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Power Measurements&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;RS-232/UART Serial&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Segmented Memory&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Mask Test&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;FlexRay Conformance&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;I2S Serial&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;FlexRay Trigger/Decode&gt; ::= {0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;MIL-STD 1553 Trigger/Decode&gt; ::= {0</td>
</tr>
</tbody>
</table>
### Table 2  Common (*) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>*RCL &lt;value&gt;</td>
<td>n/a</td>
<td>&lt;value&gt; ::= {0</td>
</tr>
<tr>
<td>*RST</td>
<td>n/a</td>
<td>See *RST (Reset) (see page 128)</td>
</tr>
<tr>
<td>*SAV &lt;value&gt;</td>
<td>n/a</td>
<td>&lt;value&gt; ::= {0</td>
</tr>
<tr>
<td>*SRE &lt;mask&gt;</td>
<td>*SRE?</td>
<td>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. &lt;mask&gt; ::= following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Weight Name Enables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--- ------ ---- ---------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 128 OPER Operation Status Reg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 64 ---- (Not used.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 32 ESB Event Status Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 16 MAV Message Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 8 ---- (Not used.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 4 MSG Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 2 USR User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 TRG Trigger</td>
</tr>
<tr>
<td>*TRG</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>*TST?</td>
<td>n/a</td>
<td>&lt;result&gt; ::= 0 or non-zero value; an integer in NR1 format</td>
</tr>
<tr>
<td>*WAI</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Command</td>
<td>Query</td>
<td>Options and Query Returns</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>:ACTivity (see page 142)</td>
<td>:ACTivity? (see page 142)</td>
<td>&lt;return value&gt; ::= &lt;edges&gt;,&lt;levels&gt; &lt;edges&gt; ::= presence of edges (32-bit integer in NR1 format) &lt;levels&gt; ::= logical highs or lows (32-bit integer in NR1 format)</td>
</tr>
<tr>
<td>n/a</td>
<td>:AER? (see page 143)</td>
<td>(0</td>
</tr>
<tr>
<td>:AUToscale [&lt;source&gt;[,...,&lt;source&gt;]] (see page 144)</td>
<td>n/a</td>
<td>&lt;source&gt; ::= CHANnel&lt;n&gt; for DSO models &lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:AUToscale:AMODE &lt;value&gt; (see page 146)</td>
<td>:AUToscale:AMODE? (see page 146)</td>
<td>&lt;value&gt; ::= {NORMal</td>
</tr>
<tr>
<td>:AUToscale:CHANnels &lt;value&gt; (see page 147)</td>
<td>:AUToscale:CHANnels? (see page 147)</td>
<td>&lt;value&gt; ::= {ALL</td>
</tr>
<tr>
<td>:BLANK [&lt;source&gt;] (see page 148)</td>
<td>n/a</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;}</td>
</tr>
<tr>
<td>:CDISplay (see page 149)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:DIGitize [&lt;source&gt;[,...,&lt;source&gt;]] (see page 150)</td>
<td>n/a</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;}</td>
</tr>
</tbody>
</table>
### Table 3  Root (:) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:HWEenable &lt;n&gt; (see page 152)</td>
<td>:HWEenable? (see page 152)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:HWERegister:CONDition? (see page 154)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:HWERegister[:EVENT]? (see page 156)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>:MERGe &lt;pixel memory&gt; (see page 158)</td>
<td>n/a</td>
<td>&lt;pixel memory&gt; ::= {PMEMory{0</td>
</tr>
<tr>
<td>:MTEenable &lt;n&gt; (see page 159)</td>
<td>:MTEenable? (see page 159)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:MTERegister[:EVENT]? (see page 161)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>:OPEE &lt;n&gt; (see page 163)</td>
<td>:OPEE? (see page 164)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:OPERegister:CONDition? (see page 165)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:OPERegister[:EVENT]? (see page 167)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>:OVLenable &lt;mask&gt; (see page 169)</td>
<td>:OVLenable? (see page 170)</td>
<td>&lt;mask&gt; ::= 16-bit integer in NR1 format as shown: Bit Weight Input --- ------ ---------- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL</td>
</tr>
<tr>
<td>:PRINT [&lt;options&gt;] (see page 173)</td>
<td>n/a</td>
<td>&lt;options&gt; ::= [&lt;print option&gt;],...,[&lt;print option&gt;] &lt;print option&gt; ::= {COlor</td>
</tr>
</tbody>
</table>
### Table 3  Root (:) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:RUN (see page 174)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SERial (see page 175)</td>
<td>&lt;return value&gt; ::= unquoted string containing serial number</td>
</tr>
<tr>
<td>:SINGLE (see page 176)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:STATus? &lt;display&gt; (see page 177)</td>
<td>{0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;display&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:STOP (see page 178)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:TER? (see page 179)</td>
<td>{0</td>
</tr>
<tr>
<td>:VIEW &lt;source&gt; (see page 180)</td>
<td>n/a</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
</tbody>
</table>

### Table 4  :ACQuire Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:ACQuire:AALias? (see page 183)</td>
<td>{1</td>
</tr>
<tr>
<td>:ACQuire:COMPLETE &lt;complete&gt; (see page 184)</td>
<td>:ACQuire:COMPLETE? (see page 184)</td>
<td>&lt;complete&gt; ::= 100; an integer in NR1 format</td>
</tr>
<tr>
<td>:ACQuire:COUNT &lt;count&gt; (see page 185)</td>
<td>:ACQuire:COUNT? (see page 185)</td>
<td>&lt;count&gt; ::= an integer from 2 to 65536 in NR1 format</td>
</tr>
<tr>
<td>:ACQuire:DAALias &lt;mode&gt; (see page 186)</td>
<td>:ACQuire:DAALias? (see page 186)</td>
<td>&lt;mode&gt; ::= {DISable</td>
</tr>
<tr>
<td>:ACQuire:MODE &lt;mode&gt; (see page 187)</td>
<td>:ACQuire:MODE? (see page 187)</td>
<td>&lt;mode&gt; ::= {RTIME</td>
</tr>
</tbody>
</table>
### Table 4  :ACQuire Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:ACQuire:POINts? (see page 188)</td>
<td>&lt;# points&gt; ::= an integer in NR1 format</td>
</tr>
<tr>
<td>:ACQuire:RSIGnal &lt;ref_signal_mode&gt; (see page 189)</td>
<td>:ACQuire:RSIGnal? (see page 189)</td>
<td>&lt;ref_signal_mode&gt; ::= {OFF</td>
</tr>
<tr>
<td>:ACQuire:SEGMented:ANALyze (see page 190)</td>
<td>n/a</td>
<td>n/a (with Option SGM)</td>
</tr>
<tr>
<td>:ACQuire:SEGMented:COUNT&lt;br&gt; &lt;count&gt; (see page 191)</td>
<td>:ACQuire:SEGMented:COUNT? (see page 191)</td>
<td>&lt;count&gt; ::= an integer from 2 to 2000 (w/8M memory) in NR1 format (with Option SGM)</td>
</tr>
<tr>
<td>:ACQuire:SEGMented:INDEX &lt;index&gt; (see page 192)</td>
<td>:ACQuire:SEGMented:INDEX? (see page 192)</td>
<td>&lt;index&gt; ::= an integer from 2 to 2000 (w/8M memory) in NR1 format (with Option SGM)</td>
</tr>
<tr>
<td>n/a</td>
<td>:ACQuire:SRATE? (see page 195)</td>
<td>&lt;sample_rate&gt; ::= sample rate (samples/s) in NR3 format</td>
</tr>
<tr>
<td>:ACQuire:TYPE &lt;type&gt; (see page 196)</td>
<td>:ACQuire:TYPE? (see page 196)</td>
<td>&lt;type&gt; ::= {NORMal</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:BUS&lt;n&gt;:BIT&lt;m&gt; {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:BITS &lt;channel_list&gt;, {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:CLEar (see page 203)</td>
<td>n/a</td>
<td>&lt;n&gt; ::= 1 or 2; an integer in NR1 format</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
</tbody>
</table>
### Table 5  :BUS<n> Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:BUS&lt;n&gt;:LABel &lt;string&gt; (see page 205)</td>
<td>:BUS&lt;n&gt;:LABel? (see page 205)</td>
<td>&lt;string&gt; ::= quoted ASCII string up to 10 characters&lt;br&gt;&lt;n&gt; ::= 1 or 2; an integer in NR1 format</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:MASK &lt;mask&gt; (see page 206)</td>
<td>:BUS&lt;n&gt;:MASK? (see page 206)</td>
<td>&lt;mask&gt; ::= 32-bit integer in decimal, &lt;nondecimal&gt;, or&lt;br&gt;&lt;string&gt; &lt;nondecimal&gt; ::= #Hnn...n where n ::= (0,..9</td>
</tr>
</tbody>
</table>

### Table 6  :CALibrate Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:CALibrate:DATE? (see page 209)</td>
<td>&lt;return value&gt; ::= &lt;day&gt;,&lt;month&gt;,&lt;year&gt;; all in NR1 format</td>
</tr>
<tr>
<td>:CALibrate:LABel &lt;string&gt; (see page 210)</td>
<td>:CALibrate:LABel? (see page 210)</td>
<td>&lt;string&gt; ::= quoted ASCII string up to 32 characters</td>
</tr>
<tr>
<td>:CALibrate:OUTPut &lt;signal&gt; (see page 211)</td>
<td>:CALibrate:OUTPut? (see page 211)</td>
<td>&lt;signal&gt; ::= {TRIGgers</td>
</tr>
<tr>
<td>:CALibrate:STARt (see page 212)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:STATUS? (see page 213)</td>
<td>&lt;return value&gt; ::= ALL,&lt;status_code&gt;,&lt;status_string&gt;&lt;status_code&gt; ::= an integer status code&lt;br&gt;&lt;status_string&gt; ::= an ASCII status string</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:SWITCh? (see page 214)</td>
<td>(PROTected</td>
</tr>
</tbody>
</table>
### Table 6  :CALibrate Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:CALibrate:TEMPerature? (see page 215)</td>
<td>&lt;return value&gt; ::= degrees C delta since last cal in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:TIME? (see page 216)</td>
<td>&lt;return value&gt; ::= &lt;hours&gt;,&lt;minutes&gt;,&lt;seconds&gt;; all in NR1 format</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;n&gt;:BWLimit {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:COUPLing &lt;coupling&gt; (see page 221)</td>
<td>:CHANnel&lt;n&gt;:COUPLing? (see page 221)</td>
<td>&lt;coupling&gt; ::= {AC</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:IMPedance &lt;impedance&gt; (see page 223)</td>
<td>:CHANnel&lt;n&gt;:IMPedance? (see page 223)</td>
<td>&lt;impedance&gt; ::= {ONEMeg</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:INVert {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:LABel &lt;string&gt; (see page 225)</td>
<td>:CHANnel&lt;n&gt;:LABel? (see page 225)</td>
<td>&lt;string&gt; ::= any series of 10 or less ASCII characters enclosed in quotation marks &lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:OFFSet &lt;offset&gt;[suffix] (see page 226)</td>
<td>:CHANnel&lt;n&gt;:OFFSet? (see page 226)</td>
<td>&lt;offset&gt; ::= Vertical offset value in NR3 format [suffix] ::= {V</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe &lt;attenuation&gt; (see page 227)</td>
<td>:CHANnel&lt;n&gt;:PROBe? (see page 227)</td>
<td>&lt;attenuation&gt; ::= Probe attenuation ratio in NR3 format &lt;n&gt; ::= 1-2 or 1-4r in NR1 format</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:HEAD [{:TYPE} &lt;head_param&gt;] (see page 228)</td>
<td>:CHANnel&lt;n&gt;:PROBe:HEAD [{:TYPE}]? (see page 228)</td>
<td>&lt;head_param&gt; ::= {SEND0</td>
</tr>
</tbody>
</table>
### Table 7 :CHANnel<n> Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:CHANnel&lt;n&gt;:PROBe:ID? (see page 229)</td>
<td>&lt;probe id&gt; ::= unquoted ASCII string up to 11 characters &lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:SKEW &lt;skew_value&gt; (see page 230)</td>
<td>:CHANnel&lt;n&gt;:PROBe:SKEW? (see page 230)</td>
<td>&lt;skew_value&gt; ::= -100 ns to +100 ns in NR3 format &lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:STYPe &lt;signal type&gt; (see page 231)</td>
<td>:CHANnel&lt;n&gt;:PROBe:STYPe? (see page 231)</td>
<td>&lt;signal type&gt; ::= {DIFFerential</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROTectio n (see page 232)</td>
<td>:CHANnel&lt;n&gt;:PROTectio n? (see page 232)</td>
<td>{NORM</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:RANGE &lt;range&gt;[suffix] (see page 233)</td>
<td>:CHANnel&lt;n&gt;:RANGE? (see page 233)</td>
<td>&lt;range&gt; ::= Vertical full-scale range value in NR3 format [suffix] ::= {V</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:SCALE &lt;scale&gt;[suffix] (see page 234)</td>
<td>:CHANnel&lt;n&gt;:SCALE? (see page 234)</td>
<td>&lt;scale&gt; ::= Vertical units per division value in NR3 format [suffix] ::= {V</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:UNITS &lt;units&gt; (see page 235)</td>
<td>:CHANnel&lt;n&gt;:UNITs? (see page 235)</td>
<td>&lt;units&gt; ::= {VOLT</td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:VERNier {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DIGital&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:DIGital&lt;n&gt;:LABel &lt;string&gt; (see page 240)</td>
<td>:DIGital&lt;n&gt;:LABel? (see page 240)</td>
<td>&lt;string&gt; ::= any series of 10 or less ASCII characters enclosed in quotation marks &lt;n&gt; ::= 0-15; an integer in NR1 format</td>
</tr>
</tbody>
</table>
### Table 8: :DIGital<n> Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :DIGital<n>:POSITION <position> (see page 241) | :DIGital<n>:POSITION? (see page 241) | <n> ::= 0-15; an integer in NR1 format  
|                                 |                             | <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small |
| :DIGital<n>:SIZE <value> (see page 242) | :DIGital<n>:SIZE? (see page 242) | <value> ::= {SMALL | MEDIUM | LARGE} |
| :DIGital<n>:THReshold <value>[suffix] (see page 243) | :DIGital<n>:THReshold? (see page 243) | <n> ::= 0-15; an integer in NR1 format  
|                                 |                             | <value> ::= {CMOS | ECL | TTL | <user defined value>}  
|                                 |                             | <user defined value> ::= value in NR3 format from -8.00 to +8.00  
|                                 |                             | [suffix] ::= {V | mV | uV} |

### Table 9: :DISPlay Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DISPlay:CLEar (see page 246)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
| :DISPlay:DATA [format][,][area][,][palette]<display data> (see page 247) | :DISPlay:DATA? [format][,][area][,][palette] (see page 247) | <format> ::= {TIFF} (command)  
|                                 |                             | <area> ::= {GRAticule} (command)  
|                                 |                             | <palette> ::= {MONochrome} (command)  
|                                 |                             | <format> ::= {TIFF | BMP | BMP8bit | PNG} (query)  
|                                 |                             | <area> ::= {GRAticule | SCReen} (query)  
|                                 |                             | <palette> ::= {MONochrome | GRAYscale | COlor} (query)  
<p>|                                 |                             | &lt;display data&gt; ::= data in IEEE 488.2 # format |
| :DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 249) | :DISPlay:LABel? (see page 249) | (0 | 1) |
| :DISPlay:LABList &lt;binary block&gt; (see page 250) | :DISPlay:LABList? (see page 250) | &lt;binary block&gt; ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters |</p>
<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DISPlay:PERSistence</td>
<td>:DISPlay:PERSistence? (see page 251)</td>
<td>&lt;value&gt; ::= {MINimum</td>
</tr>
<tr>
<td>&lt;value&gt; (see page 251)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:DISPlay:SOURce</td>
<td>:DISPlay:SOURce? (see page 252)</td>
<td>&lt;value&gt; ::= {PMEMory{0</td>
</tr>
<tr>
<td>&lt;value&gt; (see page 252)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:DISPlay:VECTors</td>
<td>:DISPlay:VECTors? (see page 253)</td>
<td>(1</td>
</tr>
<tr>
<td>{(1</td>
<td>ON)</td>
<td>{0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:EXTernal:BWLimit</td>
<td>:EXTernal:BWLimit? (see page 256)</td>
<td>&lt;bwlimit&gt; ::= {0</td>
</tr>
<tr>
<td>&lt;bwlimit&gt; (see page 256)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:EXTernal:IMPedance</td>
<td>:EXTernal:IMPedance? (see page 257)</td>
<td>&lt;impedance&gt; ::= {ONEMeg</td>
</tr>
<tr>
<td>&lt;value&gt; (see page 257)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:EXTernal:PROBe</td>
<td>:EXTernal:PROBe? (see page 258)</td>
<td>&lt;attenuation&gt; ::= probe attenuation ratio in NR3 format</td>
</tr>
<tr>
<td>&lt;attenuation&gt; (see page 258)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>:EXTernal:PROBe:ID? (see page 259)</td>
<td>&lt;probe id&gt; ::= unquoted ASCII string up to 11 characters</td>
</tr>
<tr>
<td>:EXTernal:PROBe:STYPe</td>
<td>:EXTernal:PROBe:STYPe? (see page 260)</td>
<td>&lt;signal type&gt; ::= {DIFFerential</td>
</tr>
<tr>
<td>&lt;signal type&gt; (see page 260)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:EXTernal:PROTection[]</td>
<td>:EXTernal:PROTection? (see page 261)</td>
<td>{NORM</td>
</tr>
<tr>
<td>:CLEar (see page 261)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:EXTernal:RANGe</td>
<td>:EXTernal:RANGe? (see page 262)</td>
<td>&lt;range&gt; ::= vertical full-scale range value in NR3 format</td>
</tr>
<tr>
<td>&lt;range&gt; [&lt;suffix&gt;] (see page 262)</td>
<td></td>
<td>&lt;suffix&gt; ::= {V</td>
</tr>
<tr>
<td>:EXTernal:UNITs</td>
<td>:EXTernal:UNITs? (see page 263)</td>
<td>&lt;units&gt; ::= {VOLT</td>
</tr>
<tr>
<td>&lt;units&gt; (see page 263)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 11 :FUNCtion Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:FUNCtion:CENTer &lt;frequency&gt; (see page 267)</td>
<td>:FUNCtion:CENTer? (see page 267)</td>
<td>&lt;frequency&gt; ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.</td>
</tr>
<tr>
<td>:FUNCtion:DISPLAY {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:FUNCtion:GOFT:OPERation &lt;operation&gt; (see page 269)</td>
<td>:FUNCtion:GOFT:OPERation? (see page 269)</td>
<td>&lt;operation&gt; ::= {ADD</td>
</tr>
<tr>
<td>:FUNCtion:GOFT:SOURce 1 &lt;source&gt; (see page 270)</td>
<td>:FUNCtion:GOFT:SOURce 1? (see page 270)</td>
<td>&lt;source&gt; ::= CHANnel&lt;n&gt; &lt;n&gt; ::= {1</td>
</tr>
<tr>
<td>:FUNCtion:GOFT:SOURce 2 &lt;source&gt; (see page 271)</td>
<td>:FUNCtion:GOFT:SOURce 2? (see page 271)</td>
<td>&lt;source&gt; ::= CHANnel&lt;n&gt; &lt;n&gt; ::= {(1</td>
</tr>
<tr>
<td>:FUNCtion:OFFSET &lt;offset&gt; (see page 272)</td>
<td>:FUNCtion:OFFSET? (see page 272)</td>
<td>&lt;offset&gt; ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.</td>
</tr>
<tr>
<td>:FUNCtion:OPERation &lt;operation&gt; (see page 273)</td>
<td>:FUNCtion:OPERation? (see page 273)</td>
<td>&lt;operation&gt; ::= {ADD</td>
</tr>
<tr>
<td>:FUNCtion:RANGE &lt;range&gt; (see page 274)</td>
<td>:FUNCtion:RANGE? (see page 274)</td>
<td>&lt;range&gt; ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTEGRate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.</td>
</tr>
</tbody>
</table>
### Table 11: `:FUNCtion` Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:FUNCtion:REFerence</code> &lt;level&gt;</td>
<td><code>:FUNCtion:REFerence?</code> &lt;level&gt;</td>
<td><code>&lt;level&gt; ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.</code></td>
</tr>
<tr>
<td><code>:FUNCtion:SCALe &lt;scale value&gt;[&lt;suffix&gt;]</code></td>
<td><code>:FUNCtion:SCALe?</code> &lt;scale value&gt;[&lt;suffix&gt;]</td>
<td><code>&lt;scale value&gt; ::= integer in NR1 format </code>&lt;suffix&gt; ::= {V</td>
</tr>
<tr>
<td><code>:FUNCtion:SOURce1 &lt;source&gt;</code></td>
<td><code>:FUNCtion:SOURce1?</code> &lt;source&gt;</td>
<td>`&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td><code>:FUNCtion:SOURce2 &lt;source&gt;</code></td>
<td><code>:FUNCtion:SOURce2?</code> &lt;source&gt;</td>
<td>`&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td><code>:FUNCtion:SPAN &lt;span&gt;</code></td>
<td><code>:FUNCtion:SPAN?</code> &lt;span&gt;</td>
<td><code>&lt;span&gt; ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.</code></td>
</tr>
<tr>
<td><code>:FUNCtion:WINDow &lt;window&gt;</code></td>
<td><code>:FUNCtion:WINDow?</code> &lt;window&gt;</td>
<td>`&lt;window&gt; ::= {RECTangular</td>
</tr>
</tbody>
</table>

### Table 12: `:HARDcopy` Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:HARDcopy:AREA &lt;area&gt;</code> (see page 283)</td>
<td><code>:HARDcopy:AREA?</code> (see page 283)</td>
<td><code>&lt;area&gt; ::= SCReen</code></td>
</tr>
<tr>
<td><code>:HARDcopy:APRinter &lt;active_printer&gt;</code> (see page 284)</td>
<td><code>:HARDcopy:APRinter?</code> (see page 284)</td>
<td>`&lt;active_printer&gt; ::= {&lt;index&gt;</td>
</tr>
</tbody>
</table>
### Table 12: :HARDcopy Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:HARDcopy:FACTors</code></td>
<td><code>:HARDcopy:FACTors?</code></td>
<td>`{0</td>
</tr>
<tr>
<td>`{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td><code>:HARDcopy:FFEed</code></td>
<td><code>:HARDcopy:FFEed?</code></td>
<td>`{0</td>
</tr>
<tr>
<td>`{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td><code>:HARDcopy:INKSaver</code></td>
<td><code>:HARDcopy:INKSaver?</code></td>
<td>`{0</td>
</tr>
<tr>
<td>`{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td><code>:HARDcopy:LAYout</code></td>
<td><code>:HARDcopy:LAYout?</code></td>
<td>`&lt;layout&gt; ::= {LANDscape</td>
</tr>
<tr>
<td><code>&lt;layout&gt;</code> (see page 288)</td>
<td>[see page 288]</td>
<td></td>
</tr>
<tr>
<td><code>:HARDcopy:PAlette</code></td>
<td><code>:HARDcopy:PAlette?</code></td>
<td>`&lt;palette&gt; ::= {COLor</td>
</tr>
<tr>
<td><code>&lt;palette&gt;</code> (see page 289)</td>
<td>[see page 289]</td>
<td></td>
</tr>
</tbody>
</table>
| `n/a`                    | `:HARDcopy:PRINTER:LIST?`   | `<list> ::= [{printer_spec}] ... [printer_spec]`
|                          |                             | `<printer_spec> ::= 
|                          |                             | "<index>,<active>,<name>;"
|                          |                             | `<index> ::= integer index of printer`
|                          |                             | `<active> ::= {Y | N}`
|                          |                             | `<name> ::= name of printer` |
| `:HARDcopy:START`       | n/a                         | n/a                       |
| (see page 291)           |                             | [see page 291]             |

### Table 13: :LISTer Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| `n/a`                    | `:LISTer:DATA?`             | `<binary_block> ::= comma-separated data with
|                          | (see page 293)              | newlines at the end of each row |
| `:LISTer:DISPLAY`        | `:LISTer:DISPLAY?`          | `{0 | 1}`                   |
| `{0 | OFF} | {1 | ON}` (see page 294) | [see page 294]             |
## Table 14: :MARDer Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MARDer:MODE &lt;mode&gt; (see page 297)</td>
<td>:MARDer:MODE? (see page 297)</td>
<td>&lt;mode&gt; ::= {OFF</td>
</tr>
<tr>
<td>:MARDer:X1Position &lt;position&gt;[suffix] (see page 298)</td>
<td>:MARDer:X1Position? (see page 298)</td>
<td>&lt;position&gt; ::= X1 cursor position value in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:MARDer:X1Y1source &lt;source&gt; (see page 299)</td>
<td>:MARDer:X1Y1source? (see page 299)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MARDer:X2Position &lt;position&gt;[suffix] (see page 300)</td>
<td>:MARDer:X2Position? (see page 300)</td>
<td>&lt;position&gt; ::= X2 cursor position value in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:MARDer:X2Y2source &lt;source&gt; (see page 301)</td>
<td>:MARDer:X2Y2source? (see page 301)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td>:MARDer:XDELta? (see page 302)</td>
<td>&lt;return_value&gt; ::= X cursors delta value in NR3 format</td>
</tr>
<tr>
<td>:MARDer:Y1Position &lt;position&gt;[suffix] (see page 303)</td>
<td>:MARDer:Y1Position? (see page 303)</td>
<td>&lt;position&gt; ::= Y1 cursor position value in NR3 format [suffix] ::= {V</td>
</tr>
<tr>
<td>:MARDer:Y2Position &lt;position&gt;[suffix] (see page 304)</td>
<td>:MARDer:Y2Position? (see page 304)</td>
<td>&lt;position&gt; ::= Y2 cursor position value in NR3 format [suffix] ::= {V</td>
</tr>
<tr>
<td>n/a</td>
<td>:MARDer:YDELta? (see page 305)</td>
<td>&lt;return_value&gt; ::= Y cursors delta value in NR3 format</td>
</tr>
</tbody>
</table>
### Table 15: :MEASURE Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASURE:CLEar (see page 314)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:MEASURE:COUNter [&lt;source&gt;] (see page 315)</td>
<td>:MEASURE:COUNter? [&lt;source&gt;] (see page 315)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:DEFine DELay, &lt;delay spec&gt; (see page 316)</td>
<td>:MEASURE:DEFine? DELay (see page 317)</td>
<td>&lt;delay spec&gt; ::= &lt;edge_spec1&gt;,&lt;edge_spec2&gt;</td>
</tr>
<tr>
<td>:MEASURE:DEFine THresholds, &lt;threshold spec&gt; (see page 316)</td>
<td>:MEASURE:DEFine? THresholds (see page 317)</td>
<td>&lt;threshold spec&gt; ::= {STANdard}</td>
</tr>
<tr>
<td>:MEASURE:DELay [&lt;source1&gt;] [,&lt;source2&gt;] (see page 319)</td>
<td>:MEASURE:DELay? [&lt;source1&gt;] [,&lt;source2&gt;] (see page 319)</td>
<td>&lt;source1,2&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:DUTYcycle [&lt;source&gt;] (see page 321)</td>
<td>:MEASURE:DUTYcycle? [&lt;source&gt;] (see page 321)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
### Table 15: :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :MEASure:FALLtime [<source>] (see page 322) | :MEASure:FALLtime? [<source>] (see page 322) | <source> ::= {CHANnel<n> | FUNCTION | MATH} for DSO models  
              <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCTION | MATH} for MSO models  
              <n> ::= 1-2 or 1-4 in NR1 format  
              <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format |
| :MEASure:FREQuency [<source>] (see page 323) | :MEASure:FREQuency? [<source>] (see page 323) | <source> ::= {CHANnel<n> | FUNCTION | MATH} for DSO models  
              <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCTION | MATH} for MSO models  
              <n> ::= 1-2 or 1-4 in NR1 format  
              <return_value> ::= frequency in Hertz in NR3 format |
| :MEASure:NWIDth  [<source>] (see page 324)   | :MEASure:NWIDth?  [<source>] (see page 324)   | <source> ::= {CHANnel<n> | FUNCTION | MATH} for DSO models  
              <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCTION | MATH} for MSO models  
              <n> ::= 1-2 or 1-4 in NR1 format  
              <return_value> ::= negative pulse width in seconds-NR3 format |
| :MEASure:OVERshoot [<source>] (see page 325) | :MEASure:OVERshoot? [<source>] (see page 325) | <source> ::= {CHANnel<n> | FUNCTION | MATH}  
              <n> ::= 1-2 or 1-4 in NR1 format  
              <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format |
| :MEASure:PERiod  [<source>] (see page 327)    | :MEASure:PERiod?  [<source>] (see page 327)    | <source> ::= {CHANnel<n> | FUNCTION | MATH} for DSO models  
              <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCTION | MATH} for MSO models  
              <n> ::= 1-2 or 1-4 in NR1 format  
              <return_value> ::= waveform period in seconds in NR3 format |
## Table 15: MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:PHASE [&lt;source1&gt;] [,&lt;source2&gt;] (see page 328)</td>
<td>:MEASure:PHASE? [,&lt;source1&gt;] [,&lt;source2&gt;] (see page 328)</td>
<td>&lt;source1,2&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:PRESHoot [&lt;source&gt;] (see page 329)</td>
<td>:MEASure:PRESHoot? [&lt;source&gt;] (see page 329)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:PWIDth [&lt;source&gt;] (see page 330)</td>
<td>:MEASure:PWIDth? [&lt;source&gt;] (see page 330)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td>:MEASure:RES ults? &lt;result_list&gt; (see page 331)</td>
<td>&lt;result_list&gt; ::= comma-separated list of measurement results</td>
</tr>
<tr>
<td>:MEASure:RISetime [&lt;source&gt;] (see page 334)</td>
<td>:MEASure:RISetime? [&lt;source&gt;] (see page 334)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:SDEVIation [&lt;source&gt;] (see page 335)</td>
<td>:MEASure:SDEVIation? [&lt;source&gt;] (see page 335)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:SHOW {1</td>
<td>ON} (see page 336)</td>
<td>:MEASure:SHOW? (see page 336)</td>
</tr>
</tbody>
</table>
### Table 15 :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:SOURce &lt;source1&gt; [,&lt;source2&gt;]</td>
<td>:MEASure:SOURce? (see page 337)</td>
<td>&lt;source1,2&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:STATistics &lt;type&gt;</td>
<td>:MEASure:STATistics? (see page 339)</td>
<td>&lt;type&gt; ::= {{ON</td>
</tr>
<tr>
<td>:MEASure:STATistics:INCREMENT</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:MEASure:STATistics:RESET</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:MEASure:TEDGe? &lt;slope&gt;&lt;occurrence&gt;[, &lt;source&gt;]</td>
<td>&lt;slope&gt; ::= direction of the waveform &lt;occurrence&gt; ::= the transition to be reported &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
Table 15 :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:MEASure:TVALue?</td>
<td>&lt;value&gt; ::= voltage level that the waveform must cross.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;slope&gt; ::= direction of the waveform when &lt;value&gt; is crossed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;occurrence&gt; ::= transitions reported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;return_value&gt; ::= time in seconds of specified voltage crossing in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;return_value&gt; ::= the amplitude of the selected waveform in volts in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;return_value&gt; ::= calculated average voltage in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;base_voltage&gt; ::= voltage at the base of the selected waveform in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;return_value&gt; ::= maximum voltage of the selected waveform in NR3 format</td>
</tr>
</tbody>
</table>
### Table 15: MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :MEASure:VMIN    | :MEASure:VMIN?       | <source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1-2 or 1-4 in NR1 format  
$return_value$ ::= minimum voltage of the selected waveform in NR3 format |
| [<source>]       | [see page 350]       |                                                                                                                                                                                                                         |
|                  | :MEASure:VMIN?       |                                                                                                                                                                                                                         |
|                  | [see page 350]       |                                                                                                                                                                                                                         |
|                  | [source]             |                                                                                                                                                                                                                         |
| :MEASure:VPP     | :MEASure:VPP?        | <source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1-2 or 1-4 in NR1 format  
$return_value$ ::= voltage peak-to-peak of the selected waveform in NR3 format |
| [<source>]       | [see page 351]       |                                                                                                                                                                                                                         |
|                  | :MEASure:VPP?        |                                                                                                                                                                                                                         |
|                  | [see page 351]       |                                                                                                                                                                                                                         |
|                  | [source]             |                                                                                                                                                                                                                         |
| :MEASure:VRATio  | :MEASure:VRATio?     | <source1,2> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1-2 or 1-4 in NR1 format  
$return_value$ ::= the ratio value in dB in NR3 format |
| [<source1>]      | [see page 328]       |                                                                                                                                                                                                                         |
| [,<source2>]     | (see page 328)       |                                                                                                                                                                                                                         |
|                  | :MEASure:VRATio?     |                                                                                                                                                                                                                         |
|                  | [see page 352]       |                                                                                                                                                                                                                         |
|                  | [source]             |                                                                                                                                                                                                                         |
| :MEASure:VRMS    | :MEASure:VRMS?       | <interval> ::= {CYCLE | DISPLAY | AUTO}  
<source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1-2 or 1-4 in NR1 format  
$return_value$ ::= calculated dc RMS voltage in NR3 format |
| [<interval>]     | [see page 353]       |                                                                                                                                                                                                                         |
| [:][,<source>]   | (see page 353)       |                                                                                                                                                                                                                         |
|                  | :MEASure:VRMS?       |                                                                                                                                                                                                                         |
|                  | [see page 353]       |                                                                                                                                                                                                                         |
|                  | [source]             |                                                                                                                                                                                                                         |
| n/a              | :MEASure:VTIME?      | <vt ime> ::= displayed time from trigger in seconds in NR3 format  
$return_value$ ::= voltage at the specified time in NR3 format  
<source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1-2 or 1-4 in NR1 format  
$return_value$ ::= calculated dc RMS voltage in NR3 format |
| [source]         | [see page 354]       |                                                                                                                                                                                                                         |
|                  | :MEASure:VTIME?      |                                                                                                                                                                                                                         |
|                  | (see page 354)       |                                                                                                                                                                                                                         |
| :MEASure:VTOP    | :MEASure:VTOP?       | <source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1-2 or 1-4 in NR1 format  
$return_value$ ::= voltage at the top of the waveform in NR3 format  
<window> ::= (MAIN | ZOOM | AUTO) |
| [<source>]       | (see page 355)       |                                                                                                                                                                                                                         |
|                  | :MEASure:VTOP?       |                                                                                                                                                                                                                         |
|                  | (see page 355)       |                                                                                                                                                                                                                         |
|                  | [source]             |                                                                                                                                                                                                                         |
| :MEASure:WINDow  | :MEASure:WINDow?     | <window> ::= (MAIN | ZOOM | AUTO)                                                                                                                                                                                               |
| <window>         | (see page 356)       |                                                                                                                                                                                                                         |
|                  | :MEASure:WINDow?     |                                                                                                                                                                                                                         |
|                  | (see page 356)       |                                                                                                                                                                                                                         |
### Table 15 :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:XMAX [source] (see page 357)</td>
<td>:MEASure:XMAX? [source] (see page 357)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:XMIN [source] (see page 358)</td>
<td>:MEASure:XMIN? [source] (see page 358)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>

### Table 16 :MTESt Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MTEST:AMASK:CREate (see page 364)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:MTEST:AMASK:SOURce &lt;source&gt; (see page 365)</td>
<td>:MTEST:AMASK:SOURce? (see page 365)</td>
<td>&lt;source&gt; ::= CHANnel&lt;n&gt; &lt;n&gt; ::= (1</td>
</tr>
<tr>
<td>:MTEST:AMASK:UNITs &lt;units&gt; (see page 366)</td>
<td>:MTEST:AMASK:UNITs? (see page 366)</td>
<td>&lt;units&gt; ::= {CURRent</td>
</tr>
<tr>
<td>:MTEST:AMASK:XDELta &lt;value&gt; (see page 367)</td>
<td>:MTEST:AMASK:XDELta? (see page 367)</td>
<td>&lt;value&gt; ::= X delta value in NR3 format</td>
</tr>
<tr>
<td>:MTEST:AMASK:YDELta &lt;value&gt; (see page 368)</td>
<td>:MTEST:AMASK:YDELta? (see page 368)</td>
<td>&lt;value&gt; ::= Y delta value in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:MTEST:COUNT:FWAVeforms [CHANnel&lt;n&gt;] (see page 369)</td>
<td>&lt;failed&gt; ::= number of failed waveforms in NR1 format</td>
</tr>
<tr>
<td>:MTEST:COUNT:RESet (see page 370)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:MTEST:COUNT:TIME? (see page 371)</td>
<td>&lt;time&gt; ::= elapsed seconds in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:MTEST:COUNT:WAVEform s? (see page 372)</td>
<td>&lt;count&gt; ::= number of waveforms in NR1 format</td>
</tr>
<tr>
<td>Command</td>
<td>Query</td>
<td>Options and Query Returns</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>:MTESt:DATA &lt;mask&gt; (see page 373)</td>
<td>:MTESt:DATA? (see page 373)</td>
<td>&lt;mask&gt; ::= data in IEEE 488.2 # format.</td>
</tr>
<tr>
<td>:MTESt:DELete (see page 374)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:MTESt:ENABLE {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:LOCK {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:OUTPut &lt;signal&gt; (see page 377)</td>
<td>:MTESt:OUTPut? (see page 377)</td>
<td>&lt;signal&gt; ::= {FAIL</td>
</tr>
<tr>
<td>:MTESt:RMODe &lt;rmode&gt; (see page 378)</td>
<td>:MTESt:RMODe? (see page 378)</td>
<td>&lt;rmode&gt; ::= {FORever</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:MEASure {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:PRINT {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:SAVE {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:STOP {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:SIGMa &lt;level&gt; (see page 383)</td>
<td>:MTESt:RMODe:SIGMa? (see page 383)</td>
<td>&lt;level&gt; ::= from 0.1 to 9.3 in NR3 format</td>
</tr>
<tr>
<td>:MTESt:RMODe:TIME &lt;seconds&gt; (see page 384)</td>
<td>:MTESt:RMODe:TIME? (see page 384)</td>
<td>&lt;seconds&gt; ::= from 1 to 86400 in NR3 format</td>
</tr>
<tr>
<td>:MTESt:RMODe:WAVeform s &lt;count&gt; (see page 385)</td>
<td>:MTESt:RMODe:WAVeform s? (see page 385)</td>
<td>&lt;count&gt; ::= number of waveforms in NR1 format</td>
</tr>
<tr>
<td>:MTESt:SCALE:BIND {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
</tbody>
</table>
### Table 16  :MTESt Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MTESt:SCALE:X1 &lt;x1_value&gt;</td>
<td>:MTESt:SCALE:X1?</td>
<td>&lt;x1_value&gt; ::= X1 value in NR3 format</td>
</tr>
<tr>
<td></td>
<td>(see page 387)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:SCALE:XDELta &lt;xdelta_value&gt;</td>
<td>:MTESt:SCALE:XDELta? (see page 388)</td>
<td>&lt;xdelta_value&gt; ::= X delta value in NR3 format</td>
</tr>
<tr>
<td></td>
<td>(see page 388)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:SCALE:Y1 &lt;y1_value&gt;</td>
<td>:MTESt:SCALE:Y1?</td>
<td>&lt;y1_value&gt; ::= Y1 value in NR3 format</td>
</tr>
<tr>
<td></td>
<td>(see page 389)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:SCALE:Y2 &lt;y2_value&gt;</td>
<td>:MTESt:SCALE:Y2?</td>
<td>&lt;y2_value&gt; ::= Y2 value in NR3 format</td>
</tr>
<tr>
<td></td>
<td>(see page 390)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:SOURce &lt;source&gt;</td>
<td>:MTESt:SOURce?</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td>(see page 391)</td>
<td>&lt;n&gt; ::= {1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;n&gt; ::= {1</td>
</tr>
<tr>
<td>n/a</td>
<td>:MTESt:TITLe?</td>
<td>&lt;title&gt; ::= a string of up to 128 ASCII characters</td>
</tr>
<tr>
<td></td>
<td>(see page 392)</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:POD&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td></td>
<td>(see page 394)</td>
<td>{0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;n&gt; ::= 1-2 in NR1 format</td>
</tr>
<tr>
<td>:POD&lt;n&gt;:SIZE &lt;value&gt;</td>
<td>:POD&lt;n&gt;:SIZE?</td>
<td>&lt;value&gt; ::= {SMALl</td>
</tr>
<tr>
<td>(see page 395)</td>
<td>(see page 395)</td>
<td></td>
</tr>
<tr>
<td>:POD&lt;n&gt;:THReshold &lt;type&gt;[suffix]</td>
<td>:POD&lt;n&gt;:THReshold? (see page 396)</td>
<td>&lt;n&gt; ::= 1-2 in NR1 format</td>
</tr>
<tr>
<td></td>
<td>(see page 396)</td>
<td>&lt;type&gt; ::= {CMOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;user defined value&gt; ::= value in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[suffix] ::= {V</td>
</tr>
</tbody>
</table>
### Table 18 :RECall Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:RECall:FILENAME &lt;base_name&gt;  (see page 399)</td>
<td>:RECall:FILENAME? (see page 399)</td>
<td>&lt;base_name&gt; ::= quoted ASCII string</td>
</tr>
</tbody>
</table>
| :RECall:IMAGe[:START] [<file_spec>] (see page 400) | n/a | <file_spec> ::= {<internal_loc> | <file_name>}  
<internal_loc> ::= 0-9; an integer in NR1 format  
<file_name> ::= quoted ASCII string |
| :RECall:MASK[:START] [<file_spec>] (see page 401) | 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 402) | :RECall:PWD? (see page 402) | <path_name> ::= quoted ASCII string |
| :RECall:SETup[:START] [<file_spec>] (see page 403) | n/a | <file_spec> ::= {<internal_loc> | <file_name>}  
<internal_loc> ::= 0-9; an integer in NR1 format  
<file_name> ::= quoted ASCII string |

### Table 19 :SAVE Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SAVE:FILENAME &lt;base_name&gt;  (see page 406)</td>
<td>:SAVE:FILENAME? (see page 406)</td>
<td>&lt;base_name&gt; ::= quoted ASCII string</td>
</tr>
</tbody>
</table>
| :SAVE:IMAGe[:START] [<file_spec>] (see page 407) | n/a | <file_spec> ::= {<internal_loc> | <file_name>}  
<internal_loc> ::= 0-9; an integer in NR1 format  
<file_name> ::= quoted ASCII string |
| n/a | :SAVE:IMAGe:AREA? (see page 408) | <area> ::= {GRAT | SCR} |
Table 19  :SAVE Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SAVE:IMAGE:FACTors</td>
<td>:SAVE:IMAGE:FACTors?</td>
<td>{0</td>
</tr>
<tr>
<td>{(0</td>
<td>OFF)</td>
<td>(1</td>
</tr>
<tr>
<td>:SAVE:IMAGE:FORMat</td>
<td>:SAVE:IMAGE:FORMat?</td>
<td>&lt;format&gt; ::= {TIFF</td>
</tr>
<tr>
<td>&lt;format&gt;</td>
<td>(see page 410)</td>
<td>BMP24bit}</td>
</tr>
<tr>
<td>:SAVE:IMAGE:INKSaver</td>
<td>:SAVE:IMAGE:INKSaver?</td>
<td>{0</td>
</tr>
<tr>
<td>{(0</td>
<td>OFF)</td>
<td>(1</td>
</tr>
<tr>
<td>:SAVE:IMAGE:PALETTE</td>
<td>:SAVE:IMAGE:PALETTE?</td>
<td>&lt;palette&gt; ::= {COlor</td>
</tr>
<tr>
<td>&lt;palette&gt;</td>
<td>(see page 412)</td>
<td></td>
</tr>
<tr>
<td>:SAVE:LISTer[:START]</td>
<td>n/a</td>
<td>&lt;file_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>[&lt;file_name&gt;]</td>
<td>(see page 413)</td>
<td></td>
</tr>
<tr>
<td>:SAVE:MASK[:START]</td>
<td>n/a</td>
<td>&lt;file_spec&gt; ::= {&lt;internal_loc&gt;</td>
</tr>
<tr>
<td>[&lt;file_spec&gt;]</td>
<td>(see page 414)</td>
<td>&lt;file_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>:SAVE:PWD &lt;path_name&gt;</td>
<td>:SAVE:PWD? (see page 415)</td>
<td>&lt;path_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>(see page 415)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:SAVE:SETup[:START]</td>
<td>n/a</td>
<td>&lt;file_spec&gt; ::= {&lt;internal_loc&gt;</td>
</tr>
<tr>
<td>[&lt;file_spec&gt;]</td>
<td>(see page 416)</td>
<td>&lt;file_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>:SAVE:WAVEform[:START]</td>
<td>n/a</td>
<td>&lt;file_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>[&lt;file_name&gt;]</td>
<td>(see page 417)</td>
<td></td>
</tr>
<tr>
<td>:SAVE:WAVEform:FORMat</td>
<td>:SAVE:WAVEform:FORMat?</td>
<td>&lt;format&gt; ::= {ALB</td>
</tr>
<tr>
<td>&lt;format&gt;</td>
<td>(see page 418)</td>
<td>BINary</td>
</tr>
<tr>
<td>:SAVE:WAVEform:LENGTH</td>
<td>:SAVE:WAVEform:LENGTH?</td>
<td>&lt;length&gt; ::= 100 to max. length; an</td>
</tr>
<tr>
<td>&lt;length&gt;</td>
<td>(see page 419)</td>
<td>integer in NR1 format</td>
</tr>
<tr>
<td>&lt;option&gt;</td>
<td>(see page 420)</td>
<td></td>
</tr>
</tbody>
</table>
Table 20 :SBUS Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNT:ERROR? (see page 423)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNT:OVERload? (see page 424)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:CAN:COUNT:RESet</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNT:TOTAL? (see page 426)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNT:UTILization? (see page 427)</td>
<td>&lt;percent&gt; ::= floating-point in NR3 format</td>
</tr>
<tr>
<td>:SBUS:DISPLAY {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:FLEXray:COUNT:NULL? (see page 429)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:FLEXray:COUNT:RES</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:FLEXray:COUNT:SYNC? (see page 431)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:FLEXray:COUNT:TOTAL? (see page 432)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:I2S:BASE &lt;base&gt;</td>
<td>:SBUS:I2S:BASE? (see page 433)</td>
<td>&lt;base&gt; ::= {DECimal</td>
</tr>
<tr>
<td>:SBUS:IIC:ASIZE &lt;size&gt;</td>
<td>:SBUS:IIC:ASIZE? (see page 434)</td>
<td>&lt;size&gt; ::= {BIT7</td>
</tr>
<tr>
<td>:SBUS:LIN:PARity {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:SBUS:M1553:BASE &lt;base&gt;</td>
<td>:SBUS:M1553:BASE? (see page 436)</td>
<td>&lt;base&gt; ::= {DECimal</td>
</tr>
<tr>
<td>:SBUS:MODE &lt;mode&gt; (see page 437)</td>
<td>:SBUS:MODE? (see page 437)</td>
<td>&lt;mode&gt; ::= {CAN</td>
</tr>
<tr>
<td>:SBUS:SPI:BITorder &lt;order&gt; (see page 438)</td>
<td>:SBUS:SPI:BITorder? (see page 438)</td>
<td>&lt;order&gt; ::= {LSBFirst</td>
</tr>
</tbody>
</table>
### Table 20 :SBUS Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SBUS:SPI:WIDTh &lt;word_width&gt;</td>
<td>:SBUS:SPI:WIDTh?</td>
<td>&lt;word_width&gt; ::= integer 4-16 in NR1 format</td>
</tr>
<tr>
<td>:SBUS:UART:BASE &lt;base&gt;</td>
<td>:SBUS:UART:BASE?</td>
<td>&lt;base&gt; ::= {ASCii</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:UART:COUNt:ERROR?</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:UART:COUNt:RESET</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:UART:COUNt:RXFRAMES?</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:UART:COUNt:TXFRAMES?</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:UART:FRAMing &lt;value&gt;</td>
<td>:SBUS:UART:FRAMing?</td>
<td>&lt;value&gt; ::= {OFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;decimal&gt; ::= 8-bit integer from 0-255 (0x00-0xff)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;nondecimal&gt; ::= #Hnn where n ::= {0,..,9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;nondecimal&gt; ::= #Bnn...n where n ::= (0</td>
</tr>
</tbody>
</table>

### Table 21 :SYSTem Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SYSTem:DATE &lt;date&gt;</td>
<td>:SYSTem:DATE?</td>
<td>&lt;date&gt; ::= &lt;year&gt;,&lt;month&gt;,&lt;day&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;year&gt; ::= 4-digit year in NR1 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;month&gt; ::= {1,..,12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;day&gt; ::= {1,..31}</td>
</tr>
<tr>
<td>:SYSTem:DSP &lt;string&gt;</td>
<td>n/a</td>
<td>&lt;string&gt; ::= up to 254 characters as a quoted ASCII string</td>
</tr>
<tr>
<td>n/a</td>
<td>:SYSTem:ERROR?</td>
<td>&lt;error&gt; ::= an integer error code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;error string&gt; ::= quoted ASCII string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Error Messages (see page 745).</td>
</tr>
</tbody>
</table>
### Table 21 :SYSTem Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SYSTem:LOCK &lt;value&gt; (see page 450)</td>
<td>:SYSTem:LOCK? (see page 450)</td>
<td>&lt;value&gt; ::= {{1</td>
</tr>
<tr>
<td>:SYSTem:PRECision &lt;value&gt; (see page 451)</td>
<td>:SYSTem:PRECision? (see page 451)</td>
<td>&lt;value&gt; ::= {{1</td>
</tr>
<tr>
<td>:SYSTem:PROTection:LOCK &lt;value&gt; (see page 452)</td>
<td>:SYSTem:PROTection:LOCK? (see page 452)</td>
<td>&lt;value&gt; ::= {{1</td>
</tr>
<tr>
<td>:SYSTem:SETup &lt;setup_data&gt; (see page 453)</td>
<td>:SYSTem:SETup? (see page 453)</td>
<td>&lt;setup_data&gt; ::= data in IEEE 488.2 # format.</td>
</tr>
<tr>
<td>:SYSTem:TIME &lt;time&gt; (see page 455)</td>
<td>:SYSTem:TIME? (see page 455)</td>
<td>&lt;time&gt; ::= hours,minutes,seconds in NR1 format</td>
</tr>
</tbody>
</table>

### Table 22 :TIMebase Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TIMebase:MODE &lt;value&gt; (see page 458)</td>
<td>:TIMebase:MODE? (see page 458)</td>
<td>&lt;value&gt; ::= {MAIN</td>
</tr>
<tr>
<td>:TIMebase:POSition &lt;pos&gt; (see page 459)</td>
<td>:TIMebase:POSition? (see page 459)</td>
<td>&lt;pos&gt; ::= time from the trigger event to the display reference point in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:RANGe &lt;range_value&gt; (see page 460)</td>
<td>:TIMebase:RANGe? (see page 460)</td>
<td>&lt;range_value&gt; ::= 5 ns through 500 s in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:REFClock {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:TIMebase:REFerence {LEFT</td>
<td>CENTER</td>
<td>RIGHT} (see page 462)</td>
</tr>
<tr>
<td>:TIMebase:SCALe &lt;scale_value&gt; (see page 463)</td>
<td>:TIMebase:SCALe? (see page 463)</td>
<td>&lt;scale_value&gt; ::= scale value in seconds in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:VERNier {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
</tbody>
</table>
### Table 22: :TIMebase Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TIMebase:WINDow:POSITION &lt;pos&gt; (see page 465)</td>
<td>:TIMebase:WINDow:POSITION? (see page 465)</td>
<td>&lt;pos&gt; ::= time from the trigger event to the zoomed view reference point in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:WINDow:RANGE &lt;range_value&gt; (see page 466)</td>
<td>:TIMebase:WINDow:RANGE? (see page 466)</td>
<td>&lt;range_value&gt; ::= range value in seconds in NR3 format for the zoomed window</td>
</tr>
<tr>
<td>:TIMebase:WINDow:SCALE &lt;scale_value&gt; (see page 467)</td>
<td>:TIMebase:WINDow:SCALE? (see page 467)</td>
<td>&lt;scale_value&gt; ::= scale value in seconds in NR3 format for the zoomed window</td>
</tr>
</tbody>
</table>

### Table 23: General :TRIGger Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:HFReject {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:TRIGger:HOLDoff &lt;holdoff_time&gt; (see page 473)</td>
<td>:TRIGger:HOLDoff? (see page 473)</td>
<td>&lt;holdoff_time&gt; ::= 60 ns to 10 s in NR3 format</td>
</tr>
<tr>
<td>:TRIGger:LPIFty (see page 474)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:TRIGger:MODE &lt;mode&gt; (see page 475)</td>
<td>:TRIGger:MODE? (see page 475)</td>
<td>&lt;mode&gt; ::= {EDGE</td>
</tr>
<tr>
<td>:TRIGger:NREJect {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
</tbody>
</table>
### Table 23  General :TRIGger Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>] (see page 477) | :TRIGger:PATTern? (see page 478) | <value> ::= integer in NR1 format or <string>  
<mask> ::= integer in NR1 format or <string>  
<string> ::= "0xnnnnn"; n ::= {0,..,9 | A,..,F} (# bits = # channels)  
<edge source> ::= {CHANnel<n> | EXTernal | NONE} for DSO models  
<edge source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | NONE} for MSO models  
<edge> ::= {POSitive | NEGative}  
<n> ::= 1-2 or 1-4 in NR1 format |

| :TRIGger:SWEep <sweep> (see page 479) | :TRIGger:SWEep? (see page 479) | <sweep> ::= {AUTO | NORMal} |

### Table 24  :TRIGger:CAN Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :TRIGger:CAN:PATTern:DATA <value>, <mask> (see page 482) | :TRIGger:CAN:PATTern:DATA? (see page 482) | <value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS)  
<mask> ::= 64-bit integer in decimal, <nondecimal>, or <string>  
<nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal  
<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary  
<string> ::= "0xnnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal |

<p>| :TRIGger:CAN:PATTern:DATA:LENGth &lt;length&gt; (see page 483) | :TRIGger:CAN:PATTern:DATA:LENGth? (see page 483) | &lt;length&gt; ::= integer from 1 to 8 in NR1 format (with Option AMS) |</p>
<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:CAN:PATTern:ID &lt;value&gt;, &lt;mask&gt; (see page 484)</td>
<td>:TRIGger:CAN:PATTern:ID? (see page 484)</td>
<td>&lt;value&gt; ::= 32-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; (with Option AMS) &lt;mask&gt; ::= 32-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; &lt;nondecimal&gt; ::= #Hnn...n where n ::= (0,...,9</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTern:ID:MODE &lt;value&gt; (see page 485)</td>
<td>:TRIGger:CAN:PATTern:ID:MODE? (see page 485)</td>
<td>&lt;value&gt; ::= {STANdard</td>
</tr>
<tr>
<td>:TRIGger:CAN:SAMPLEpoint&lt;int &lt;value&gt; (see page 486)</td>
<td>:TRIGger:CAN:SAMPLEpoint&lt;int? (see page 486)</td>
<td>&lt;value&gt; ::= {60</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:B AUDrate &lt;baudrate&gt; (see page 487)</td>
<td>:TRIGger:CAN:SIGNal:B AUDrate? (see page 487)</td>
<td>&lt;baudrate&gt; ::= integer from 10000 to 100000 in 100 b/s increments</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:D EFinition &lt;value&gt; (see page 488)</td>
<td>:TRIGger:CAN:SIGNal:D EFinition? (see page 488)</td>
<td>&lt;value&gt; ::= {CANH</td>
</tr>
<tr>
<td>:TRIGger:CAN:SOURce &lt;source&gt; (see page 489)</td>
<td>:TRIGger:CAN:SOURce? (see page 489)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:CAN:TRIGger &lt;condition&gt; (see page 490)</td>
<td>:TRIGger:CAN:TRIGger? (see page 491)</td>
<td>&lt;condition&gt; ::= {SOF} (without Option AMS) &lt;condition&gt; ::= {SOF</td>
</tr>
</tbody>
</table>
### Table 25: \(\text{:TRIGger:DURation} \) Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{:TRIGger:DURation:GREaterthan} &lt;\text{greater_than_time}&gt;[\text{suffix}]) (see page 493)</td>
<td>(\text{:TRIGger:DURation:GREaterthan?}) (see page 493)</td>
<td>(&lt;\text{greater_than_time}&gt; ::= \text{floating-point number in NR3 format}) [\text{suffix}] ::= {s</td>
</tr>
<tr>
<td>(\text{:TRIGger:DURation:LESsthans} &lt;\text{less_than_time}&gt;[\text{suffix}]) (see page 494)</td>
<td>(\text{:TRIGger:DURation:LESsthans?}) (see page 494)</td>
<td>(&lt;\text{less_than_time}&gt; ::= \text{floating-point number from in NR3 format}) [\text{suffix}] ::= {s</td>
</tr>
<tr>
<td>(\text{:TRIGger:DURation:PATtern} &lt;\text{value}&gt;, &lt;\text{mask}&gt;) (see page 495)</td>
<td>(\text{:TRIGger:DURation:PATtern?}) (see page 495)</td>
<td>(&lt;\text{value}&gt; ::= \text{integer or &lt;string&gt;}) (&lt;\text{mask}&gt; ::= \text{integer or &lt;string&gt;}) (&lt;\text{string}&gt; ::= &quot;0xnnnnnn&quot; n ::= {0,..,9</td>
</tr>
<tr>
<td>(\text{:TRIGger:DURation:QUALifier} &lt;\text{qualifier}&gt;) (see page 496)</td>
<td>(\text{:TRIGger:DURation:QUALifier?}) (see page 496)</td>
<td>(&lt;\text{qualifier}&gt; ::= {\text{GREaterthan</td>
</tr>
<tr>
<td>(\text{:TRIGger:DURation:RANGE} &lt;\text{less_than_time}&gt;[\text{suffix}], &lt;\text{greater_than_time}&gt;[\text{suffix}]) (see page 497)</td>
<td>(\text{:TRIGger:DURation:RANGE?}) (see page 497)</td>
<td>(&lt;\text{less_than_time}&gt; ::= 15 \text{ ns to 10 seconds in NR3 format}) (&lt;\text{greater_than_time}&gt; ::= 10 \text{ ns to 9.99 seconds in NR3 format}) [\text{suffix}] ::= {s</td>
</tr>
</tbody>
</table>

### Table 26: \(\text{:TRIGger:EBURst} \) Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{:TRIGger:EBURst:COUNT} &lt;\text{count}&gt;) (see page 499)</td>
<td>(\text{:TRIGger:EBURst:COUNT?}) (see page 499)</td>
<td>(&lt;\text{count}&gt; ::= \text{integer in NR1 format}]</td>
</tr>
<tr>
<td>(\text{:TRIGger:EBURst:IDLE} &lt;\text{time_value}&gt;) (see page 500)</td>
<td>(\text{:TRIGger:EBURst:IDLE?}) (see page 500)</td>
<td>(&lt;\text{time_value}&gt; ::= \text{time in seconds in NR3 format}]</td>
</tr>
<tr>
<td>(\text{:TRIGger:EBURst:SLOPe} &lt;\text{slope}&gt;) (see page 501)</td>
<td>(\text{:TRIGger:EBURst:SLOPe?}) (see page 501)</td>
<td>(&lt;\text{slope}&gt; ::= {\text{NEGative</td>
</tr>
</tbody>
</table>
Table 27: :TRIGger[:EDGE] Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger[:EDGE]:COUPling {AC</td>
<td>DC</td>
<td>LF} (see page 503)</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:LEVel &lt;level&gt; [,&lt;source&gt;] (see page 504)</td>
<td>:TRIGger[:EDGE]:LEVel? [&lt;source&gt;] (see page 504)</td>
<td>For internal triggers, &lt;level&gt; ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, &lt;level&gt; ::= ±(external range setting) in NR3 format. For digital channels (MSO models), &lt;level&gt; ::= ±8 V. &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:REJec t {OFF</td>
<td>LF</td>
<td>HF} (see page 505)</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:SLOPe &lt;polarity&gt; (see page 506)</td>
<td>:TRIGger[:EDGE]:SLOPe? (see page 506)</td>
<td>&lt;polarity&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:SOURce &lt;source&gt; (see page 507)</td>
<td>:TRIGger[:EDGE]:SOURce? (see page 507)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>

Table 28: :TRIGger:FLEXray Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:FLEXray:AUTo setup (see page 509)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:BAUD rate &lt;baudrate&gt; (see page 510)</td>
<td>:TRIGger:FLEXray:BAUD rate? (see page 510)</td>
<td>&lt;baudrate&gt; ::= {2500000</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:CHAN nel &lt;channel&gt; (see page 511)</td>
<td>:TRIGger:FLEXray:CHAN nel? (see page 511)</td>
<td>&lt;channel&gt; ::= {A</td>
</tr>
</tbody>
</table>
Table 28  :TRIGger:FLEXray Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:FLEXray:ERRo&lt;br&gt;Type &lt;error_type&gt; (see page 512)</td>
<td>:TRIGger:FLEXray:ERRo&lt;br&gt;Type? (see page 512)</td>
<td>&lt;error_type&gt; ::= {ALL</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVERn&lt;br&gt;Type &lt;event&gt; (see page 513)</td>
<td>:TRIGger:FLEXray:EVERn&lt;br&gt;Type? (see page 513)</td>
<td>&lt;event&gt; ::= {WAKeup</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:CCBase&lt;br&gt;&lt;cycle_count_base&gt; (see page 514)</td>
<td>:TRIGger:FLEXray:FRAMe:CCBase? (see page 514)</td>
<td>&lt;cycle_count_base&gt; ::= integer from 0-63</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:CCRepetition&lt;br&gt;&lt;cycle_count_repetition&gt; (see page 515)</td>
<td>:TRIGger:FLEXray:FRAMe:CCRepetition? (see page 515)</td>
<td>&lt;cycle_count_repetition&gt; ::= {ALL</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:ID &lt;frame_id&gt; (see page 516)</td>
<td>:TRIGger:FLEXray:FRAMe:ID? (see page 516)</td>
<td>&lt;frame_id&gt; ::= {ALL</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:TYPE &lt;frame_type&gt; (see page 517)</td>
<td>:TRIGger:FLEXray:FRAMe:TYPE? (see page 517)</td>
<td>&lt;frame_type&gt; ::= {NORMal</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:SOURce &lt;source&gt; (see page 518)</td>
<td>:TRIGger:FLEXray:SOURce? (see page 518)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TRIGger &lt;condition&gt; (see page 519)</td>
<td>:TRIGger:FLEXray:TRIGger? (see page 519)</td>
<td>&lt;condition&gt; ::= {FRAME</td>
</tr>
</tbody>
</table>

Table 29  :TRIGger:GLITch Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:GLITch:GREAt&lt;br&gt;er than &lt;greater_than_time&gt;[suffix] (see page 522)</td>
<td>:TRIGger:GLITch:GREAt&lt;br&gt;er than? (see page 522)</td>
<td>&lt;greater_than_time&gt; ::= floating-point number in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:TRIGger:GLITch:LESSt&lt;br&gt;han &lt;less_than_time&gt;[suffix] (see page 523)</td>
<td>:TRIGger:GLITch:LESSt&lt;br&gt;han? (see page 523)</td>
<td>&lt;less_than_time&gt; ::= floating-point number in NR3 format [suffix] ::= {s</td>
</tr>
</tbody>
</table>
### Table 29 :TRIGger:GLITch Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:GLITch:LEVel &lt;level&gt; [&lt;source&gt;] (see page 524)</td>
<td>:TRIGger:GLITch:LEVel? (see page 524)</td>
<td>For internal triggers, &lt;level&gt; ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), &lt;level&gt; ::= ±(external range setting) in NR3 format. For digital channels (MSO models), &lt;level&gt; ::= ±8 V. &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:GLITch:POLarity &lt;polarity&gt; (see page 525)</td>
<td>:TRIGger:GLITch:POLarity? (see page 525)</td>
<td>&lt;polarity&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger:GLITch:QUALifier &lt;qualifier&gt; (see page 526)</td>
<td>:TRIGger:GLITch:QUALifier? (see page 526)</td>
<td>&lt;qualifier&gt; ::= {GREaterthan</td>
</tr>
<tr>
<td>:TRIGger:GLITch:RANGe &lt;less_than_time&gt;[suffix], &lt;greater_than_time&gt;[suffix] (see page 527)</td>
<td>:TRIGger:GLITch:RANGe? (see page 527)</td>
<td>&lt;less_than_time&gt; ::= 15 ns to 10 seconds in NR3 format &lt;greater_than_time&gt; ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:TRIGger:GLITch:SOURce &lt;source&gt; (see page 528)</td>
<td>:TRIGger:GLITch:SOURce? (see page 528)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>

### Table 30 :TRIGger:I2S Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:I2S:ALIGnmen_t &lt;setting&gt; (see page 531)</td>
<td>:TRIGger:I2S:ALIGnmen_t? (see page 531)</td>
<td>&lt;setting&gt; ::= {I2S</td>
</tr>
<tr>
<td>:TRIGger:I2S:AUDio &lt;audio_ch&gt; (see page 532)</td>
<td>:TRIGger:I2S:AUDio? (see page 532)</td>
<td>&lt;audio_ch&gt; ::= {RIGHT</td>
</tr>
</tbody>
</table>
## Table 30  :TRIGger:I2S Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;slope&gt; ::= {NEGative</td>
<td>POSitive}</td>
<td>(see page 533)</td>
</tr>
<tr>
<td>&lt;string&gt; ::= &quot;nn...n&quot; where n ::= {0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>&lt;string&gt; ::= &quot;0xnn...n&quot; where n ::= {0,..,9</td>
<td>A,..,F</td>
<td>X</td>
</tr>
<tr>
<td>:TRIGger:I2S:PATTERN:</td>
<td>:TRIGger:I2S:PATTERN:</td>
<td>&lt;base&gt; ::= {BINary</td>
</tr>
<tr>
<td>FORMat &lt;base&gt;</td>
<td>FORMat?</td>
<td>(see page 536)</td>
</tr>
<tr>
<td>:TRIGger:I2S:RANGE</td>
<td>:TRIGger:I2S:RANGE?</td>
<td>&lt;upper&gt; ::= 32-bit integer in signed decimal, &lt;nondecimal&gt;, or &lt;string&gt;</td>
</tr>
<tr>
<td>&lt;upper&gt;,&lt;lower&gt;</td>
<td>(see page 537)</td>
<td>&lt;lower&gt; ::= 32-bit integer in signed decimal, &lt;nondecimal&gt;, or &lt;string&gt;</td>
</tr>
<tr>
<td>&lt;nondecimal&gt; ::= #Hnn...n where n ::= {0,..,9</td>
<td>A,..,F} for hexadecimal</td>
<td></td>
</tr>
<tr>
<td>&lt;nondecimal&gt; ::= #Bnn...n where n ::= {0</td>
<td>1} for binary</td>
<td></td>
</tr>
<tr>
<td>&lt;string&gt; ::= &quot;0xnn...n&quot; where n ::= {0,..,9</td>
<td>A,..,F} for hexadecimal</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:RWIDTH</td>
<td>:TRIGger:I2S:RWIDTH?</td>
<td>&lt;receiver&gt; ::= 4-32 in NR1 format</td>
</tr>
<tr>
<td>&lt;receiver&gt;</td>
<td>(see page 539)</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:SOURce:CLOCK</td>
<td>:TRIGger:I2S:SOURce:CLOCK?</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
<td>EXTernal} for DSO models</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:SOURce:DATA</td>
<td>:TRIGger:I2S:SOURce:DATA?</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
<td>DIGital0,...,DIGital15 } for MSO models</td>
<td></td>
</tr>
<tr>
<td>&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
<td>(see page 540)</td>
<td>(see page 541)</td>
</tr>
</tbody>
</table>

Agilent InfiniiVision 7000A Series Oscilloscopes Programmer’s Guide
### Table 30: `:TRIGger:I2S` Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:TRIGger:I2S:SOURce:W SELect &lt;source&gt;</code> (see page 542)</td>
<td><code>:TRIGger:I2S:SOURce:W SELect?</code> (see page 542)</td>
<td>`&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td><code>:TRIGger:I2S:TRIGger &lt;operator&gt;</code> (see page 543)</td>
<td><code>:TRIGger:I2S:TRIGger?</code> (see page 543)</td>
<td>`&lt;operator&gt; ::= {EQUal</td>
</tr>
<tr>
<td><code>:TRIGger:I2S:TWIDth &lt;word_size&gt;</code> (see page 545)</td>
<td><code>:TRIGger:I2S:TWIDth?</code> (see page 545)</td>
<td>`&lt;word_size&gt; ::= 4-32 in NR1 format</td>
</tr>
<tr>
<td><code>:TRIGger:I2S:WSLow &lt;low_def&gt;</code> (see page 546)</td>
<td><code>:TRIGger:I2S:WSLow?</code> (see page 546)</td>
<td>`&lt;low_def&gt; ::= {LEFT</td>
</tr>
</tbody>
</table>

### Table 31: `:TRIGger:IIC` Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:TRIGger:IIC:PATTern:ADDRess &lt;value&gt;</code> (see page 548)</td>
<td><code>:TRIGger:IIC:PATTern:ADDRess?</code> (see page 548)</td>
<td>`&lt;value&gt; ::= integer or &lt;string&gt; &lt;string&gt; ::= &quot;0xnn&quot; n ::= {0,...,9</td>
</tr>
<tr>
<td><code>:TRIGger:IIC:PATTern:DATA &lt;value&gt;</code> (see page 549)</td>
<td><code>:TRIGger:IIC:PATTern:DATA?</code> (see page 549)</td>
<td>`&lt;value&gt; ::= integer or &lt;string&gt; &lt;string&gt; ::= &quot;0xnn&quot; n ::= {0,...,9</td>
</tr>
<tr>
<td><code>:TRIGger:IIC:PATTern:DATa2 &lt;value&gt;</code> (see page 550)</td>
<td><code>:TRIGger:IIC:PATTern:DATa2?</code> (see page 550)</td>
<td>`&lt;value&gt; ::= integer or &lt;string&gt; &lt;string&gt; ::= &quot;0xnn&quot; n ::= {0,...,9</td>
</tr>
<tr>
<td><code>:TRIGger:IIC[:SOURce]:CLOCk &lt;source&gt;</code> (see page 551)</td>
<td><code>:TRIGger:IIC[:SOURce]:CLOCk?</code> (see page 551)</td>
<td>`&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td><code>:TRIGger:IIC[:SOURce]:DATA &lt;source&gt;</code> (see page 552)</td>
<td><code>:TRIGger:IIC[:SOURce]:DATA?</code> (see page 552)</td>
<td>`&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
### Table 31: \texttt{:TRIGger:IIC} Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{:TRIGger:IIC:TRIGger:QUALifier &lt;value&gt;} (see page 553)</td>
<td>\texttt{:TRIGger:IIC:TRIGger:QUALifier?} (see page 553)</td>
<td>\texttt{&lt;value&gt;} ::= {EQUal</td>
</tr>
<tr>
<td>\texttt{:TRIGger:IIC:TRIGger[:TYPE] &lt;type&gt;} (see page 554)</td>
<td>\texttt{:TRIGger:IIC:TRIGger[:TYPE]?} (see page 554)</td>
<td>\texttt{&lt;type&gt;} ::= {START</td>
</tr>
</tbody>
</table>

### Table 32: \texttt{:TRIGger:LIN} Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{:TRIGger:LIN:ID &lt;value&gt;} (see page 558)</td>
<td>\texttt{:TRIGger:LIN:ID?} (see page 558)</td>
<td>\texttt{&lt;value&gt;} ::= 7-bit integer in decimal, \langle\textit{nondecimal}\rangle, or \langle\textit{string}\rangle from 0–63 or 0x00–0x3f (with Option AMS) &lt;\textit{nondecimal}\rangle ::= #Hnn where \textit{n} ::= {0,...,9</td>
</tr>
<tr>
<td>\texttt{:TRIGger:LIN:PATTern:DATA &lt;string&gt;} (see page 559)</td>
<td>\texttt{:TRIGger:LIN:PATTern:DATA?} (see page 560)</td>
<td>\texttt{&lt;string&gt;} ::= &quot;n&quot; where \textit{n} ::= 32-bit integer in signed decimal when \texttt{&lt;base&gt;} = DECimal &lt;\textit{string}\rangle ::= &quot;nn...n&quot; where \textit{n} ::= {0</td>
</tr>
<tr>
<td>\texttt{:TRIGger:LIN:PATTern:DATA:LENGTH &lt;length&gt;} (see page 561)</td>
<td>\texttt{:TRIGger:LIN:PATTern:DATA:LENGTH?} (see page 561)</td>
<td>\texttt{&lt;length&gt;} ::= integer from 1 to 8 in NR1 format</td>
</tr>
<tr>
<td>\texttt{:TRIGger:LIN:PATTern:FORMat &lt;base&gt;} (see page 562)</td>
<td>\texttt{:TRIGger:LIN:PATTern:FORMat?} (see page 562)</td>
<td>\texttt{&lt;base&gt;} ::= {BINary</td>
</tr>
<tr>
<td>\texttt{:TRIGger:LIN:SAMPlepo\nt &lt;value&gt;} (see page 563)</td>
<td>\texttt{:TRIGger:LIN:SAMPlepo\nt?} (see page 563)</td>
<td>\texttt{&lt;value&gt;} ::= {60</td>
</tr>
<tr>
<td>\texttt{:TRIGger:LIN:SIGNal:B\AUDrate &lt;baudrate&gt;} (see page 564)</td>
<td>\texttt{:TRIGger:LIN:SIGNal:B\AUDrate?} (see page 564)</td>
<td>\texttt{&lt;baudrate&gt;} ::= integer from 2400 to 625000 in 100 b/s increments</td>
</tr>
</tbody>
</table>
Table 32: :TRIGger:LIN Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:LIN:SOURce &lt;source&gt; (see page 565)</td>
<td>:TRIGger:LIN:SOURce? (see page 565)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:LIN:STANdard &lt;std&gt; (see page 566)</td>
<td>:TRIGger:LIN:STANdard? (see page 566)</td>
<td>&lt;std&gt; ::= {LIN13</td>
</tr>
<tr>
<td>:TRIGger:LIN:SYNCbreak &lt;value&gt; (see page 567)</td>
<td>:TRIGger:LIN:SYNCbreak? (see page 567)</td>
<td>&lt;value&gt; ::= integer = {11</td>
</tr>
<tr>
<td>:TRIGger:LIN:TRIGger &lt;condition&gt; (see page 568)</td>
<td>:TRIGger:LIN:TRIGger? (see page 568)</td>
<td>&lt;condition&gt; ::= {SYNCbreak} (without Option AMS) &lt;condition&gt; ::= {SYNCbreak</td>
</tr>
</tbody>
</table>

Table 33: :TRIGger:M1553 Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:M1553:AUTose tup (see page 570)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:TRIGger:M1553:PATTern n:DATA &lt;string&gt; (see page 571)</td>
<td>:TRIGger:M1553:PATTern n:DATA? (see page 571)</td>
<td>&lt;string&gt; ::= &quot;nn...n&quot; where n ::= {0</td>
</tr>
<tr>
<td>:TRIGger:M1553:RTA &lt;value&gt; (see page 572)</td>
<td>:TRIGger:M1553:RTA? (see page 572)</td>
<td>&lt;value&gt; ::= 5-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; from 0-31 &lt;nondecimal&gt; ::= #Hnn where n ::= {0,...,9</td>
</tr>
<tr>
<td>:TRIGger:M1553:SOURce :LOWer &lt;source&gt; (see page 573)</td>
<td>:TRIGger:M1553:SOURce :LOWer? (see page 573)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= {2</td>
</tr>
<tr>
<td>:TRIGger:M1553:SOURce :UPPer &lt;source&gt; (see page 574)</td>
<td>:TRIGger:M1553:SOURce :UPPer? (see page 574)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= {1</td>
</tr>
<tr>
<td>:TRIGger:M1553:TYPE &lt;type&gt; (see page 575)</td>
<td>:TRIGger:M1553:TYPE? (see page 575)</td>
<td>&lt;type&gt; ::= {DSTArt</td>
</tr>
</tbody>
</table>
### Table 34: :TRIGger:SEQUence Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:SEQUence:COUNT &lt;count&gt; (see page 577)</td>
<td>:TRIGger:SEQUence:COUNT? (see page 577)</td>
<td>&lt;count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:SEQUence:EDGE{1</td>
<td>2} &lt;source&gt;, &lt;slope&gt; (see page 578)</td>
<td>:TRIGger:SEQUence:EDGE{1</td>
</tr>
<tr>
<td>:TRIGger:SEQUence:FIN D &lt;value&gt; (see page 579)</td>
<td>:TRIGger:SEQUence:FIN D? (see page 579)</td>
<td>&lt;value&gt; ::= {PATTern1,ENTERed</td>
</tr>
<tr>
<td>:TRIGger:SEQUence:PATTERN{1</td>
<td>2} &lt;value&gt;, &lt;mask&gt; (see page 580)</td>
<td>:TRIGger:SEQUence:PATTERN{1</td>
</tr>
<tr>
<td>:TRIGger:SEQUence:REST et &lt;value&gt; (see page 581)</td>
<td>:TRIGger:SEQUence:REST et? (see page 581)</td>
<td>&lt;value&gt; ::= {NONE</td>
</tr>
<tr>
<td>:TRIGger:SEQUence:TIMER &lt;time_value&gt; (see page 582)</td>
<td>:TRIGger:SEQUence:TIMER? (see page 582)</td>
<td>&lt;time_value&gt; ::= time from 10 ns to 10 seconds in NR3 format</td>
</tr>
<tr>
<td>:TRIGger:SEQUence:TRIGGER &lt;value&gt; (see page 583)</td>
<td>:TRIGger:SEQUence:TRIGGER? (see page 583)</td>
<td>&lt;value&gt; ::= {PATTern2,ENTERed</td>
</tr>
</tbody>
</table>
### Table 35: :TRIGger:SPI Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:SPI:CLOCK:SL OPe &lt;slope&gt; (see page 585)</td>
<td>:TRIGger:SPI:CLOCK:SL OPe? (see page 585)</td>
<td>&lt;slope&gt; ::= {NEGative</td>
</tr>
<tr>
<td>:TRIGger:SPI:CLOCK:TI Meout &lt;time_value&gt; (see page 586)</td>
<td>:TRIGger:SPI:CLOCK:TI Meout? (see page 586)</td>
<td>&lt;time_value&gt; ::= time in seconds in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:SPI:FRAMing &lt;value&gt; (see page 587)</td>
<td>:TRIGger:SPI:FRAMing? (see page 587)</td>
<td>&lt;value&gt; ::= {CHIPselect</td>
</tr>
<tr>
<td>:TRIGger:SPI:PATTern:DATA &lt;value&gt;, &lt;mask&gt; (see page 588)</td>
<td>:TRIGger:SPI:PATTern:DATA? (see page 588)</td>
<td>&lt;value&gt; ::= integer or &lt;string&gt; &lt;mask&gt; ::= integer or &lt;string&gt; &lt;string&gt; ::= &quot;0xnnnnnn&quot; where n ::= (0,..,9</td>
</tr>
<tr>
<td>:TRIGger:SPI:PATTern:WIDTh &lt;width&gt; (see page 589)</td>
<td>:TRIGger:SPI:PATTern:WIDTh? (see page 589)</td>
<td>&lt;width&gt; ::= integer from 4 to 32 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:CLOCk &lt;source&gt; (see page 590)</td>
<td>:TRIGger:SPI:SOURce:CLOCk? (see page 590)</td>
<td>&lt;value&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:DATA &lt;source&gt; (see page 591)</td>
<td>:TRIGger:SPI:SOURce:DATA? (see page 591)</td>
<td>&lt;value&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:FRAMe &lt;source&gt; (see page 592)</td>
<td>:TRIGger:SPI:SOURce:FRAMe? (see page 592)</td>
<td>&lt;value&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
### Table 36 :TRIGger:TV Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:TV:LINE &lt;line number&gt; (see page 594)</td>
<td>:TRIGger:TV:LINE? (see page 594)</td>
<td>&lt;line number&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:TV:MODE &lt;tv mode&gt; (see page 595)</td>
<td>:TRIGger:TV:MODE? (see page 595)</td>
<td>&lt;tv mode&gt; ::= {FILEd1</td>
</tr>
<tr>
<td>:TRIGger:TV:POLarity &lt;polarity&gt; (see page 596)</td>
<td>:TRIGger:TV:POLarity? (see page 596)</td>
<td>&lt;polarity&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger:TV:SOURce &lt;source&gt; (see page 597)</td>
<td>:TRIGger:TV:SOURce? (see page 597)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= 1-2 or 1-4 integer in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:TV:STANdard &lt;standard&gt; (see page 598)</td>
<td>:TRIGger:TV:STANdard? (see page 598)</td>
<td>&lt;standard&gt; ::= {GENeric</td>
</tr>
</tbody>
</table>

### Table 37 :TRIGger:UART Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:UART:BASE &lt;base&gt; (see page 601)</td>
<td>:TRIGger:UART:BASE? (see page 601)</td>
<td>&lt;base&gt; ::= {ASCii</td>
</tr>
<tr>
<td>:TRIGger:UART:BAUDrate &lt;baudrate&gt; (see page 602)</td>
<td>:TRIGger:UART:BAUDrate? (see page 602)</td>
<td>&lt;baudrate&gt; ::= integer from 1200 to 3000000 in 100 b/s increments</td>
</tr>
<tr>
<td>:TRIGger:UART:BITorde r &lt;bitorder&gt; (see page 603)</td>
<td>:TRIGger:UART:BITorde r? (see page 603)</td>
<td>&lt;bitorder&gt; ::= {LSBFirst</td>
</tr>
<tr>
<td>:TRIGger:UART:BURSt &lt;value&gt; (see page 604)</td>
<td>:TRIGger:UART:BURSt? (see page 604)</td>
<td>&lt;value&gt; ::= {OFF</td>
</tr>
</tbody>
</table>
### Table 37: :TRIGger:UART Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:UART:DATA &lt;value&gt; (see page 605)</td>
<td>:TRIGger:UART:DATA? (see page 605)</td>
<td>&lt;value&gt; ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, hexadecimal, binary, or quoted_string format &lt;hexadecimal&gt; ::= #Hnn where n ::= {0,..,9</td>
</tr>
<tr>
<td>:TRIGger:UART:IDLE &lt;time_value&gt; (see page 606)</td>
<td>:TRIGger:UART:IDLE? (see page 606)</td>
<td>&lt;time_value&gt; ::= time from 1 us to 10 s in NR3 format</td>
</tr>
<tr>
<td>:TRIGger:UART:PARity &lt;parity&gt; (see page 607)</td>
<td>:TRIGger:UART:PARity? (see page 607)</td>
<td>&lt;parity&gt; ::= {EVEN</td>
</tr>
<tr>
<td>:TRIGger:UART:POLarity &lt;polarity&gt; (see page 608)</td>
<td>:TRIGger:UART:POLarity? (see page 608)</td>
<td>&lt;polarity&gt; ::= {HIGH</td>
</tr>
<tr>
<td>:TRIGger:UART:QUALifier &lt;value&gt; (see page 609)</td>
<td>:TRIGger:UART:QUALifier? (see page 609)</td>
<td>&lt;value&gt; ::= {EQUal</td>
</tr>
<tr>
<td>:TRIGger:UART:SOURce:RX &lt;source&gt; (see page 610)</td>
<td>:TRIGger:UART:SOURce:RX? (see page 610)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:UART:SOURce:TX &lt;source&gt; (see page 611)</td>
<td>:TRIGger:UART:SOURce:TX? (see page 611)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:UART:TYPE &lt;value&gt; (see page 612)</td>
<td>:TRIGger:UART:TYPE? (see page 612)</td>
<td>&lt;value&gt; ::= {RSTArt</td>
</tr>
<tr>
<td>:TRIGger:UART:WIDTH &lt;width&gt; (see page 613)</td>
<td>:TRIGger:UART:WIDTH? (see page 613)</td>
<td>&lt;width&gt; ::= {5</td>
</tr>
</tbody>
</table>
### Table 38: :TRIGger:USB Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:USB:SOURce:D MINus &lt;source&gt; (see page 615)</td>
<td>:TRIGger:USB:SOURce:D MINus? (see page 615)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:USB:SOURce:D PLus &lt;source&gt; (see page 616)</td>
<td>:TRIGger:USB:SOURce:D PLus? (see page 616)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:USB:SPEed &lt;value&gt; (see page 617)</td>
<td>:TRIGger:USB:SPEed? (see page 617)</td>
<td>&lt;value&gt; ::= {LOW</td>
</tr>
<tr>
<td>:TRIGger:USB:TRIGger &lt;value&gt; (see page 618)</td>
<td>:TRIGger:USB:TRIGger? (see page 618)</td>
<td>&lt;value&gt; ::= {SOP</td>
</tr>
</tbody>
</table>

### Table 39: :WAVeform Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:WAVeform:BYTeorder &lt;value&gt; (see page 627)</td>
<td>:WAVeform:BYTeorder? (see page 627)</td>
<td>&lt;value&gt; ::= {LSBFirst</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:COUNt? (see page 628)</td>
<td>&lt;count&gt; ::= an integer from 1 to 65536 in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:DATA? (see page 629)</td>
<td>&lt;binary block length bytes&gt;, &lt;binary data&gt; For example, to transmit 1000 bytes of data, the syntax would be: #800001000&lt;1000 bytes of data&gt;&lt;NL&gt; 8 is the number of digits that follow 0001000 is the number of bytes to be transmitted &lt;1000 bytes of data&gt; is the actual data</td>
</tr>
<tr>
<td>:WAVeform:FORMat &lt;value&gt; (see page 631)</td>
<td>:WAVeform:FORMat? (see page 631)</td>
<td>&lt;value&gt; ::= {WORD</td>
</tr>
</tbody>
</table>
## Table 39: :WAVEform Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:WAVEform:POINTS &lt;# points&gt; (see page 632)</td>
<td>:WAVEform:POINTS? (see page 632)</td>
<td>&lt;# points&gt; ::= {100</td>
</tr>
<tr>
<td>:WAVEform:POINTS:MODE &lt;points_mode&gt; (see page 634)</td>
<td>:WAVEform:POINTS:MODE? (see page 635)</td>
<td>&lt;points_mode&gt; ::= {NORMal</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVEform:PREAMble? (see page 636)</td>
<td>&lt;preamble_block&gt; ::= &lt;format NR1&gt;, &lt;type NR1&gt;, &lt;points NR1&gt;, &lt;count NR1&gt;, &lt;xincrement NR3&gt;, &lt;xorigin NR3&gt;, &lt;xreference NR1&gt;, &lt;yincrement NR3&gt;, &lt;yorigin NR3&gt;, &lt;yreference NR1&gt; &lt;format&gt; ::= an integer in NR1 format: • 0 for BYTE format • 1 for WORD format • 2 for ASCII format &lt;type&gt; ::= an integer in NR1 format: • 0 for NORMal type • 1 for PEAK detect type • 2 for AVERAGE type • 3 for HRESolution type &lt;count&gt; ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVEform:SEGmented:COUNT? (see page 639)</td>
<td>&lt;count&gt; ::= an integer from 2 to 2000 in NR1 format (with Option SGM)</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVEform:SEGmented:TAG? (see page 640)</td>
<td>&lt;time_tag&gt; ::= in NR3 format (with Option SGM)</td>
</tr>
</tbody>
</table>
### Table 39: :WAVeform Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:WAVeform:SOURce &lt;source&gt; (see page 641)</td>
<td>:WAVeform:SOURce? (see page 641)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:WAVeform:SOURce:SUBS&lt;subsource&gt; (see page 645)</td>
<td>:WAVeform:SOURce:SUBS&lt;subsource&gt;? (see page 645)</td>
<td>&lt;subsource&gt; ::= {NONE</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:TYPE? (see page 646)</td>
<td>&lt;return_mode&gt; ::= {NORM</td>
</tr>
<tr>
<td>:WAVeform:UNSIGNED {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:WAVeform:VIEW &lt;view&gt; (see page 648)</td>
<td>:WAVeform:VIEW? (see page 648)</td>
<td>&lt;view&gt; ::= {MAIN}</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:XINCREMENT? (see page 649)</td>
<td>&lt;return_value&gt; ::= x-increment in the current preamble in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:XORIGIN? (see page 650)</td>
<td>&lt;return_value&gt; ::= x-origin value in the current preamble in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:XREFERENCE? (see page 651)</td>
<td>&lt;return_value&gt; ::= 0 (x-reference value in the current preamble in NR1 format)</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:YINCREMENT? (see page 652)</td>
<td>&lt;return_value&gt; ::= y-increment value in the current preamble in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:YORIGIN? (see page 653)</td>
<td>&lt;return_value&gt; ::= y-origin in the current preamble in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:YREFERENCE? (see page 654)</td>
<td>&lt;return_value&gt; ::= y-reference value in the current preamble in NR1 format</td>
</tr>
</tbody>
</table>
Syntax Elements

- "Number Format" on page 108
- "<NL> (Line Terminator)" on page 108
- "[ ] (Optional Syntax Terms)" on page 108
- "{ } (Braces)" on page 108
- ":= (Defined As)" on page 108
- "< > (Angle Brackets)" on page 109
- "... (Ellipsis)" on page 109
- "n,...p (Value Ranges)" on page 109
- "d (Digits)" on page 109
- "Quoted ASCII String" on page 109
- "Definite-Length Block Response Data" on page 109

Number Format

NR1 specifies integer data.

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

<NL> (Line Terminator)

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

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

[ ] (Optional Syntax Terms)

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

{} (Braces)

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

::= (Defined As)

::= means "defined as".
For example, $<A> ::= <B>$ indicates that $<A>$ can be replaced by $<B>$ in any statement containing $<A>$.

$<>$ (Angle Brackets)

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

... (Ellipsis)

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

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

$n,...,p ::= \text{all integers between } n \text{ and } p \text{ inclusive.}$

$d$ (Digits)

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

Quoted ASCII String

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

```plaintext
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
## Commands by Subsystem

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Common (*) Commands&quot; on page 113</td>
<td>Commands defined by IEEE 488.2 standard that are common to all instruments.</td>
</tr>
<tr>
<td>&quot;Root (: ) Commands&quot; on page 139</td>
<td>Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.</td>
</tr>
<tr>
<td>&quot;:ACQuire Commands&quot; on page 181</td>
<td>Set the parameters for acquiring and storing data.</td>
</tr>
<tr>
<td>&quot;:BUS&lt;n&gt; Commands&quot; on page 198</td>
<td>Control all oscilloscope functions associated with the digital channels bus display.</td>
</tr>
<tr>
<td>&quot;:CALibrate Commands&quot; on page 207</td>
<td>Utility commands for determining the state of the calibration factor protection switch.</td>
</tr>
<tr>
<td>&quot;:CHANnel&lt;n&gt; Commands&quot; on page 217</td>
<td>Control all oscilloscope functions associated with individual analog channels or groups of channels.</td>
</tr>
<tr>
<td>&quot;:DIGital&lt;n&gt; Commands&quot; on page 237</td>
<td>Control all oscilloscope functions associated with individual digital channels.</td>
</tr>
<tr>
<td>&quot;:DISPLAY Commands&quot; on page 244</td>
<td>Control how waveforms, graticule, and text are displayed and written on the screen.</td>
</tr>
<tr>
<td>&quot;:EXTERNAL Trigger Commands&quot; on page 254</td>
<td>Control the input characteristics of the external trigger input.</td>
</tr>
<tr>
<td>&quot;:FUNCTION Commands&quot; on page 264</td>
<td>Control functions in the measurement/storage module.</td>
</tr>
<tr>
<td>&quot;:HARDcopy Commands&quot; on page 281</td>
<td>Set and query the selection of hardcopy device and formatting options.</td>
</tr>
<tr>
<td>&quot;:LISTer Commands&quot; on page 292</td>
<td>Turn on/off the List'er display for decoded serial data and get the List'er data.</td>
</tr>
<tr>
<td>&quot;:MARKer Commands&quot; on page 295</td>
<td>Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).</td>
</tr>
</tbody>
</table>
## Commands by Subsystem

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;MEASure Commands&quot; on page 306</td>
<td>Select automatic measurements to be made and control time markers.</td>
</tr>
<tr>
<td>&quot;MTESt Commands&quot; on page 359</td>
<td>Control the mask test features provided with Option LMT.</td>
</tr>
<tr>
<td>&quot;POD Commands&quot; on page 393</td>
<td>Control all oscilloscope functions associated with groups of digital channels.</td>
</tr>
<tr>
<td>&quot;RECall Commands&quot; on page 398</td>
<td>Recall previously saved oscilloscope setups and traces.</td>
</tr>
<tr>
<td>&quot;SAVE Commands&quot; on page 404</td>
<td>Save oscilloscope setups and traces, screen images, and data.</td>
</tr>
<tr>
<td>&quot;SBUS Commands&quot; on page 421</td>
<td>Control oscilloscope functions associated with the serial decode bus.</td>
</tr>
<tr>
<td>&quot;SYSTem Commands&quot; on page 446</td>
<td>Control basic system functions of the oscilloscope.</td>
</tr>
<tr>
<td>&quot;TIMebase Commands&quot; on page 456</td>
<td>Control all horizontal sweep functions.</td>
</tr>
<tr>
<td>&quot;TRIGger Commands&quot; on page 468</td>
<td>Control the trigger modes and parameters for each trigger type.</td>
</tr>
<tr>
<td>&quot;WAVEform Commands&quot; on page 619</td>
<td>Provide access to waveform data.</td>
</tr>
</tbody>
</table>

### Command Types

Three types of commands are used:

- **Common (*) Commands** — See "Introduction to Common (*) Commands" on page 116 for more information.

- **Root Level (:) Commands** — See "Introduction to Root (:) Commands" on page 141 for more information.

- **Subsystem Commands** — Subsystem commands are grouped together under a common node of the "Command Tree" on page 793, such as the :TIMebase commands. Only one subsystem may be selected at any given time. When the instrument is initially turned on, the command parser is set to the root of the command tree; therefore, no subsystem is selected.
Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See "Introduction to Common (*) Commands" on page 116.

Table 40  Common (*) Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CLS (see page 117)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>*ESE &lt;mask&gt; (see page 118)</td>
<td>*ESE? (see page 119)</td>
<td>&lt;mask&gt; ::= 0 to 255; an integer in NR1 format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Weight Name Enables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 128 PON Power On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 64 URQ User Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 32 CME Command Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 16 EXE Execution Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 8 DDE Dev. Dependent Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 4 QYE Query Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 2 RQL Request Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 OPC Operation Complete</td>
</tr>
<tr>
<td>n/a</td>
<td>*ESR? (see page 120)</td>
<td>&lt;status&gt; ::= 0 to 255; an integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>*IDN? (see page 120)</td>
<td>AGILENT TECHNOLOGIES,&lt;model&gt;,&lt;serial number&gt;,X.XX.XX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;model&gt; ::= the model number of the instrument</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;serial number&gt; ::= the serial number of the instrument</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;X.XX.XX&gt; ::= the software revision of the instrument</td>
</tr>
<tr>
<td>n/a</td>
<td>*LRN? (see page 123)</td>
<td>&lt;learn_string&gt; ::= current instrument setup as a block of data in IEEE 488.2 # format</td>
</tr>
<tr>
<td>*OPC (see page 124)</td>
<td>*OPC? (see page 124)</td>
<td>ASCII &quot;1&quot; is placed in the output queue when all pending device operations have completed.</td>
</tr>
</tbody>
</table>
### Commands by Subsystem

#### Table 40 Common (*) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>*OPT? (see page 125)</td>
<td></td>
</tr>
</tbody>
</table>

<return_value> ::= 0, 0, <license info>

<license info> ::= <All field>, <reserved>, <Factory MSO>, <Upgraded MSO>,

<Xilinx FPGA Probe>, <Memory>, <Low Speed Serial>,

<Automotive Serial>, <reserved>, <Secure>, <Battery>,

<Altera FPGA Probe>, <FlexRay Serial>,

<Power Measurements>, <RS-232/UART Serial>, <reserved>, <Segmented Memory>,

<Mask Test>, <reserved>, <reserved>, <FlexRay Conformance>, <reserved>,

<reserved>, <I2S Serial>, <FlexRay Trigger/Decode>,

<reserved>, <reserved>, <MIL-STD 1553 Trigger/Decode>, <reserved>

<All field> ::= (0 | All)

<reserved> ::= 0

<Factory MSO> ::= (0 | MSO)

<Upgraded MSO> ::= (0 | MSO)

<Xilinx FPGA Probe> ::= (0 | FPG)

<Memory> ::= (0 | mem2M | mem8M)

<Low Speed Serial> ::= (0 | LSS)

<Automotive Serial> ::= (0 | AMS)

<Secure> ::= (0 | SEC)

<Battery> ::= (0 | BAT)

<Altera FPGA Probe> ::= (0 | ALT)

<FlexRay Serial> ::= (0 | FRS)

<Power Measurements> ::= (0 | PWR)

<RS-232/UART Serial> ::= (0 | 232)

<Segmented Memory> ::= (0 | SGM)

<Mask Test> ::= (0 | LMT)

<FlexRay Conformance> ::= (0 | FRC)

<I2S Serial> ::= (0 | SND)

<FlexRay Trigger/Decode> ::= (0 | FLX)

<MIL-STD 1553 Trigger/Decode> ::= (0 | 553)
**Table 40** Common (*) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>*RCL &lt;value&gt; (see page 127)</td>
<td>n/a</td>
<td>&lt;value&gt; ::= {0</td>
</tr>
<tr>
<td>*RST (see page 128)</td>
<td>n/a</td>
<td>See *RST (Reset) (see page 128)</td>
</tr>
<tr>
<td>*SAV &lt;value&gt; (see page 131)</td>
<td>n/a</td>
<td>&lt;value&gt; ::= {0</td>
</tr>
<tr>
<td>*SRE &lt;mask&gt; (see page 132)</td>
<td>*SRE? (see page 133)</td>
<td>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. &lt;mask&gt; ::= following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Weight Name Enables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--- ------ ---- ----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 128 OPER Operation Status Reg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 64 ---- (Not used.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 32 ESB Event Status Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 16 MAV Message Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 8 ---- (Not used.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 4 MSG Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 2 USR User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 TRG Trigger</td>
</tr>
<tr>
<td>n/a</td>
<td>*STB? (see page 134)</td>
<td>&lt;value&gt; ::= 0 to 255; an integer in NR1 format, as shown in the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Weight Name <em>1</em> Indicates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--- ------ ---- ------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 128 OPER Operation status condition occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 64 RQS/ Instrument is MSS requesting service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 32 ESB Enabled event status condition occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 16 MAV Message available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 8 ---- (Not used.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 4 MSG Message displayed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 2 USR User event condition occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 TRG A trigger</td>
</tr>
<tr>
<td>*TRG (see page 136)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>*TST? (see page 137)</td>
<td>&lt;result&gt; ::= 0 or non-zero value; an integer in NR1 format</td>
</tr>
<tr>
<td>*WAI (see page 138)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
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 788)

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 116
- "*STB (Read Status Byte)" on page 134
- "*ESE (Standard Event Status Enable)" on page 118
- "*ESR (Standard Event Status Register)" on page 120
- "*SRE (Service Request Enable)" on page 132
- ":SYSTem:ERRor" on page 449
**ESE (Standard Event Status Enable)**

(see page 788)

**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 41: Standard Event Status Enable (ESE)**

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

---
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 116
- "*ESR (Standard Event Status Register)" on page 120
- "*OPC (Operation Complete)" on page 124
- "*CLS (Clear Status)" on page 117
**ESR (Standard Event Status Register)**

(see page 788)

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

Table 42: Standard Event Status Register (ESR)
Table 42  Standard Event Status Register (ESR) (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RQL</td>
<td>Request Control</td>
<td>The device is requesting control. (Not used.)</td>
</tr>
<tr>
<td>0</td>
<td>OPC</td>
<td>Operation Complete</td>
<td>Operation is complete.</td>
</tr>
</tbody>
</table>

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 116
- "*ESE (Standard Event Status Enable)" on page 118
- "*OPC (Operation Complete)" on page 124
- "*CLS (Clear Status)" on page 117
- ":SYSTem:ERRor" on page 449
**IDN (Identification Number)**

(see page 788)

**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 116
- "*OPT (Option Identification)" on page 125
**LRN (Learn Device Setup)**

(see page 788)

**Query Syntax**

*LRN?*

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see page 453) 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

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

**See Also**

- "Introduction to Common (*) Commands" on page 116
- "**RCL (Recall)" on page 127
- "**SAV (Save)" on page 131
- "*:SYSTem:SETup" on page 453
**OPC (Operation Complete)**

(see page 788)

**Command Syntax**

```plaintext
*OPC
```

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

**Query Syntax**

```plaintext
*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**

```plaintext
<complete><NL>
```

<complete> ::= 1

**See Also**

- "Introduction to Common (*) Commands" on page 116
- "*ESE (Standard Event Status Enable)" on page 118
- "*ESR (Standard Event Status Register)" on page 120
- "*CLS (Clear Status)" on page 117
*OPT (Option Identification)

(see page 788)

Query Syntax  *OPT?

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

Return Format  

0,0,<license info>

<license info> ::= <All field>,<reserved>,<Factory MSO>,<Upgraded MSO>, <Xilinx FPGA Probe>,<Memory>,<Low Speed Serial>, <Automotive Serial>,<reserved>,<Secure>,<Battery>, <Altera FPGA Probe>,<FlexRay Serial>, <Power Measurements>,<RS-232/UART Serial>,<reserved>, <Segmented Memory>,<Mask Test>,<reserved>,<reserved>, <FlexRay Conformance>,<reserved>,<reserved>, <I2S Serial>,<FlexRay Trigger/Decode>,<reserved>, <reserved>,<MIL-STD 1553 Trigger/Decode>,<reserved>

<All field> ::= (0 | All)

<reserved> ::= 0

<Factory MSO> ::= (0 | MSO)

<Upgraded MSO> ::= (0 | MSO)

<Xilinx FPGA Probe> ::= (0 | FPG)

<Memory> ::= (0 | mem2M | mem8M)

<Low Speed Serial> ::= (0 | LSS)

<Automotive Serial> ::= (0 | AMS)

<Secure> ::= (0 | SEC)

<Battery> ::= (0 | BAT)

<Altera FPGA Probe> ::= (0 | ALT)

<FlexRay Serial> ::= (0 | FRS)

<Power Measurements> ::= (0 | PWR)

<RS-232/UART Serial> ::= (0 | 232)

<Segmented Memory> ::= (0 | SGM)

<Mask Test> ::= (0 | LMT)

<FlexRay Conformance> ::= (0 | FRC)

<I2S Serial> ::= (0 | SND)

<FlexRay Trigger/Decode> ::= (0 | FLX)

<MIL-STD 1553 Trigger/Decode> ::= (0 | 553)
The <Factory MSO> <Upgraded MSO> fields indicate whether the unit is a mixed-signal oscilloscope and, if so, whether it was factory installed or upgraded from an analog channels only oscilloscope (DSO).

The *OPT? query returns the following:

<table>
<thead>
<tr>
<th>Module</th>
<th>Module Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>No modules attached</td>
<td>0,0,0,0,MSO,0,0,mem8M,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</td>
</tr>
</tbody>
</table>

See Also

- "Introduction to Common (*) Commands" on page 116
- "*IDN (Identification Number)" on page 122
**RCL (Recall)**

(see page 788)

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 116
- "*SAV (Save)" on page 131
### *RST (Reset)

*(see page 788)*

**Command Syntax**

```
*RST
```

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

<table>
<thead>
<tr>
<th>Acquire Menu</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Normal</td>
</tr>
<tr>
<td>Realtime</td>
<td>On</td>
</tr>
<tr>
<td>Averaging</td>
<td>Off</td>
</tr>
<tr>
<td># Averages</td>
<td>8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Cursor Menu</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Channel 1</td>
</tr>
</tbody>
</table>
### Digital Channel Menu (MSO models only)

<table>
<thead>
<tr>
<th>Channel 0 - 15</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labels</td>
<td>Off</td>
</tr>
<tr>
<td>Threshold</td>
<td>TTL (1.4V)</td>
</tr>
</tbody>
</table>

### Display Menu

<table>
<thead>
<tr>
<th>Definite persistence</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>33%</td>
</tr>
<tr>
<td>Vectors</td>
<td>On</td>
</tr>
</tbody>
</table>

### Quick Meas Menu

<table>
<thead>
<tr>
<th>Source</th>
<th>Channel 1</th>
</tr>
</thead>
</table>

### Run Control

- Scope is running

### Time Base Menu

<table>
<thead>
<tr>
<th>Main time/division</th>
<th>100 us</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main time base delay</td>
<td>0.00 s</td>
</tr>
<tr>
<td>Delay time/division</td>
<td>500 ns</td>
</tr>
<tr>
<td>Delay time base delay</td>
<td>0.00 s</td>
</tr>
<tr>
<td>Reference</td>
<td>center</td>
</tr>
<tr>
<td>Mode</td>
<td>main</td>
</tr>
<tr>
<td>Vernier</td>
<td>Off</td>
</tr>
</tbody>
</table>

### Trigger Menu

<table>
<thead>
<tr>
<th>Type</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Auto</td>
</tr>
<tr>
<td>Coupling</td>
<td>dc</td>
</tr>
<tr>
<td>Source</td>
<td>Channel 1</td>
</tr>
<tr>
<td>Level</td>
<td>0.0 V</td>
</tr>
</tbody>
</table>
Commands by Subsystem

### Trigger Menu

<table>
<thead>
<tr>
<th>Slope</th>
<th>Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>HF Reject and noise reject</td>
<td>Off</td>
</tr>
<tr>
<td>Holdoff</td>
<td>60 ns</td>
</tr>
<tr>
<td>External probe attenuation</td>
<td>AutoProbe (if AutoProbe is connected), otherwise 1.0:1</td>
</tr>
<tr>
<td>External Units</td>
<td>Volts</td>
</tr>
<tr>
<td>External Impedance</td>
<td>1 M Ohm</td>
</tr>
</tbody>
</table>

#### See Also

- "Introduction to Common (*) Commands" on page 116

#### Example Code

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

Example program from the start: "VISA COM Example in Visual Basic" on page 814
*SAV (Save)

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 116
- "*RCL (Recall)" on page 127
**SRE (Service Request Enable)**

(see page 788)

**Command Syntax**  
\[ \text{*SRE <mask>} \]

\(<\text{mask}> ::= \text{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.

![Diagram of service request enable register](image)

**Table 43**  Service Request Enable Register (SRE)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Enables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>OPER</td>
<td>Operation Status Register</td>
<td>Interrupts when enabled conditions in the Operation Status Register (OPER) occur.</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
<td>...</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>
**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 116
- "*STB (Read Status Byte)" on page 134
- "*CLS (Clear Status)" on page 117

---

**Table 43  Service Request Enable Register (SRE) (continued)**

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

---

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.
**STB (Read Status Byte)**

(see page 788)

**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>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>OPER</td>
<td>Operation Status Register</td>
<td>An enabled condition in the Operation Status Register (OPER) has occurred.</td>
</tr>
</tbody>
</table>
### Table 44  Status Byte Register (STB) (continued)

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

**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 116
- “*SRE (Service Request Enable)” on page 132
**TRG (Trigger)**

(see page 788)

**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 116
- ":DIGitize" on page 150
- ":RUN" on page 174
- ":STOP" on page 178
**TST (Self Test)**

(see page 788)

* **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 116
*WAI (Wait To Continue)

(see page 788)

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 116
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 141.

Table 45  Root (: ) Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :ACTivity (see page 142) | :ACTivity? (see page 142) | <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 143)   | (0 | 1); an integer in NR1 format                                                          |
| :AUToscale [<source>[],..,<source>] (see page 144) | n/a                    | <source> ::= CHANnel<n> for DSO models  
<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | POD1 | POD2} for MSO models  
<source> can be repeated up to 5 times  
<n> ::= 1-2 or 1-4 in NR1 format |
| :AUToscale:AMODE <value> (see page 146) | :AUToscale:AMODE? (see page 146) | <value> ::= {NORMal | CURRent} |
| :AUToscale:CHANnels <value> (see page 147) | :AUToscale:CHANnels? (see page 147) | <value> ::= {ALL | DISplayed}} |
| :BLANK [<source>] (see page 148) | n/a                    | <source> ::= {CHANnel<n>} | FUNCTION | MATH | SBUS} for DSO models  
<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | POD{1 | 2} | BUS{1 | 2} | FUNCTION | MATH | SBUS} for MSO models  
<n> ::= 1-2 or 1-4 in NR1 format |
| :CDISplay (see page 149) | n/a                    | n/a |
## Commands by Subsystem

### Table 45  Root (: ) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DIGitize [&lt;source&gt;[,...,&lt;source &gt;]] (see page 150)</td>
<td>n/a</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:HWEenable &lt;n&gt; (see page 152)</td>
<td>:HWEenable? (see page 152)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:HWERregister:CONDITION? (see page 154)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:MERGE &lt;pixel memory&gt; (see page 158)</td>
<td>n/a</td>
</tr>
<tr>
<td>:MTEenable &lt;n&gt; (see page 159)</td>
<td>:MTEenable? (see page 159)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:MTERegister[:EVENT]? (see page 161)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>:OPEE &lt;n&gt; (see page 163)</td>
<td>:OPEE? (see page 164)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:OPERregister:CONDITION? (see page 165)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:OPERegister[:EVENT]? (see page 167)</td>
<td>&lt;n&gt; ::= 16-bit integer in NR1 format</td>
</tr>
<tr>
<td>:OVLenable &lt;mask&gt; (see page 169)</td>
<td>:OVLenable? (see page 170)</td>
<td>&lt;mask&gt; ::= 16-bit integer in NR1 format as shown:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Weight Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--- ------ ---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 1024 Ext Trigger Fault</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 512 Channel 4 Fault</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 256 Channel 3 Fault</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 128 Channel 2 Fault</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 64 Channel 1 Fault</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 16 Ext Trigger OVL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 8 Channel 4 OVL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 4 Channel 3 OVL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 2 Channel 2 OVL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 Channel 1 OVL</td>
</tr>
</tbody>
</table>
### Table 45  Root (:) Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:OVLRegister? (see page 171)</td>
<td>&lt;value&gt; ::= integer in NR1 format. See OVLenable for &lt;value&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td>:PRINT [&lt;options&gt;] (see page 173)</td>
<td>&lt;options&gt; ::= [&lt;print option&gt;] [,..,&lt;print option&gt;]</td>
</tr>
<tr>
<td>n/a</td>
<td>:RUN (see page 174)</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SERial (see page 175)</td>
<td>&lt;return value&gt; ::= unquoted string containing serial number</td>
</tr>
<tr>
<td>n/a</td>
<td>:SINGLE (see page 176)</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:STATus? &lt;display&gt; (see page 177)</td>
<td>{0</td>
</tr>
<tr>
<td>n/a</td>
<td>:STOP (see page 178)</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:TER? (see page 179)</td>
<td>{0</td>
</tr>
<tr>
<td>n/a</td>
<td>:VIEW &lt;source&gt; (see page 180)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>

### 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.
5 Commands by Subsystem

:ACTivity

(see page 788)

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 141
- ":POD<n>:THReshold" on page 396
- ":DIGital<n>:THReshold" on page 243
:AER (Arm Event Register)

(see page 788)

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 141
- ":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
- "*STB (Read Status Byte)" on page 134
- "*SRE (Service Request Enable)" on page 132
5 Commands by Subsystem

:AUToscale

(see page 788)

Command Syntax

:AUToscale

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

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

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

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

<n> ::= (1 | 2) for the two channel oscilloscope models

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

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

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see ":AUToscale:CHANnels" on page 147) is set to DISPlayed channels. Then, the autoscale is performed.

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

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

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

- Cursors.
- Measurements.
- Trace memories.
- Zoomed (delayed) time base mode.

For further information on :AUToscale, see the User’s Guide.
See Also

- "Introduction to Root (:) Commands" on page 141
- ":AUToscale:CHANnels" on page 147
- ":AUToscale:AMODE" on page 146

Example Code

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

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:AUToscale:AMODE**

(see page 788)

**Command Syntax**

```
:AUToscale:AMODE <value>
```

<value> ::= {NORMal | CURRent}

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

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

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

**Query Syntax**

```
:AUTOscale:AMODE?
```

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

**Return Format**

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

<value> ::= (NORM | CURR)

**See Also**

- "Introduction to Root (:) Commands" on page 141
- ":AUToscale" on page 144
- ":AUToscale:CHANnels" on page 147
- ":ACQuire:TYPE" on page 196
- ":ACQuire:MODE" on page 187
`:AUToscale:CHANnels`

(see page 788)

**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 141
- "`:AUToscale" on page 144
- "`:AUToscale:AMODE" on page 146
- "`:VIEW" on page 180
- "`:BLANk" on page 148
5  Commands by Subsystem

:BLAnk

(see page 788)

Command Syntax

:BLAnk [<source>]

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

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

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

NOTE

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

NOTE

MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 141
- ".:CDISplay" on page 149
- ".:CHANnel<n>:DISPlay" on page 222
- ".:DIGital<n>:DISPlay" on page 239
- ".:FUNCtion:DISPlay" on page 268
- ".:POD<n>:DISPlay" on page 394
- ":STATus" on page 177
- ":VIEW" on page 180

Example Code

- "Example Code" on page 180
:CDISplay  
(see page 788)

**Command Syntax**  
:CDISplay

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

**See Also**
- "Introduction to Root (:) Commands" on page 141
- ":DISPlay:CLEar" on page 246
5 Commands by Subsystem

:DIGitize

(see page 788)

Command Syntax

:DIGitize [<source>[,...,<source>]]

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

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

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

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

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

NOTE MATH is an alias for FUNCTION.

See Also

- "Introduction to Root (: ) Commands" on page 141
- ":RUN" on page 174
- ":SINGle" on page 176
- ":STOP" on page 178
- ":ACQuire Commands" on page 181
- ":WAVeform Commands" on page 619

Example Code

' DIGITIZE - Used to acquire the waveform data for transfer over the interface. Sending this command causes an acquisition to take place with the resulting data being placed in the buffer.
'
' NOTE! The DIGITIZE command is highly recommended for triggering modes other than SINGLE. This ensures that sufficient data is available for measurement. If DIGITIZE is used with single mode, the completion criteria may never be met. The number of points gathered in Single mode is related to the sweep speed, memory depth, and maximum sample rate. For example, take an oscilloscope with a 1000-point memory, a sweep speed of 10 us/div (100 us total time across the screen), and a 20 MSa/s maximum sample rate.
' 1000 divided by 100 us equals 10 MSa/s. Because this number is
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition. Now, use 1 us/div
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;
' because this is greater than the maximum sample rate by 5 times,
' only 400 points (or 1/5 the points) can be gathered on a single
' trigger. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data buffers
' to be cleared and internal hardware to be reconfigured. If a
' measurement is immediately requested, there may have not been
' enough time for the data acquisition process to collect data, and
' the results may not be accurate. An error value of 9.9E+37 may be
' returned over the bus in this situation.
'
    myScope.WriteString "::DIGITIZE CHAN1"

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:HWEenable (Hardware Event Enable Register)**

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

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

* Query Syntax
  
  :HWEenable?

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

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

---

**Table 46  Hardware Event Enable Register (HWEenable)**

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

---

To HWE bit in Operation Status Condition Register
See Also

- "Introduction to Root (:) Commands" on page 141
- ":AER (Arm Event Register)" on page 143
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":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 134
- "**SRE (Service Request Enable)" on page 132
:HWERegister:CONDition (Hardware Event Condition Register)

(see page 788)

Query Syntax

:HWERegister:CONDition?

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

Return Format

<value><NL>

<value> ::= integer in NR1 format.

See Also

- "Introduction to Root (:) Commands" on page 141
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":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 134
• "SRE (Service Request Enable)" on page 132
**:HWERegister[:EVENt] (Hardware Event Event Register)**

(see page 788)

**Query Syntax**

:HWERegister[:EVENt]?

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

**Return Format**

\(<value><NL>\)

\(<value> ::= \text{integer in NR1 format.}\)

**See Also**

- "Introduction to Root (:) Commands" on page 141
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":OPEE (Operation Status Enable Register)" on page 163

---

**Table 48 Hardware Event Event Register**

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

---

![Diagram of Hardware Event Event Register and related registers](image-url)
- ":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 134
- "**SRE (Service Request Enable)" on page 132
5 Commands by Subsystem

:MERGe

(see page 788)

Command Syntax

::MERGe <pixel memory>

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

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

See Also

- "Introduction to Root (: Commands" on page 141
- "*:SAV (Save)" on page 131
- "*:RCL (Recall)" on page 127
- "*:VIEW" on page 180
- "*:BLANK" on page 148
**:MTEenable (Mask Test Event Enable Register)**

<see page 788>

**Command Syntax**

:MTEnable <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.

**Query Syntax**

:MTEenable?

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

---

**Table 49**  
Mask Test Event Enable Register (MTEenable)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Enables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-11</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>10</td>
<td>Auto Mask</td>
<td>Auto Mask Created</td>
<td>Auto mask creation completed.</td>
</tr>
<tr>
<td>9</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>8</td>
<td>Started</td>
<td>Mask Testing Started</td>
<td>Mask testing started.</td>
</tr>
<tr>
<td>7-2</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>1</td>
<td>Fail</td>
<td>Mask Test Fail</td>
<td>Mask test failed.</td>
</tr>
<tr>
<td>0</td>
<td>Complete</td>
<td>Mask Test Complete</td>
<td>Mask test is complete.</td>
</tr>
</tbody>
</table>
Return Format
<value><NL>
<value> ::= integer in NR1 format.

See Also
- "Introduction to Root (: Commands" on page 141
- ":AER (Arm Event Register)" on page 143
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":OPRRegister[: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 134
- "*:SRE (Service Request Enable)" on page 132
:MTERegister[:EVENt] (Mask Test Event Event Register)

(see page 788)

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 50  Mask Test Event Event Register

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

Return Format

<value><NL>

<value> ::= integer in NR1 format.

See Also

- "Introduction to Root (:) Commands" on page 141
- "CHANnel<n>:PROTection" on page 232
- "EXTernal:PROTection" on page 261
5  Commands by Subsystem

- "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 134
- "**SRE (Service Request Enable)" on page 132
**:OPEE (Operation Status Enable Register)**

(see page 788)

**Command Syntax**

```
:OPEE <mask>
```

<mask> ::= 16-bit integer

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

---

**Table 51  Operation Status Enable Register (OPEE)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Enables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-13</td>
<td>...</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>12</td>
<td>HWE</td>
<td>Hardware Event</td>
<td>Event when hardware event occurs.</td>
</tr>
<tr>
<td>11</td>
<td>OVLR</td>
<td>Overload</td>
<td>Event when 50Ω input overload occurs.</td>
</tr>
<tr>
<td>10</td>
<td>...</td>
<td>...</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>9</td>
<td>MTE</td>
<td>Mask Test Event</td>
<td>Event when mask test event occurs.</td>
</tr>
<tr>
<td>8-6</td>
<td>...</td>
<td>...</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>
5 Commands by Subsystem

Table 51 Operation Status Enable Register (OPEE) (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Enables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Wait Trig</td>
<td>Wait Trig</td>
<td>Event when the trigger is armed.</td>
</tr>
<tr>
<td>4</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>3</td>
<td>Run</td>
<td>Running</td>
<td>Event when the oscilloscope is running (not stopped).</td>
</tr>
<tr>
<td>2-0</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>

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 141
- "AER (Arm Event Register)" on page 143
- "CHANnel<n>:PROTection" on page 232
- "EXternal:PROTection" on page 261
- "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 134
- "SRE (Service Request Enable)" on page 132
:OPERegister:CONDition (Operation Status Condition Register)

(see page 788)

Query Syntax: :OPERegister:CONDition?

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-13</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>12</td>
<td>HWE</td>
<td>Hardware Event</td>
<td>A hardware event has occurred.</td>
</tr>
<tr>
<td>11</td>
<td>OVLR</td>
<td>Overload</td>
<td>A 50Ω input overload has occurred.</td>
</tr>
<tr>
<td>10</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>9</td>
<td>MTE</td>
<td>Mask Test Event</td>
<td>A mask test event has occurred.</td>
</tr>
<tr>
<td>8-6</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>5</td>
<td>Wait Trig</td>
<td>Wait Trig</td>
<td>The trigger is armed (set by the Trigger Armed Event Register (TER)).</td>
</tr>
<tr>
<td>4</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>
Table 52  Operation Status Condition Register (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Run</td>
<td>Running</td>
<td>The oscilloscope is running (not stopped).</td>
</tr>
<tr>
<td>2:0</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>

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

See Also
- "Introduction to Root (:) Commands" on page 141
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":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 134
- "*SRE (Service Request Enable)" on page 132
- ":HWERegister[:EVENT] (Hardware Event Event Register)" on page 156
- ":HWEenable (Hardware Event Enable Register)" on page 152
- ":MTERegister[:EVENT] (Mask Test Event Event Register)" on page 161
- ":MTEenable (Mask Test Event Enable Register)" on page 159
Commands by Subsystem

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

(see page 788)

Query Syntax

:OPERegister[:EVENt]?

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

Table 53 Operation Status Event Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True). Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-13</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>12</td>
<td>HWE</td>
<td>Hardware Event</td>
<td>A hardware event has occurred.</td>
</tr>
<tr>
<td>11</td>
<td>OVLR</td>
<td>Overload</td>
<td>A 50Ω input overload has occurred.</td>
</tr>
<tr>
<td>10</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>9</td>
<td>MTE</td>
<td>Mask Test Event</td>
<td>A mask test event has occurred.</td>
</tr>
<tr>
<td>8-6</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>5</td>
<td>Wait Trig</td>
<td>Wait Trig</td>
<td>The trigger is armed (set by the Trigger Armed Event Register (TER)).</td>
</tr>
<tr>
<td>4</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>
Table 53  Operation Status Event Register (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Run</td>
<td>Running</td>
<td>The oscilloscope has gone from a stop state to a single or running state.</td>
</tr>
<tr>
<td>2:0</td>
<td>---</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>

Return Format  

<value><NL>

<value> ::= integer in NR1 format.

See Also

- "Introduction to Root (:) Commands" on page 141
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":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 134
- "*SRE (Service Request Enable)" on page 132
- ":HWERegister[:EVENt] (Hardware Event Event Register)" on page 156
- ":HWEenable (Hardware Event Enable Register)" on page 152
- ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 161
- ":MTEenable (Mask Test Event Enable Register)" on page 159
:OVLenable (Overload Event Enable Register)

(see page 788)

**Command Syntax**

```
:OVLenable <enable_mask>
```

<enable_mask> ::= 16-bit integer

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

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

**NOTE**

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

---

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>When Set (1 = High = True), Enables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-11</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>10</td>
<td>External Trigger Fault</td>
<td>Event when fault occurs on External Trigger input.</td>
</tr>
<tr>
<td>9</td>
<td>Channel 4 Fault</td>
<td>Event when fault occurs on Channel 4 input.</td>
</tr>
<tr>
<td>8</td>
<td>Channel 3 Fault</td>
<td>Event when fault occurs on Channel 3 input.</td>
</tr>
</tbody>
</table>
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 141
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":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 134
- "*SRE (Service Request Enable)" on page 132

---

### Table 54 Overload Event Enable Register (OVL) (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>When Set (1 = High = True), Enables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Channel 2 Fault</td>
<td>Event when fault occurs on Channel 2 input.</td>
</tr>
<tr>
<td>6</td>
<td>Channel 1 Fault</td>
<td>Event when fault occurs on Channel 1 input.</td>
</tr>
<tr>
<td>5</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>4</td>
<td>External Trigger OVL</td>
<td>Event when overload occurs on External Trigger input.</td>
</tr>
<tr>
<td>3</td>
<td>Channel 4 OVL</td>
<td>Event when overload occurs on Channel 4 input.</td>
</tr>
<tr>
<td>2</td>
<td>Channel 3 OVL</td>
<td>Event when overload occurs on Channel 3 input.</td>
</tr>
<tr>
<td>1</td>
<td>Channel 2 OVL</td>
<td>Event when overload occurs on Channel 2 input.</td>
</tr>
<tr>
<td>0</td>
<td>Channel 1 OVL</td>
<td>Event when overload occurs on Channel 1 input.</td>
</tr>
</tbody>
</table>
:OVLRegister (Overload Event Register)

(see page 788)

Query Syntax

:OVLRegister?

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

NOTE

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

Table 55 Overload Event Register (OVLR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-11</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
<tr>
<td>10</td>
<td>External Trigger Fault</td>
<td>Fault has occurred on External Trigger input.</td>
</tr>
<tr>
<td>9</td>
<td>Channel 4 Fault</td>
<td>Fault has occurred on Channel 4 input.</td>
</tr>
<tr>
<td>8</td>
<td>Channel 3 Fault</td>
<td>Fault has occurred on Channel 3 input.</td>
</tr>
<tr>
<td>7</td>
<td>Channel 2 Fault</td>
<td>Fault has occurred on Channel 2 input.</td>
</tr>
<tr>
<td>6</td>
<td>Channel 1 Fault</td>
<td>Fault has occurred on Channel 1 input.</td>
</tr>
<tr>
<td>5</td>
<td>---</td>
<td>(Not used.)</td>
</tr>
</tbody>
</table>
Table 55  Overload Event Register (OVLR) (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>When Set (1 = High = True), Indicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>External Trigger OVL</td>
<td>Overload has occurred on External Trigger input.</td>
</tr>
<tr>
<td>3</td>
<td>Channel 4 OVL</td>
<td>Overload has occurred on Channel 4 input.</td>
</tr>
<tr>
<td>2</td>
<td>Channel 3 OVL</td>
<td>Overload has occurred on Channel 3 input.</td>
</tr>
<tr>
<td>1</td>
<td>Channel 2 OVL</td>
<td>Overload has occurred on Channel 2 input.</td>
</tr>
<tr>
<td>0</td>
<td>Channel 1 OVL</td>
<td>Overload has occurred on Channel 1 input.</td>
</tr>
</tbody>
</table>

Return Format

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

See Also

- "Introduction to Root (:) Commands" on page 141
- ":CHANnel<n>:PROTection" on page 232
- ":EXTernal:PROTection" on page 261
- ":OPEE (Operation Status Enable Register)" on page 163
- ":OVLenable (Overload Event Enable Register)" on page 169
- "*STB (Read Status Byte)" on page 134
- "*SRE (Service Request Enable)" on page 132
:PRINt

(see page 788)

Command Syntax

:PRINt [<options>]

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

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

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

The PRINt command formats the output according to the currently
selected format (device). If an option is not specified, the value selected in
the Print Config menu is used. Refer to ":HARDcopy:FORMat" on page 711
for more information.

See Also

- "Introduction to Root (:) Commands" on page 141
- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:FORMat" on page 711
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:GRAYscale" on page 712
- ":DISPlay:DATA" on page 247
**:RUN**

(see page 788)

**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 141
- ":.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.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:SERial

Query Syntax: :SERial?

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

Return Format: Unquoted string

See Also:
- "Introduction to Root (:) Commands" on page 141
**5  Commands by Subsystem**

### :SINGle

(see page 788)

**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 141
- ":RUN" on page 174
- ":STOP" on page 178
Commands by Subsystem

:STATus

N  (see page 788)

Query Syntax

:STATus? <source>

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

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

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

NOTE

MATH is an alias for FUNCtion.

Return Format

[value]<NL>

[value] ::= {1 | 0}

See Also

- "Introduction to Root (:) Commands" on page 141
- ":BLANK" on page 148
- ":CHANnel<n>:DISPlay" on page 222
- ":DIGital<n>:DISPlay" on page 239
- ":FUNCtion:DISPlay" on page 268
- ":POD<n>:DISPlay" on page 394
- ":VIEW" on page 180
5 Commands by Subsystem

:STOP

(see page 788)

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 141
- ":RUN" on page 174
- ":SINGle" on page 176

Example Code
- "Example Code" on page 174
**:TER (Trigger Event Register)**

(see page 788)

**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**

`${<\text{value}><\text{NL}}>$

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

**See Also**

- "Introduction to Root (:) Commands" on page 141
- "*SRE (Service Request Enable)" on page 132
- "*STB (Read Status Byte)" on page 134
5  Commands by Subsystem

:VIEW

(see page 788)

Command Syntax

:VIEW <source>

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

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

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

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

NOTE

MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 141
- ":BLANK" on page 148
- ":CHANnel<n>:DISPlay" on page 222
- ":DIGital<n>:DISPlay" on page 239
- ":FUNCtion:DISPlay" on page 268
- ":POD<n>:DISPlay" on page 394
- ":STATus" on page 177

Example Code

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

Example program from the start: "VISA COM Example in Visual Basic" on page 814
Set the parameters for acquiring and storing data. See "Introduction to :ACQuire Commands" on page 181.

### Table 56 :ACQuire Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:ACQuire:AALias? (see page 183)</td>
<td>{1</td>
</tr>
<tr>
<td>:ACQuire:COMPLETE &lt;complete&gt; (see page 184)</td>
<td>:ACQuire:COMPLETE? (see page 184)</td>
<td>&lt;complete&gt; ::= 100; an integer in NR1 format</td>
</tr>
<tr>
<td>:ACQuire:COUNT &lt;count&gt; (see page 185)</td>
<td>:ACQuire:COUNT? (see page 185)</td>
<td>&lt;count&gt; ::= an integer from 2 to 65536 in NR1 format</td>
</tr>
<tr>
<td>:ACQuire:DAALias &lt;mode&gt; (see page 186)</td>
<td>:ACQuire:DAALias? (see page 186)</td>
<td>&lt;mode&gt; ::= {DISable</td>
</tr>
<tr>
<td>:ACQuire:MODE &lt;mode&gt; (see page 187)</td>
<td>:ACQuire:MODE? (see page 187)</td>
<td>&lt;mode&gt; ::= {RTIMe</td>
</tr>
<tr>
<td>n/a</td>
<td>:ACQuire:POINts? (see page 188)</td>
<td>&lt;# points&gt; ::= an integer in NR1 format</td>
</tr>
<tr>
<td>:ACQuire:RSIGnal &lt;ref_signal_mode&gt; (see page 189)</td>
<td>:ACQuire:RSIGnal? (see page 189)</td>
<td>&lt;ref_signal_mode&gt; ::= {OFF</td>
</tr>
<tr>
<td>:ACQuire:SEGMented:ANALyze (see page 190)</td>
<td>n/a</td>
<td>n/a (with Option SGM)</td>
</tr>
<tr>
<td>:ACQuire:SEGMented:COUNT &lt;count&gt; (see page 191)</td>
<td>:ACQuire:SEGMented:COUNT? (see page 191)</td>
<td>&lt;count&gt; ::= an integer from 2 to 2000 (w/8M memory) in NR1 format (with Option SGM)</td>
</tr>
<tr>
<td>:ACQuire:SEGMented:IN Dex &lt;index&gt; (see page 192)</td>
<td>:ACQuire:SEGMented:IN Dex? (see page 192)</td>
<td>&lt;index&gt; ::= an integer from 2 to 2000 (w/8M memory) in NR1 format (with Option SGM)</td>
</tr>
<tr>
<td>n/a</td>
<td>:ACQuire:SRATe? (see page 195)</td>
<td>&lt;sample_rate&gt; ::= sample rate (samples/s) in NR3 format</td>
</tr>
<tr>
<td>:ACQuire:TYPE &lt;type&gt; (see page 196)</td>
<td>:ACQuire:TYPE? (see page 196)</td>
<td>&lt;type&gt; ::= {NORMAL</td>
</tr>
</tbody>
</table>

**Introduction to :ACQuire Commands**

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

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

Averaging

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

High-Resolution

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

Peak Detect

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

Real-time Mode

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

Equivalent-time Mode

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

Reporting the Setup

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

Return Format

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

:ACQ:MODE RTIM;TYPE NORM;COMP 100;COUNT 8;SEGMENT:COUN 2
:ACQuire:AALias

(see page 788)

Query Syntax

:ACQuire:AALias?

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

Return Format

<value><NL>

<value> ::= {1 | 0}

See Also

- "Introduction to :ACQuire Commands" on page 181
- ":ACQuire:DAALias" on page 186
**:ACQuire:**COMPLETE

See page 788

**Command Syntax**

```plaintext
: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**

```plaintext
: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 196
- ":DIGitize" on page 150
- ":WAVEform:POINts" on page 632

**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"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:ACQuire:COUNt**

(see page 788)

**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.

**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 196
- ":DIGitize" on page 150
- ":WAVEform:COUNt" on page 628

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

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

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

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

Return Format

<mode><NL>

See Also

- "Introduction to :ACQuire Commands" on page 181
- ":ACQuire:AALias" on page 183
:ACQuire:MODE

(see page 788)

Command Syntax
:ACQuire:MODE <mode>

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

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

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

  Real time mode is not available when averaging (:ACQuire:TYPE AVERAGE).

Query Syntax
:ACQuire:MODE?

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

Return Format
<mode_argument><NL>

<mode_argument> ::= {RTIM | ETIM | SEG}

See Also
- "Introduction to :ACQuire Commands" on page 181
- ":ACQuire:TYPE" on page 196

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

(see page 788)

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 150
- "WAVeform:POINts" on page 632
**:ACQuire:RSIGnal**

(see page 788)

**Command Syntax**

:\texttt{ACQuire:RSIGnal} <\texttt{ref\_signal\_mode}>

<\texttt{ref\_signal\_mode}> ::= \{OFF | OUT | IN\}

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

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

**CAUTION**

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

**Query Syntax**

:\texttt{ACQuire:RSIGnal}\?  

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

**Return Format**

<\texttt{ref\_signal\_mode}><NL>

<\texttt{ref\_signal\_mode}> ::= \{OFF | OUT | IN\}

**See Also**

- "\texttt{TIMebase:REFClock}" on page 461
- The *Agilent InfiniiVision 7000 Series Oscilloscope User's Guide* for information on using the 10 MHz reference clock.
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 187
- ":ACQuire:SEGメントed:COUNT" on page 191
- "Introduction to :ACQuire Commands" on page 181

NOTE
This command is available when the segmented memory option (Option SGM) is enabled.
**:ACQuire:SEGMENTed:COUNt**

(see page 788)

**Command Syntax**

:ACQuire:SEGMENTed:COUNt <count>

<count> ::= an integer from 2 to 2000 (w/8M 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 1M memory allows a maximum of 250 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 2000 (w/8M memory) in NR1 format

**See Also**

- ":ACQuire:MODE" on page 187
- "DIGitize" on page 150
- "SINGLE" on page 176
- "RUN" on page 174
- "WAVEform:SEGMENTed:COUNt" on page 639
- "ACQuire:SEGMENTed:ANALyze" on page 190
- "Introduction to :ACQuire Commands" on page 181

**Example Code**

- "Example Code" on page 192
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 1M memory allows a maximum of 250 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 2 to 2000 (w/8M memory) in NR1 format

See Also

- ":ACQuire:MODE" on page 187
- ":ACQuire:SEGmenteD:COUNt" on page 191
- ":DIGitize" on page 150
- ":SINGle" on page 176
- ":RUN" on page 174
- ":WAVeform:SEGmenteD:COUNt" on page 639
- ":WAVeform:SEGmenteD:TTAG" on page 640
- ":ACQuire:SEGmenteD:ANALyze" on page 190
- "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("TCPIP0::130.29.70.228::inst0::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 50.
    myScope.WriteString ":ACQuire:SEGMenTed:COUNt 50"
    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 = 50

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

    Dim lngSegments As Long
    lngSegments = varQueryResult

    ' For each segment:
    Dim dblTimeTag As Double
    Dim lngI As Long
For lngI = lngSegments To 1 Step -1

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

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

Next lngI

Exit Sub

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

End Sub
:ACQuire:SRATe

(see page 788)

**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 188
The :ACQuire:TYPE command selects the type of data acquisition that is to take place. The acquisition types are: NORMal, AVERage, HRESolution, and PEAK.

- The :ACQuire:TYPE NORMal command sets the oscilloscope in the normal mode.
- The :ACQuire:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQuire:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 1 to 65536. The COUNt value determines the number of averages that must be acquired.
- Setting the :ACQuire:TYPE to AVERage automatically sets :ACQuire:MODE to ETIMe (equivalent time sampling). The AVERage type is not available when in segmented memory mode (:ACQuire:MODE SEGMented).
- The :ACQuire:TYPE HRESolution command sets the oscilloscope in the high-resolution mode (also known as smoothing). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.
- For example, if the digitizer samples at 200 MSa/s, but the effective sample rate is 1 MSa/s (because of a slower sweep speed), only 1 out of every 200 samples needs to be stored. Instead of storing one sample (and throwing others away), the 200 samples are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.
- The :ACQuire:TYPE PEAK command sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

**NOTE**

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

- "Introduction to :ACQuire Commands" on page 181
- ":ACQuire:COUNt" on page 185
- ":ACQuire:MODE" on page 187
- ":DIGitize" on page 150
- ":WAVEform:TYPE" on page 646
- ":WAVEform:PREamble" on page 636

Example Code

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

Example program from the start: "VISA COM Example in Visual Basic" on page 814
## :BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See "Introduction to :BUS<n> Commands" on page 199.

### Table 57 :BUS<n> Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:BUS&lt;n&gt;:BIT&lt;m&gt; {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:BITs &lt;channel_list&gt;, {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:CLear (see page 203)</td>
<td>n/a</td>
<td>&lt;n&gt; ::= 1 or 2; an integer in NR1 format</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:LABel &lt;string&gt; (see page 205)</td>
<td>:BUS&lt;n&gt;:LABel? (see page 205)</td>
<td>&lt;string&gt; ::= quoted ASCII string up to 10 characters &lt;n&gt; ::= 1 or 2; an integer in NR1 format</td>
</tr>
<tr>
<td>:BUS&lt;n&gt;:MASK &lt;mask&gt; (see page 206)</td>
<td>:BUS&lt;n&gt;:MASK? (see page 206)</td>
<td>&lt;mask&gt; ::= 32-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; &lt;nondecimal&gt; ::= #Hnn...n where n ::= (0,..,9</td>
</tr>
</tbody>
</table>
Introduction to :BUS\(<n>\) Commands

\(<n> ::= \{1 \mid 2\}\)

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
**5 Commands by Subsystem**

### :BUS<n>:BIT<m>

(see page 788)

**Command Syntax**

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

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

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

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

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

**NOTE**

This command is only valid for the MSO models.

**Query Syntax**

:BUS<n>:BIT<m>?

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

**Return Format**

<display><NL>

<display> ::= {0 | 1}

**See Also**

- "Introduction to :BUS<n> Commands" on page 199
- ":BUS<n>:BITS" on page 201
- ":BUS<n>:CLEar" on page 203
- ":BUS<n>:DISPlay" on page 204
- ":BUS<n>:LABel" on page 205
- ":BUS<n>:MASK" on page 206

**Example Code**

' Include digital channel 1 in bus 1:
myScope.WriteString ":BUS1:BIT1 ON"
:BUS<n>:BITS

(see page 788)

Command Syntax

:BUS<n>:BITS <channel_list>, <display>

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

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

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

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

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

NOTE

This command is only valid for the MSO models.

Query Syntax

:BUS<n>:BITS?

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

Return Format

<channel_list>, <display><NL>

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

<display> ::= (0 | 1)

See Also

- "Introduction to :BUS<n> Commands" on page 199
- ":BUS<n>:BIT<m>" on page 200
- ":BUS<n>:CLEar" on page 203
- ":BUS<n>:DISPlay" on page 204
- ":BUS<n>:LABel" on page 205
- ":BUS<n>:MASK" on page 206

Example Code

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

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

' Include digital channels 1 through 15 in bus 1:
myScope.WriteString ":BUS1:BITS (@1:15), ON"
Include digital channels 1 through 5, 8, and 14 in bus 1:
myScope.WriteString "::BUS1::BITS (@1:5,8,14), ON"

:BUS<n>:CLEar

(see page 788)

**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 199
- ":BUS<n>:BIT<m>" on page 200
- ":BUS<n>:BITS" on page 201
- ":BUS<n>:DISPlay" on page 204
- ":BUS<n>:LABel" on page 205
- ":BUS<n>:MASK" on page 206
5 Commands by Subsystem

:BUS<n>:DISPLAY

N (see page 788)

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 199
- ":BUS<n>:BIT<m>" on page 200
- ":BUS<n>:BITS" on page 201
- ":BUS<n>:CLEar" on page 203
- ":BUS<n>:LABel" on page 205
- ":BUS<n>:MASK" on page 206
:BUS<n>:LABel

(see page 788)

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 199
- ":BUS<n>:BIT<m>" on page 200
- ":BUS<n>:BITS" on page 201
- ":BUS<n>:CLEar" on page 203
- ":BUS<n>:DISPlay" on page 204
- ":BUS<n>:MASK" on page 206
- ":CHANnel<n>:LABel" on page 225
- ":DISPlay:LABList" on page 250
- ":DIGital<n>:LABel" on page 240

Example Code

' Set the bus 1 label to "Data":
myScope.WriteString "::BUS1:LABel 'Data'"
5 Commands by Subsystem

**:BUS\langle n\rangle:MASK**

(see page 788)

**Command Syntax**

:\texttt{:BUS\langle n\rangle:MASK \langle mask\rangle}

\(<\text{mask}\rangle \,:=\, 32\text{-bit integer in decimal, <nondecimal>, or <string>}

\(<\text{nondecimal}\rangle \,:=\, \#\text{Hnn...n where n ::= (0,...,9 | A,...,F)} \text{ for hexadecimal}

\(<\text{nondecimal}\rangle \,:=\, \#\text{Bnn...n where n ::= (0 \mid 1)} \text{ for binary}

\(<\text{string}\rangle \,:=\, \"0xnn...n\" \text{ where n ::= (0,...,9 | A,...,F) for hexadecimal}

\(<n\rangle \,:=\, \text{An integer, 1 or 2, is attached as a suffix to BUS and defines the bus that is affected by the command.}

The :BUS\langle n\rangle: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**

:\texttt{:BUS\langle n\rangle:MASK?}

The :BUS\langle n\rangle:MASK? query returns the mask value for the specified bus.

**Return Format**

\(<\text{mask}\><\text{NL}> \text{in decimal format}

**See Also**

- "Introduction to :BUS\langle n\rangle Commands" on page 199
- ":BUS\langle n\rangle:BIT\langle m\rangle" on page 200
- ":BUS\langle n\rangle:BITS" on page 201
- ":BUS\langle n\rangle:CLEar" on page 203
- ":BUS\langle n\rangle:DISPlay" on page 204
- ":BUS\langle n\rangle:LABel" on page 205
Commands by Subsystem

CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "Introduction to :CALibrate Commands" on page 207.

Table 58 :CALibrate Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:CALibrate:DATE? (see page 209)</td>
<td>&lt;return value&gt; ::= &lt;day&gt;,&lt;month&gt;,&lt;year&gt;; all in NR1 format</td>
</tr>
<tr>
<td>:CALibrate:LABel &lt;string&gt; (see page 210)</td>
<td>:CALibrate:LABel? (see page 210)</td>
<td>&lt;string&gt; ::= quoted ASCII string up to 32 characters</td>
</tr>
<tr>
<td>:CALibrate:OUTPut &lt;signal&gt; (see page 211)</td>
<td>:CALibrate:OUTPut? (see page 211)</td>
<td>&lt;signal&gt; ::= {TRIGgers</td>
</tr>
<tr>
<td>:CALibrate:STARt (see page 212)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:STATus? (see page 213)</td>
<td>&lt;return value&gt; ::= ALL,&lt;status_code&gt;,&lt;status_string &gt; &lt;status_code&gt; ::= an integer status code &lt;status_string&gt; ::= an ASCII status string</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:SWITch? (see page 214)</td>
<td>{PROTected</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:TEMPerature? (see page 215)</td>
<td>&lt;return value&gt; ::= degrees C delta since last cal in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:CALibrate:TIME? (see page 216)</td>
<td>&lt;return value&gt; ::= &lt;hours&gt;,&lt;minutes&gt;,&lt;seconds&gt;; all in NR1 format</td>
</tr>
</tbody>
</table>

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

(see page 788)

Query Syntax :CALibrate:DATE?

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

Return Format <date><NL>

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

See Also • "Introduction to :CALibrate Commands" on page 207
## :CALibrate:LABel

(see page 788)

**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 207
:CALibrate:OUTPut

(see page 788)

Command Syntax

:CALibrate:OUTPut <signal>

<signal> ::= {TRIGgers | SOURce | DSOurce | MASK}

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

• TRIGgers – pulse when a trigger event occurs.
• SOURce – raw output of trigger comparator.
• DSOurce – SOURce frequency divided by 8.
• MASK – signal from mask test indicating a success or fail mask test.

Query Syntax

:CALibrate:OUTPut?

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

Return Format

<signal><NL>

<signal> ::= {TRIG | SOUR | DSO | MASK}

See Also

• "Introduction to :CALibrate Commands" on page 207
• ":MTESt:OUTPut" on page 377
:CALibrate:STARt

(see page 788)

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 207
- ":CALibrate:SWITch" on page 214
:**CALibrate:**STATus

(see page 788)

**Query Syntax**

**:CALibrate:**STATus?

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

**Return Format**

<return value><NL>

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

<status_code> ::= an integer status code

<status_string> ::= an ASCII status string

**See Also**

• "Introduction to **:CALibrate Commands" on page 207
**:CALibrate:SWITch**

(see page 788)

**Query Syntax**

`:CALibrate:SWITch?`

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

**Return Format**

```
<switch><NL>
```

```
<switch> ::= {PROT | UNPR}
```

**See Also**

- "Introduction to :CALibrate Commands" on page 207
:CALibrate:TEMPerature

N (see page 788)

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 207
Commands by Subsystem

:CALibrate:TIME

(see page 788)

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 207
**:CHANnel<n> Commands**

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "Introduction to :CHANnel<n> Commands" on page 218.

**Table 59 :CHANnel<n> Commands Summary**

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;n&gt;:BWLimit {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:COUPling &lt;coupling&gt; (see page 221)</td>
<td>:CHANnel&lt;n&gt;:COUPling? (see page 221)</td>
<td>&lt;coupling&gt; ::= {AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:IMPedance &lt;impedance&gt; (see page 223)</td>
<td>:CHANnel&lt;n&gt;:IMPedance? (see page 223)</td>
<td>&lt;impedance&gt; ::= {ONEMeg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:INVert {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:LABel &lt;string&gt; (see page 225)</td>
<td>:CHANnel&lt;n&gt;:LABel? (see page 225)</td>
<td>&lt;string&gt; ::= any series of 10 or less ASCII characters enclosed in quotation marks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:OFFSet &lt;offset&gt;[suffix] (see page 226)</td>
<td>:CHANnel&lt;n&gt;:OFFSet? (see page 226)</td>
<td>&lt;offset&gt; ::= Vertical offset value in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe &lt;attenuation&gt; (see page 227)</td>
<td>:CHANnel&lt;n&gt;:PROBe? (see page 227)</td>
<td>&lt;attenuation&gt; ::= Probe attenuation ratio in NR3 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PROBe:HEA D[:TYPE] &lt;head_param&gt; (see page 228)</td>
<td>:CHANnel&lt;n&gt;:PROBe:HEAD[:TYPE]? (see page 228)</td>
<td>&lt;head_param&gt; ::= {SEND0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>:CHANnel&lt;n&gt;:PROBe:ID? (see page 229)</td>
<td>&lt;probe id&gt; ::= unquoted ASCII string up to 11 characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Commands by Subsystem

### Introduction to :CHANnel<n> Commands

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

<n> ::= (1 | 2) for the two channel oscilloscope models

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

A label command provides identifying annotations of up to 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.

### Table 59 :CHANnel<n> Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :CHANnel<n>:PROBe:SKEW <skew_value> | :CHANnel<n>:PROBe:SKEW? (see page 230) | <skew_value> ::= -100 ns to +100 ns in NR3 format
| | | <n> ::= 1-2 or 1-4 in NR1 format |
| :CHANnel<n>:PROBe:STY Pe <signal type> | :CHANnel<n>:PROBe:STY Pe? (see page 231) | <signal type> ::= (DIFFerential | SINGLE)
| | | <n> ::= 1-2 or 1-4 in NR1 format |
| :CHANnel<n>:PROTectio | :CHANnel<n>:PROTectio? (see page 232) | (NORM | TRIP)
| | | <n> ::= 1-2 or 1-4 in NR1 format |
| :CHANnel<n>:RANGe <range>[suffix] | :CHANnel<n>:RANGe? (see page 233) | <range> ::= Vertical full-scale range value in NR3 format
| | | [suffix] ::= (V | mV)
| | | <n> ::= 1-2 or 1-4 in NR1 format |
| :CHANnel<n>:SCALe <scale>[suffix] | :CHANnel<n>:SCALe? (see page 234) | <scale> ::= Vertical units per division value in NR3 format
| | | [suffix] ::= (V | mV)
| | | <n> ::= 1-2 or 1-4 in NR1 format |
| :CHANnel<n>:UNITs <units> | :CHANnel<n>:UNITs? (see page 235) | <units> ::= (VOLT | AMPere)
| | | <n> ::= 1-2 or 1-4 in NR1 format |
| :CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} | :CHANnel<n>:VERNier? (see page 236) | (0 | 1)
| | | <n> ::= 1-2 or 1-4 in NR1 format |

**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;BW L 0;INV 0;LAB "1";UNIT VOLT;PROB +10E+00;PROB:SKEW +0.00E+00;STYP SING
:CHANnel<n>:BWLimit

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

<bwlimit> ::= {{1 | ON} | {0 | OFF}}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

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

Query Syntax
:CHANnel<n>:BWLimit?

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

Return Format
<bwlimit><NL>

<bwlimit> ::= {1 | 0}

See Also
- "Introduction to :CHANnel<n> Commands" on page 218
**:CHANnel\(<n>\):COUPling**

(see page 788)

**Command Syntax**

`:CHANnel\(<n>\):COUPling \(<coupling>\)`

\(<coupling>\) ::= \{AC | DC\}

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

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

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

**Query Syntax**

`:CHANnel\(<n>\):COUPling?`

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

**Return Format**

\(<coupling value>\)<NL>

\(<coupling value>\) ::= \{AC | DC\}

**See Also**

- "Introduction to :CHANnel\(<n>\) Commands" on page 218
5 Commands by Subsystem

:CHANnel<n>:DISPlay

(see page 788)

Command Syntax

:CHANnel<n>:DISPlay <display value>

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

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

<n> ::= (1 | 2) for the two channel oscilloscope models

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

Query Syntax

:CHANnel<n>:DISPlay?

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

Return Format

<display value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 218
- ":VIEW" on page 180
- ":BLANK" on page 148
- ":STATus" on page 177
- ":POD<n>:DISPlay" on page 394
- ":DIGital<n>:DISPlay" on page 239
:CHANnel<n>:IMPedance

(see page 788)

Command Syntax

:CHANnel<n>:IMPedance <impedance>

<impedance> ::= {ONEMeg | FIFTy}

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

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

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

NOTE

The analog channel input impedance of the 100 MHz bandwidth oscilloscope models is fixed at 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 | FIFT}

See Also

• "Introduction to :CHANnel<n> Commands" on page 218
::CHANnel<n>::INVert

(see page 788)

Command Syntax

::CHANnel<n>::INVert <invert value>

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

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

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

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

Query Syntax

::CHANnel<n>::INVert?

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

Return Format

<invert value><NL>

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

See Also

• "Introduction to ::CHANnel<n> Commands" on page 218
**:CHANnel<n>:LABel**

(see page 788)

**Command Syntax**

`:CHANnel<n>:LABel <string>`

- `<string>` ::= quoted ASCII string
- `<n>` ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
- `<n>` ::= {1 | 2} for the two channel oscilloscope models

**NOTE**

Label strings are 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 218
- ":DISPlay:LABel" on page 249
- ":DIGital<n>:LABel" on page 240
- ":DISPlay:LABList" on page 250
- ":BUS<n>:LABel" on page 205

**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".

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:CHANnel<n>:OFFSet

(see page 788)

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

<offset> ::= Vertical offset value in NR3 format
<suffix> ::= (V | mV)

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

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

Query Syntax
:CHANnel<n>:OFFSet?

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

Return Format
<offset><NL>

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

See Also
- "Introduction to :CHANnel<n> Commands" on page 218
- ":CHANnel<n>:RANGe" on page 233
- ":CHANnel<n>:SCALe" on page 234
- ":CHANnel<n>:PROBe" on page 227
**:CHANnel<n>:PROBe**

(see page 788)

**Command Syntax**

```
:CHANnel<n>:PROBe <attenuation>
```

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

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

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

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

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

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

**Query Syntax**

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

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

**Return Format**

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

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

**See Also**

- "Introduction to :CHANnel<n> Commands" on page 218
- ":CHANnel<n>:RANGe" on page 233
- ":CHANnel<n>:SCALe" on page 234
- ":CHANnel<n>:OFFSet" on page 226

**Example Code**

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

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:CHANnel<n>:PROBe:HEAD[:TYPE]

Command Syntax

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

NOTE

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

The :CHANnel<n>:PROBe:HEAD[:TYPE]? query returns the current probe head type setting for the selected channel.

Return Format

See Also

- "Introduction to :CHANnel<n> Commands" on page 218
- ":CHANnel<n>:PROBe" on page 227
- ":CHANnel<n>:PROBe:ID" on page 229
- ":CHANnel<n>:PROBe:SKEW" on page 230
- ":CHANnel<n>:PROBe:STYPe" on page 231
:CHANnel<n>:PROBe:ID

(see page 788)

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

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

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

Return Format
<probe id><NL>

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

Some of the possible returned values are:
- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

See Also
- "Introduction to :CHANnel<n> Commands" on page 218
:CHANnel<n>:PROBe:SKEW

Command Syntax

:CHANnel<n>:PROBe:SKEW <skew value>

<skew value> ::= skew time in NR3 format
<skew value> ::= -100 ns to +100 ns
<n> ::= {1 | 2 | 3 | 4}

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

Query Syntax

:CHANnel<n>:PROBe:SKEW?

The :CHANnel<n>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

<skew value><NL>
<skew value> ::= skew value in NR3 format

See Also
  • "Introduction to :CHANnel<n> Commands" on page 218
:CHANnel<n>:PROBe:STYPe

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

Command Syntax

:CHANnel<n>:PROBe:STYPe <signal type>

<signal type> ::= {DIFFerential | SINGle}

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

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

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

Query Syntax

:CHANnel<n>:PROBe:STYPe?

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

Return Format

<signal type><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 218
- ":CHANnel<n>:OFFSet" on page 226
:CHANnel<n>:PROTection

(see page 788)

Command Syntax

:CHANnel<n>:PROTection[:CLEar]

<n> ::= (1 | 2 | 3 | 4)

When the analog channel input impedance is set to 50Ω (on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models), the input channels are protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the channel is automatically changed to 1 MΩ. The :CHANnel<n>:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the channel to be used again in 50Ω mode after the signal that caused the overload has been removed from the channel input. Reset the analog channel input impedance to 50Ω (see ":CHANnel<n>:IMPedance" on page 223) after clearing the overvoltage protection.

Query Syntax

:CHANnel<n>:PROTection?

The :CHANnel<n>:PROTection query returns the state of the input protection for CHANnel<n>. If a channel input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

Return Format

{NORM | TRIP}<NL>

See Also

- "Introduction to :CHANnel<n> Commands" on page 218
- ":CHANnel<n>:COUPling" on page 221
- ":CHANnel<n>:IMPedance" on page 223
- ":CHANnel<n>:PROBe" on page 227
:CHANnel<n>:RANGe

Command Syntax

:CHANnel<n>:RANGe <range>[<suffix>]

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

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

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

The :CHANnel<n>:RANGe command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, legal values for the range are shown in the following table.

Table 60  Vertical Range Values with 1:1 Probe Attenuation

<table>
<thead>
<tr>
<th>Models</th>
<th>Input Impedance</th>
<th>1 MΩ</th>
<th>50 Ω</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSO/MSO701xA/B</td>
<td>16 mV to 40 V</td>
<td>16 mV to 40 V</td>
<td></td>
</tr>
<tr>
<td>DSO/MSO703xA/B</td>
<td>16 mV to 40 V</td>
<td>16 mV to 40 V</td>
<td></td>
</tr>
<tr>
<td>DSO/MSO705xA/B</td>
<td>16 mV to 40 V</td>
<td>16 mV to 40 V</td>
<td></td>
</tr>
<tr>
<td>DSO/MSO7104A/B</td>
<td>16 mV to 40 V</td>
<td>16 mV to 8 V</td>
<td></td>
</tr>
</tbody>
</table>

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 218
- ":CHANnel<n>:SCALe" on page 234
- ":CHANnel<n>:PROBe" on page 227

Example Code

' CHANNEL_RANGE - Sets the full scale vertical range in volts. The range value is 8 times the volts per division.
myScope.WriteString "::CHANNEL1:RANGE 8" ' Set the vertical range to 8 volts.

Example program from the start: "VISA COM Example in Visual Basic" on page 814
5 Commands by Subsystem

:CHANnel<n>:SCALe

N (see page 788)

Command Syntax

:CHANnel<n>:SCALe <scale>[<suffix>]

<scale> ::= vertical units per division in NR3 format

(suffix) ::= (V | mV)

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel. When using 1:1 probe attenuation, legal values for the scale are shown in the following table.

Table 61 Vertical Scale Values with 1:1 Probe Attenuation

<table>
<thead>
<tr>
<th>Models</th>
<th>Input Impedance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 MΩ</td>
</tr>
<tr>
<td>DSO/MSO701xA/B</td>
<td>2 mV to 5 V</td>
</tr>
<tr>
<td>DSO/MSO703xA/B</td>
<td>2 mV to 5 V</td>
</tr>
<tr>
<td>DSO/MSO705xA/B</td>
<td>2 mV to 5 V</td>
</tr>
<tr>
<td>DSO/MSO7104A/B</td>
<td>2 mV to 5 V</td>
</tr>
</tbody>
</table>

If the probe attenuation is changed, the scale value is multiplied by the probe's attenuation factor.

Query Syntax

:CHANnel<n>:SCALe?

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

Return Format

<scale value><NL>

<scale value> ::= vertical units per division in NR3 format

See Also

- "Introduction to :CHANnel<n> Commands" on page 218
- ":CHANnel<n>:RANGe" on page 233
- ":CHANnel<n>:PROBe" on page 227
**:CHANnel\<n>:UNITs**

(see page 788)

**Command Syntax**

```
:CHANnel\<n>:UNITs <units>
```

\n
**:units**: {VOLT | AMPere}

**:n**: {1 | 2} for the two channel oscilloscope models

**:n**: {1 | 2 | 3 | 4} for the four channel oscilloscope models

The :CHANnel\<n>:UNITs command sets the measurement units for the connected probe. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

**Query Syntax**

```
:CHANnel\<n>:UNITs?
```

The :CHANnel\<n>:UNITs? query returns the current units setting for the specified channel.

**Return Format**

```
<units><NL>
```

**:units**: {VOLT | AMP}

**See Also**

- "Introduction to :CHANnel\<n> Commands" on page 218
- ":CHANnel\<n>:RANGE" on page 233
- ":CHANnel\<n>:PROBe" on page 227
- ":EXTernal:UNITs" on page 263
:CHANnel<n>:VERNier

Command Syntax
:CHANnel<n>:VERNier <vernier value>

<vernier value> ::= {{1 | ON} | {0 | OFF}}
<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

The :CHANnel<n>:VERNier command specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).

Query Syntax
:CHANnel<n>:VERNier?

The :CHANnel<n>:VERNier? query returns the current state of the channel's vernier setting.

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

See Also
• "Introduction to :CHANnel<n> Commands" on page 218
**:DIGital<n> Commands**

Control all oscilloscope functions associated with individual digital channels. See "Introduction to :DIGital<n> Commands" on page 237.

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DIGital&lt;n&gt;:DISPlay</td>
<td>:DIGital&lt;n&gt;:DISPlay?</td>
<td>(0</td>
</tr>
<tr>
<td>{{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:DIGital&lt;n&gt;:LABel</td>
<td>:DIGital&lt;n&gt;:LABel?</td>
<td>&lt;string&gt; ::= any series of 10 or less ASCII characters enclosed in quotation marks &lt;n&gt; ::= 0-15; an integer in NR1 format</td>
</tr>
<tr>
<td>&lt;string&gt; (see page 240)</td>
<td>(see page 240)</td>
<td></td>
</tr>
<tr>
<td>:DIGital&lt;n&gt;:POSition</td>
<td>:DIGital&lt;n&gt;:POSition?</td>
<td>&lt;n&gt; ::= 0-15; an integer in NR1 format &lt;position&gt; ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small</td>
</tr>
<tr>
<td>&lt;position&gt; (see page 241)</td>
<td>(see page 241)</td>
<td></td>
</tr>
<tr>
<td>:DIGital&lt;n&gt;:SIZE</td>
<td>:DIGital&lt;n&gt;:SIZE?</td>
<td>&lt;value&gt; ::= {SMALl</td>
</tr>
<tr>
<td>&lt;value&gt; (see page 242)</td>
<td>(see page 242)</td>
<td></td>
</tr>
<tr>
<td>:DIGital&lt;n&gt;:THReshold</td>
<td>:DIGital&lt;n&gt;:THReshold?</td>
<td>&lt;n&gt; ::= 0-15; an integer in NR1 format &lt;value&gt; ::= {CMOS</td>
</tr>
<tr>
<td>&lt;value&gt;[suffix] (see page 243)</td>
<td>(see page 243)</td>
<td></td>
</tr>
</tbody>
</table>

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

<n> ::= (0,...,15)

The DIGital subsystem commands control the viewing, labeling, and positioning of digital channels. They also control threshold settings for groups of digital channels (D0-D7, D8-D15).

**NOTE**

These commands are only valid for the MSO models.

**Reporting the Setup**

Use :DIGital<n>? to query setup information for the DIGital subsystem.
Return Format

The following is a sample response from the :DIGital0? query. In this case, the query was issued following a *RST command.

:DIG0:DISP 0;THR +1.40E+00;LAB 'D0';POS +0
:DIGital<n>:DISPlay

Command Syntax

:DIGital<n>:DISPlay <display>

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

<n> ::= An integer, 0, 1,...,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

The :DIGital<n>:DISPlay command turns digital display on or off for the specified channel.

NOTE

This command is only valid for the MSO models.

Query Syntax

:DIGital<n>:DISPlay?

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

Return Format

<display><NL>

<display> ::= {0 | 1}

See Also

- "Introduction to :DIGital<n> Commands" on page 237
- ":POD<n>:DISPlay" on page 394
- ":CHANnel<n>:DISPlay" on page 222
- ":VIEW" on page 180
- ":BLANk" on page 148
- ":STATus" on page 177
**Command Syntax**

:DIGital<n>:LABel <string>

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

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

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

**NOTE**

This command is only valid for the MSO models.

**NOTE**

Label strings are 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<n>:LABel?

The :DIGital<n>: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<n> Commands" on page 237
- ":CHANnel<n>:LABel" on page 225
- ":DISPlay:LABList" on page 250
- ":BUS<n>:LABel" on page 205
:DIGital<n>:POSition

(see page 788)

Command Syntax

:DIGital<n>:POSition <position>

<position> ::= integer in NR1 format.

<n> ::= An integer, 0, 1,\ldots,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

<table>
<thead>
<tr>
<th>Channel Size</th>
<th>Position</th>
<th>Top</th>
<th>Bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>0-7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Medium</td>
<td>0-15</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Small</td>
<td>0-31</td>
<td>31</td>
<td>0</td>
</tr>
</tbody>
</table>

The :DIGital<n>:POSition command sets the position of the specified channel.

NOTE

This command is only valid for the MSO models.

Query Syntax

:DIGital<n>:POSition?

The :DIGital<n>:POSition? query returns the position of the specified channel.

Return Format

<position><NL>

<position> ::= integer in NR1 format.

See Also

- "Introduction to :DIGital<n> Commands" on page 237
5  Commands by Subsystem

:DIgital<n>:SIZE

(see page 788)

Command Syntax

:DIgital<n>:SIZE <value>

<n> ::= An integer, 0, 1,...,15, is attached as a suffix to the command
       and defines the logic channel that is affected by the command.

<value> ::= {SMALl | MEDium | LARGe}

The :DIgital<n>:SIZE command specifies the size of digital channels on the
display. Sizes are set for all digital channels. Therefore, if you set the size
on digital channel 0 (for example), the same size is set on channels 1
through 15 as well.

NOTE

This command is only valid for the MSO models.

Query Syntax

:DIgital<n>:SIZE?

The :DIgital<n>:SIZE? query returns the size setting for the specified
digital channels.

Return Format

<size_value><NL>

<size_value> ::= {SMAL | MED | LARG}

See Also

• "Introduction to :DIgital<n> Commands" on page 237
• ":POD<n>:SIZE" on page 395
• ":DIgital<n>:POSition" on page 241
**DIGital<n>:THReshold**

(see page 788)

**Command Syntax**

```plaintext
:DIGital<n>:THReshold <value>
```

<value> ::= (CMOS | ECL | TTL | <user defined value>[:<suffix>])

<user defined value> ::= -8.00 to +8.00 in NR3 format

<suffix> ::= {V | mV | µV}

<n> ::= An integer, 0, 1,..,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

- TTL = 1.4V
- CMOS = 2.5V
- ECL = -1.3V

The :DIGital<n>:THReshold command sets the logic threshold value for all channels grouped with the specified channel (D0-D7, D8-D15). The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

**NOTE**

This command is only valid for the MSO models.

**Query Syntax**

:DIGital<n>:THReshold?

The :DIGital<n>:THReshold? query returns the threshold value for the specified channel.

**Return Format**

<value><NL>

<value> ::= threshold value in NR3 format

**See Also**

- "Introduction to :DIGital<n> Commands" on page 237
- ":POD<n>:THReshold" on page 396
- ":TRIGger[:EDGE]:LEVel" on page 504
Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 244.

### Table 63: :DISPlay Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DISPlay:CLEar (see page 246)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:DISPlay:DATA [&lt;format&gt;][,][&lt;area&gt;][,][&lt;palette&gt;]&lt;display data&gt; (see page 247)</td>
<td>:DISPlay:DATA? [&lt;format&gt;][,][&lt;area&gt;][,][&lt;palette&gt;] (see page 247)</td>
<td>&lt;format&gt; ::= {TIFF} (command) &lt;area&gt; ::= {GRATicule} (command) &lt;palette&gt; ::= {MONochrome} (command) &lt;format&gt; ::= {TIFF</td>
</tr>
<tr>
<td>:DISPlay:LABel {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:DISPlay:LABList &lt;binary block&gt; (see page 250)</td>
<td>:DISPlay:LABList? (see page 250)</td>
<td>&lt;binary block&gt; ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters</td>
</tr>
<tr>
<td>:DISPlay:PERSistence &lt;value&gt; (see page 251)</td>
<td>:DISPlay:PERSistence? (see page 251)</td>
<td>&lt;value&gt; ::= {MINimum</td>
</tr>
<tr>
<td>:DISPlay:SOURce &lt;value&gt; (see page 252)</td>
<td>:DISPlay:SOURce? (see page 252)</td>
<td>&lt;value&gt; ::= {PMEMory{0</td>
</tr>
<tr>
<td>:DISPlay:VECTors {{1</td>
<td>ON}</td>
<td>{0</td>
</tr>
</tbody>
</table>

### Introduction to :DISPlay Commands

The DISPlay subsystem is used to control the display storage and retrieval of waveform data, labels, and text. This subsystem allows the following actions:

- Clear the waveform area on the display.
- Turn vectors on or off.
• Set waveform persistence.
• Specify labels.
• Save and Recall display data.

Reporting the Setup

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

Return Format

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

:DISP:LAB 0;CONN 1;PERS MIN;SOUR PMEM9
5 Commands by Subsystem

:DISPlay:CLEar

(see page 788)

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 244
- ":CDISplay" on page 149
**:DISPlay:DATA**

(see page 788)

**Command Syntax**

`:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data>

<format> ::= {TIFF}

(area) ::= {GRATicule}

<palette> ::= {MONochrome}

<display data> ::= binary block data in IEEE-488.2 # format.

The :DISPlay:DATA command writes trace memory data (a display bitmap) to the display or to one of the trace memories in the instrument.

If a data format or area is specified, the :DISPlay:DATA command transfers the data directly to the display. If neither the data format nor the area is specified, the command transfers data to the trace memory specified by the :DISPlay:SOURce command. Available trace memories are PMEMory0-9 and these memories correspond to the INTERN_0-9 files in the front panel Save/Recall menu.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. This is the same data saved using the front panel Save/Recall menu or the *SAV (Save) command.

**Query Syntax**

`:DISPlay:DATA? [<format>][,] [<area>][,] [<palette>]`

<format> ::= {TIFF | BMP | BMP8bit | PNG}

(area) ::= {GRATicule | SCReen}

<palette> ::= {MONochrome | GRAYscale | COlor}

The :DISPlay:DATA? query reads display data from the screen or from one of the trace memories in the instrument. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

If a data format or area is specified, the :DISPlay:DATA query transfers the data directly from the display. If neither the data format nor the area is specified, the query transfers data from the trace memory specified by the :DISPlay:SOURce command.

Screen data is the full display and is high resolution in grayscale or color. The :HARDcopy:INKSaver setting also affects the screen data. It may be read from the instrument in 24-bit bmp, 8-bit bmp, or 24-bit png format. This data cannot be sent back to the instrument.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. You can get this data and send it back to the oscilloscope.
If the format is TIFF, the only valid value area parameter is GRATicule, and the only valid palette parameter is MONOchrome.

If the format is something other than TIFF, the only valid area parameter is SCReen, and the only valid values for palette are GRAYscale or COlor.

**Return Format**

<display data><NL>
<display data> ::= binary block data in IEEE-488.2 # format.

**See Also**

- "Introduction to :DISPlay Commands" on page 244
- ":DISPlay:SOURce" on page 252
- ":HARDcopy:INKSaver" on page 287
- ":MERGe" on page 158
- ":PRINT" on page 173
- "*RCL (Recall)" on page 127
- "*SAV (Save)" on page 131
- ":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, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
  Kill strPath
End If
Close #1 ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1 ' Open file f or output.
Put #1, , byteData ' Write data.
Close #1 ' Close file.
myScope.IO.Timeout = 5000

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:DISPlay:LABel

(see page 788)

**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 244
- ":CHANnel<n>:LABel" on page 225

**Example Code**

' DISP_LABEL (not executed in this example)
' - Turns label names ON or OFF on the analyzer display.
myScope.WriteString "':DISPLAY:LABEL ON'" ' Turn on labels.

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:DISPlay:LABList**

(see page 788)

**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 244
- ":DISPlay:LABel" on page 249
- ":CHANnel<n>:LABel" on page 225
- ":DIGital<n>:LABel" on page 240
- ":BUS<n>:LABel" on page 205
**:DISPlay:PERSistence**

**Command Syntax**: :DISPlay:PERSistence <value>

<value> ::= (MINimum | INFinite)

The :DISPlay:PERSistence command specifies the persistence setting. MINimum indicates zero persistence and INFinite indicates infinite persistence. Use the :DISPlay:CLEar or :CDISplay root command to erase points stored by infinite persistence.

**Query Syntax**: :DISPlay:PERSistence?

The :DISPlay:PERSistence? query returns the specified persistence value.

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

<value> ::= (MIN | INF)

**See Also**
- "Introduction to :DISPlay Commands" on page 244
- ":DISPlay:CLEar" on page 246
- ":CDISplay" on page 149
**:DISPlay:SOURce**

(see page 788)

**Command Syntax**

`:DISPlay:SOURce <value>`

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

PMEMory0-9 ::= pixel memory 0 through 9

The :DISPlay:SOURce command specifies the default source and destination for the :DISPlay:DATA command and query. PMEMory0-9 correspond to the INTERN_0-9 files found in the front panel Save/Recall menu.

**Query Syntax**

`:DISPlay:SOURce?`

The :DISPlay:SOURce? query returns the specified SOURce.

**Return Format**

<value><NL>

<value> ::= {PMEM0 | PMEM1 | PMEM2 | PMEM3 | PMEM4 | PMEM5 | PMEM6 | PMEM7 | PMEM8 | PMEM9}

**See Also**

- "Introduction to :DISPlay Commands" on page 244
- ":DISPlay:DATA" on page 247
**:DISPlay:VECTors**

(see page 788)

**Command Syntax**

```plaintext
:DISPlay:VECTors <vectors>
```

```plaintext
<vectors> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:VECTors command turns vector display on or off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

**Query Syntax**

```plaintext
:DISPlay:VECTors?
```

The :DISPlay:VECTors? query returns whether vector display is on or off.

**Return Format**

```plaintext
<vectors><NL>
<vectors> ::= {1 | 0}
```

**See Also**

- "Introduction to :DISPlay Commands" on page 244
## :EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXTernal Trigger Commands" on page 254.

### Table 64 :EXTernal Trigger Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:EXTernal:BWLimit &lt;bwlimit&gt; (see page 256)</td>
<td>:EXTernal:BWLimit? (see page 256)</td>
<td>&lt;bwlimit&gt; ::= {0</td>
</tr>
<tr>
<td>:EXTernal:IMPedance &lt;value&gt; (see page 257)</td>
<td>:EXTernal:IMPedance? (see page 257)</td>
<td>&lt;impedance&gt; ::= {ONEMeg</td>
</tr>
<tr>
<td>:EXTernal:PROBe &lt;attenuation&gt; (see page 258)</td>
<td>:EXTernal:PROBe? (see page 258)</td>
<td>&lt;attenuation&gt; ::= probe attenuation ratio in NR3 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:EXTernal:PROBe:ID? (see page 259)</td>
<td>&lt;probe id&gt; ::= unquoted ASCII string up to 11 characters</td>
</tr>
<tr>
<td>:EXTernal:PROBe:STYPe &lt;signal type&gt; (see page 260)</td>
<td>:EXTernal:PROBe:STYPe? (see page 260)</td>
<td>&lt;signal type&gt; ::= {DIFFerential</td>
</tr>
<tr>
<td>:EXTernal:PROTection[ :CLEar] (see page 261)</td>
<td>:EXTernal:PROTection? (see page 261)</td>
<td>{NORM</td>
</tr>
<tr>
<td>:EXTernal:RANGe &lt;range&gt;[&lt;suffix&gt;] (see page 262)</td>
<td>:EXTernal:RANGe? (see page 262)</td>
<td>&lt;range&gt; ::= vertical full-scale range value in NR3 format &lt;suffix&gt; ::= {V</td>
</tr>
<tr>
<td>:EXTernal:UNITs &lt;units&gt; (see page 263)</td>
<td>:EXTernal:UNITs? (see page 263)</td>
<td>&lt;units&gt; ::= {VOLT</td>
</tr>
</tbody>
</table>

### Introduction to :EXTernal Trigger Commands

The EXTernal trigger subsystem commands control the input characteristics of the external trigger input. The probe factor, impedance, input range, input protection state, units, and bandwidth limit settings may all be queried. Depending on the instrument type, some settings may be changeable.

#### Reporting the Setup

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

#### Return Format
The following is a sample response from the :EXTernal query. In this case, the query was issued following a *RST command.

:EXT:BWL 0;IMP ONEM;RANG +8.0E+00;UNIT VOLT;PROB +1.0E+00;PROB:STYP SING
5 Commands by Subsystem

:EXTernal:BWLimit

(see page 788)

**Command Syntax**

:EXTernal:BWLimit <bwlimit>

<bwlimit> ::= {0 | OFF}

The :EXTernal:BWLimit command is provided for product compatibility. The only legal value is 0 or OFF. Use the :TRIGger:HFReject command to limit bandwidth on the external trigger input.

**Query Syntax**

:EXTernal:BWLimit?

The :EXTernal:BWLimit? query returns the current setting of the low-pass filter (always 0).

**Return Format**

<bwlimit><NL>

<bwlimit> ::= 0

**See Also**

- "Introduction to :EXTernal Trigger Commands" on page 254
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:HFReject" on page 472
:EXTernal:IMPedance

(see page 788)

**Command Syntax**

:EXTernal:IMPedance <value>

<value> ::= {ONEMeg | FIFTy}

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

**NOTE**

You can set external trigger input impedance to FIFTy (50Ω) on the 2-channel, 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models.

**Query Syntax**

:EXTernal:IMPedance?

The :EXTernal:IMPedance? query returns the current input impedance setting for the external trigger.

**Return Format**

<impedance value><NL>

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

**See Also**

- "Introduction to :EXTernal Trigger Commands" on page 254
- "Introduction to :TRIGger Commands" on page 468
- ":CHANnel<n>:IMPedance" on page 223
**:EXTernal:PROBe**

(see page 788)

**Command Syntax**

`:EXTernal:PROBe <attenuation>`

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

The :EXTernal:PROBe command specifies the probe attenuation factor for the external trigger. The probe attenuation factor may be 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors and for setting trigger levels.

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

**Query Syntax**

`:EXTernal:PROBe?`

The :EXTernal:PROBe? query returns the current probe attenuation factor for the external trigger.

**Return Format**

`<attenuation><NL>`

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

**See Also**

- "Introduction to :EXTernal Trigger Commands" on page 254
- ":EXTernal:RANGe" on page 262
- "Introduction to :TRIGger Commands" on page 468
- ":CHANnel<n>:PROBe" on page 227
:EXTernAl:PROBe:ID

(see page 788)

Query Syntax

:EXTernAl:PROBe:ID?

The :EXTernAl:PROBe:ID? query returns the type of probe attached to the external trigger input.

Return Format

<probe id><NL>

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

Some of the possible returned values are:

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

See Also

"Introduction to :EXTernAl Trigger Commands" on page 254
:EXTernal:PROBe:STYPe

(see page 788)

Command Syntax

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

NOTE

The :EXTernal:PROBe:STYPe command sets the external trigger probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

Query Syntax

:EXTernal:PROBe:STYPe?

The :EXTernal:PROBe:STYPe? query returns the current probe signal type setting for the external trigger.

Return Format

<signal type><NL>

<signal type> ::= (DIFFerential | SINGle)

See Also

• "Introduction to :EXTernal Trigger Commands" on page 254
**:EXTernal:PROTection**

(see page 788)

**Command Syntax**

`:EXTernal:PROTection[:CLEar]`

When the external trigger input impedance is set to 50Ω (on the 2-channel, 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models), the external trigger input is protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the external trigger is automatically changed to 1 MΩ. The :EXTernal:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the external trigger to be used again in 50Ω mode after the signal that caused the overload has been removed from the external trigger input. Reset the external trigger input impedance to 50Ω (see ":EXTernal:IMPedance" on page 257) after clearing the overvoltage protection.

**Query Syntax**

`:EXTernal:PROTection?`

The :EXTernal:PROTection query returns the state of the input protection for external trigger. If the external trigger input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

**Return Format**

\{NORM | TRIP\}<NL>

**See Also**

- "Introduction to :EXTernal Trigger Commands" on page 254
- ":EXTernal:IMPedance" on page 257
- ":EXTernal:PROBe" on page 258
**:EXTernal:RANGE**

(see page 788)

**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:

- In 2-channel models, the range can be set to 1.0 V or 8.0 V.
- In 4-channel models, the range can only be set to 5.0 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 254
- ":EXTernal:PROBe" on page 258
- "Introduction to :TRIGger Commands" on page 468
**EXTernal:UNITs**

(see page 788)

**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 254
- "Introduction to :TRIGger Commands" on page 468
- ":EXTeRnal:RANGE" on page 262
- ":EXTeRnal:PROBe" on page 258
- ":CHANnel<n>:UNITs" on page 235
5  Commands by Subsystem

:FUNCtion Commands

Control functions in the measurement/storage module. See "Introduction to :Functio
n Commands" on page 266.

Table 65  :FUNCtion Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:FUNCTION:CENTer</td>
<td>:FUNCTION:CENTer?</td>
<td>&lt;frequency&gt; ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.</td>
</tr>
<tr>
<td>&lt;frequency&gt; (see page 267)</td>
<td>(see page 267)</td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:DISPlay</td>
<td>:FUNCTION:DISPlay?</td>
<td>{0</td>
</tr>
<tr>
<td>{{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:FUNCTION:GOFT:OPERat</td>
<td>:FUNCTION:GOFT:OPERat?</td>
<td>&lt;operation&gt; ::= {ADD</td>
</tr>
<tr>
<td>ion &lt;operation&gt; (see page 269)</td>
<td>(see page 269)</td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:GOFT:SOURce</td>
<td>:FUNCTION:GOFT:SOURce</td>
<td>&lt;source&gt; ::= CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>1 &lt;source&gt; (see page 270)</td>
<td>1? (see page 270)</td>
<td>&lt;n&gt; ::= {1</td>
</tr>
<tr>
<td>:FUNCTION:GOFT:SOURce</td>
<td>:FUNCTION:GOFT:SOURce</td>
<td>&lt;source&gt; ::= CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>2 &lt;source&gt; (see page 271)</td>
<td>2? (see page 271)</td>
<td>&lt;n&gt; ::= {(1</td>
</tr>
<tr>
<td>:FUNCTION:OFFSet</td>
<td>:FUNCTION:OFFSet?</td>
<td>&lt;offset&gt; ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.</td>
</tr>
<tr>
<td>&lt;offset&gt; (see page 272)</td>
<td>(see page 272)</td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:OPERation</td>
<td>:FUNCTION:OPERation?</td>
<td>&lt;operation&gt; ::= {ADD</td>
</tr>
<tr>
<td>&lt;operation&gt; (see page 273)</td>
<td>(see page 273)</td>
<td></td>
</tr>
</tbody>
</table>
### Table 65 :FUNCTION Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:FUNCTION:RANGE</td>
<td>:FUNCTION:RANGE?</td>
<td>&lt;range&gt; ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTEgrate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.</td>
</tr>
<tr>
<td>&lt;range&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:REFERence</td>
<td>:FUNCTION:REFERence?</td>
<td>&lt;level&gt; ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.</td>
</tr>
<tr>
<td>&lt;level&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:SCALE</td>
<td>:FUNCTION:SCALE?</td>
<td>&lt;scale value&gt; ::= integer in NR1 format &lt;suffix&gt; ::= {V</td>
</tr>
<tr>
<td>&lt;scale value&gt;[&lt;suffix&gt;]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:SOURcelp</td>
<td>:FUNCTION:SOURcel?</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>&lt;source&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:SOURcelp2</td>
<td>:FUNCTION:SOURcel2?</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>&lt;source&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:SPAN</td>
<td>:FUNCTION:SPAN?</td>
<td>&lt;span&gt; ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.</td>
</tr>
<tr>
<td>&lt;span&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:FUNCTION:WINDow</td>
<td>:FUNCTION:WINDow?</td>
<td>&lt;window&gt; ::= {RECTangular</td>
</tr>
<tr>
<td>&lt;window&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Introduction to :FUNCtion

The FUNCtion subsystem controls the math functions in the oscilloscope. Add, subtract, multiply, differentiate, integrate, square root, and FFT (Fast Fourier Transform) operations are available. These math operations only use the analog (vertical) channels.

The SOURce1, DISPlay, RANGe, and OFFSet commands apply to any function. The SPAN, CENTer, and WINDow commands are only useful for FFT functions. When FFT is selected, the cursors change from volts and time to decibels (dB) and frequency (Hz).

Reporting the Setup

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

Return Format

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

:FUNC:OPER ADD;DISP 0;SOUR1 CHAN1;SOUR2 CHAN2;RANG +8.00E+00;OFFS +0.0E+00;:FUNC:GOFT:OPER ADD;SOUR1 CHAN1;SOUR2 CHAN2
**:FUNCtion:CENTer**

(see page 788)

**Command Syntax**

:FUNCtion:CENTer <frequency>

<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.

The :FUNCtion:CENTer command sets the center frequency when FFT (Fast Fourier Transform) is selected.

**Query Syntax**

:FUNCtion:CENTer?

The :FUNCtion:CENTer? query returns the current center frequency in Hertz.

**Return Format**

<frequency><NL>

<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.

**NOTE**

After an *RST (Reset) or :AUToscale command, the values returned by the :FUNCtion:CENTer? and :FUNCtion:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FUNCtion:CENTer or :FUNCtion:SPAN value, they no longer track the :TIMebase:RANGe value.

**See Also**

- "Introduction to :FUNCtion Commands" on page 266
- ":FUNCtion:SPAN" on page 279
- ":TIMebase:RANGe" on page 460
- ":TIMebase:SCALe" on page 463
**:FUNCTION:DISPLAY**

(see page 788)

**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 266
- ":VIEW" on page 180
- ":BLANK" on page 148
- ":STATUS" on page 177
:FUNCTION:GOFT:OPERation

(see page 788)

Command Syntax

:FUNCTION:GOFT:OPERation <operation>

<operation> ::= (ADD | SUBTract | MULTiply)

The :FUNCTION:GOFT:OPERation command sets the math operation for the g(t) source that can be used as the input to the FFT, INTEGRate, DIFFerentiate, or SQRT functions:

- ADD — Source1 + source2.
- SUBTract — Source1 - source2.
- MULTiply — Source1 * source2.

The :FUNCTION:GOFT:SOURce1 and :FUNCTION:GOFT:SOURce2 commands are used to select source1 and source2.

Query Syntax

:FUNCTION:GOFT:OPERation?

The :FUNCTION:GOFT:OPERation? query returns the current g(t) source operation setting.

Return Format

<operation><NL>

<operation> ::= (ADD | SUBT | MULT)

See Also

- "Introduction to :FUNCTION Commands" on page 266
- "FUNCTION:GOFT:SOURce1" on page 270
- "FUNCTION:GOFT:SOURce2" on page 271
- "FUNCTION:SOURce1" on page 277
5  Commands by Subsystem

:FUNCtion:GOFT:SOURce1

(see page 788)

Command Syntax

:FUNCtion:GOFT:SOURce1 <value>

<value> ::= CHANnel<n>

<n> ::= (1 | 2 | 3 | 4) for 4ch models

<n> ::= (1 | 2) for 2ch models

The :FUNCtion:GOFT:SOURce1 command selects the first input channel for
the g(t) source that can be used as the input to the FFT, INTegrate,
DIFFerentiate, or SQRT functions.

Query Syntax

:FUNCtion:GOFT:SOURce1?

The :FUNCtion:GOFT:SOURce1? query returns the current selection for the
first input channel for the g(t) source.

Return Format

<value><NL>

<value> ::= CHAN<n>

<n> ::= (1 | 2 | 3 | 4) for the 4ch models

<n> ::= (1 | 2) for the 2ch models

See Also

- "Introduction to :FUNCtion Commands" on page 266
- ":FUNCtion:GOFT:SOURce2" on page 271
- ":FUNCtion:GOFT:OPERation" on page 269
:FUNCtion:GOFT:SOURce2

(see page 788)

**Command Syntax**

:FUNCtion:GOFT:SOURce2 <value>

<value> ::= CHANnel<n>

<n> ::= {(1 | 2) | (3 | 4)} for 4ch models, depending on SOURce1 selection

<n> ::= {1 | 2} for 2ch models

The :FUNCtion:GOFT:SOURce2 command selects the second input channel for the g(t) source that can be used as the input to the FFT, INTe grate, DIFFerentiate, or SQRT functions.

If CHANnel1 or CHANnel2 is selected for :FUNCtion:GOFT:SOURce1, the SOURce2 selection can be CHANnel1 or CHANnel2. Likewise, if CHANnel3 or CHANnel4 is selected for :FUNCtion:GOFT:SOURce1, the SOURce2 selection can be CHANnel3 or CHANnel4.

**Query Syntax**

:FUNCtion:GOFT:SOURce2?

The :FUNCtion:GOFT:SOURce2? query returns the current selection for the second input channel for the g(t) source.

**Return Format**

<value><NL>

<value> ::= CHAN<n>

<n> ::= {(1 | 2) | (3 | 4)} for 4ch models, depending on SOURce1 selection

<n> ::= {1 | 2} for 2ch models

**See Also**

- "Introduction to :FUNCtion Commands" on page 266
- ":FUNCtion:GOFT:SOURce1" on page 270
- ":FUNCtion:GOFT:OPERation" on page 269
**Commands by Subsystem**

---

**:FUNCtion:OFFSet**

(see page 788)

**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 266
- ":FUNCtion:RANGe" on page 274
- ":FUNCtion:REfer" on page 275
- ":FUNCtion:SCALe" on page 276
**:FUNCTION:OPERation**

(see page 788)

**Command Syntax**

:FUNCTION:OPERation <operation>

<operation> ::= (ADD | SUBTract | MULTiply | INTegrate | DIFFerentiate | FFT | SQRT)

The :FUNCTION:OPERation command sets the desired waveform math operation:

- **ADD** — Source1 + source2.
- **SUBTract** — Source1 - source2.
- **MULTiply** — Source1 * source2.
- **INTegrate** — Integrate the selected waveform source.
- **DIFFerentiate** — Differentiate the selected waveform source.
- **FFT** — Fast Fourier Transform on the selected waveform source.
- **SQRT** — Square root on the selected waveform source.

When the operation is ADD, SUBTract, or MULTiply, the :FUNCTION:SOURce1 and :FUNCTION:SOURce2 commands are used to select source1 and source2. For all other operations, the :FUNCTION:SOURce1 command selects the waveform source.

**Query Syntax**

:FUNCTION:OPERation?

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

**Return Format**

<operation><NL>

<operation> ::= (ADD | SUBT | MULT | INT | DIFF | FFT | SQRT)

**See Also**

- "Introduction to :FUNCTION Commands" on page 266
- ":FUNCTION:SOURce1" on page 277
- ":FUNCTION:SOURce2" on page 278
5 Commands by Subsystem

**:FUNCtion:RANGe**

N (see page 788)

**Command Syntax**

**:FUNCtion:RANGe** <range>

<range> ::= the full-scale vertical axis value in NR3 format.

The :FUNCtion:RANGe command defines the full-scale vertical axis for the selected function.

**Query Syntax**

**:FUNCtion:RANGe?**

The :FUNCtion:RANGe? query returns the current full-scale range value for the selected function.

**Return Format**

<range><NL>

<range> ::= the full-scale vertical axis value in NR3 format.

The range for ADD, SUBT, MULT is 8E-6 to 800E+3.

The range for the INTEGRATE function is 8E-9 to 400E+3 (depends on sweep speed).

The range for the DIFFERENTIATE function is 80E-3 to 8.0E12 (depends on sweep speed).

The range for the FFT (Fast Fourier Transform) function is 8 to 800 dBV.

**See Also**

- "Introduction to :FUNCtion Commands" on page 266
- ":FUNCtion:SCALe" on page 276
:FUNCTION:REFERENCE

(see page 788)

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 266
- ":FUNCTION:OFFSET" on page 272
- ":FUNCTION:RANGE" on page 274
- ":FUNCTION:SCALE" on page 276
Commands by Subsystem

:FUNCtion:SCALe

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 266
- ":FUNCtion:RANGe" on page 274
\textbf{:FUNCTION:SOURCE1}

\textbf{Command Syntax}

\texttt{:FUNCTION:SOURCE1 <value>}

\texttt{<value> ::= (CHANnel<n> \mid \text{GOFT})}

\texttt{<n> ::= (1 \mid 2 \mid 3 \mid 4) for 4ch models}

\texttt{<n> ::= (1 \mid 2) for 2ch models}

The :FUNCTION:SOURCE1 command is used for any :FUNCTION:OPERation selection (including the ADD, SUBtract, or MULTiply channel math operations and the FFT, INTegrate, DIFFerentiate, or SQRT transforms). This command selects the first source for channel math operations or the single source for the transforms.

The \texttt{GOFT} parameter is only available for the FFT, INTegrate, DIFFerentiate, or SQRT functions. It lets you specify, as the function input source, the addition, subtraction, or multiplication of two channels. When \texttt{GOFT} is used, the \texttt{g(t)} source is specified by the :FUNCTION:GOFT:OPERation, :FUNCTION:GOFT:SOURCE1, and :FUNCTION:GOFT:SOURCE2 commands.

\textbf{NOTE}

Another shorthand notation for \texttt{SOURCE1} in this command/query (besides \texttt{SOUR}) is \texttt{SOUR}.

\textbf{Query Syntax}

\texttt{:FUNCTION:SOURCE1?}

The :FUNCTION:SOURCE1? query returns the current source1 for function operations.

\textbf{Return Format}

\texttt{<value><NL>}

\texttt{<value> ::= (CHAN<n> \mid \text{GOFT})}

\texttt{<n> ::= (1 \mid 2 \mid 3 \mid 4) for 4ch models}

\texttt{<n> ::= (1 \mid 2) for 2ch models}

\textbf{See Also}

- "Introduction to :FUNCTION Commands" on page 266
- "FUNCTION:OPERation" on page 273
- "FUNCTION:GOFT:OPERation" on page 269
- "FUNCTION:GOFT:SOURCE1" on page 270
- "FUNCTION:GOFT:SOURCE2" on page 271
5 Commands by Subsystem

:FUNCtion:SOURce2

N (see page 788)

Command Syntax

:FUNCtion:SOURce2 <value>

<value> ::= (CHANnel<n> | NONE)

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection

<n> ::= {1 | 2} for 2ch models

The :FUNCtion:SOURce2 command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the :FUNCtion:OPERation command for more information about selecting an operation). The :FUNCtion:SOURce2 command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCtion:OPERation command.

If CHANnel1 or CHANnel2 is selected for :FUNCtion:SOURce1, the SOURce2 selection can be CHANnel1 or CHANnel2. Likewise, if CHANnel3 or CHANnel4 is selected for :FUNCtion:SOURce1, the SOURce2 selection can be CHANnel3 or CHANnel4.

Query Syntax

:FUNCtion:SOURce2?

The :FUNCtion:SOURce2? query returns the second source for function operations on two waveforms.

Return Format

<value><NL>

<value> ::= (CHAN<n> | NONE)

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection

<n> ::= {1 | 2} for 2ch models

See Also

- "Introduction to :FUNCtion Commands" on page 266
- ":FUNCtion:OPERation" on page 273
**:FUnCtion**:SPAN

(see page 788)

**Command Syntax**

**:FUnCtion**:SPAN <span>

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

If you set the frequency span to a value outside of the legal range, the step size is automatically set to the nearest legal value.

The :FUnCtion:SPAN command sets the frequency span of the display (left graticule to right graticule) when FFT (Fast Fourier Transform) is selected.

**Query Syntax**

**:FUnCtion**:SPAN?

The :FUnCtion:SPAN? query returns the current frequency span in Hertz.

**NOTE**

After a "RST (Reset) or :AUToscale command, the values returned by the :FUnCtion:CENTer? and :FUnCtion:SPAN? queries depend on the current :TImebase:RANGe value. Once you change either the :FUnCtion:CENTer or :FUnCtion:SPAN value, they no longer track the :TImebase:RANGe value.

**Return Format**

<span><NL>

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

**See Also**

- "Introduction to :FUnCtion Commands" on page 266
- ":FUnCtion:CENTer" on page 267
- ":TImebase:RANGe" on page 460
- ":TImebase:SCALe" on page 463
**Commands by Subsystem**

---

**:FUNCtion:WINDow**

(see page 788)

**Command Syntax**

```
:FUNCtion:WINDow <window>
```

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

The :FUNCtion:WINDow command allows the selection of four different windowing transforms or operations for the FFT (Fast Fourier Transform) function.

The FFT operation assumes that the time record repeats. Unless an integral number of sampled waveform cycles exist in the record, a discontinuity is created between the end of one record and the beginning of the next. This discontinuity introduces additional frequency components about the peaks into the spectrum. This is referred to as leakage. To minimize leakage, windows that approach zero smoothly at the start and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input signals.

- RECTangular — useful for transient signals, and signals where there are an integral number of cycles in the time record.
- HANNing — useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements. This is the default window.
- FLATtop — best for making accurate amplitude measurements of frequency peaks.
- BHARris (Blackman-Harris) — reduces time resolution compared to the rectangular window, but it improves the capacity to detect smaller impulses due to lower secondary lobes (provides minimal spectral leakage).

**Query Syntax**

```
:FUNCtion:WINDow?
```

The :FUNCtion:WINDow? query returns the value of the window selected for the FFT function.

**Return Format**

```
<window><NL>
```

`<window>` ::= (RECT | HANN | FLAT | BHAR)

**See Also**

- "Introduction to :FUNCtion Commands" on page 266
:HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "Introduction to :HARDcopy Commands" on page 282.

Table 66 :HARDcopy Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:HARDcopy:AREA &lt;area&gt; (see page 283)</td>
<td>:HARDcopy:AREA? (see page 283)</td>
<td>&lt;area&gt; ::= SCReen</td>
</tr>
<tr>
<td>:HARDcopy:APRinter &lt;active_printer&gt; (see page 284)</td>
<td>:HARDcopy:APRinter? (see page 284)</td>
<td>&lt;active_printer&gt; ::= {&lt;index&gt;</td>
</tr>
<tr>
<td>:HARDcopy:FACTors {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:HARDcopy:FFEed {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:HARDcopy:INKSaver {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:HARDcopy:LAYout &lt;layout&gt; (see page 288)</td>
<td>:HARDcopy:LAYout? (see page 288)</td>
<td>&lt;layout&gt; ::= {LANDscape</td>
</tr>
<tr>
<td>:HARDcopy:PALette &lt;palette&gt; (see page 289)</td>
<td>:HARDcopy:PALette? (see page 289)</td>
<td>&lt;palette&gt; ::= {COLor</td>
</tr>
<tr>
<td>n/a</td>
<td>:HARDcopy:PRINter:LIS T? (see page 290)</td>
<td>&lt;list&gt; ::= [{&lt;printer_spec&gt;] ... [printer_spec]&gt;&lt;index&gt;,&lt;active&gt;,&lt;name&gt;;&lt;index&gt; ::= integer index of printer &lt;active&gt; ::= {Y</td>
</tr>
<tr>
<td>:HARDcopy:STARt (see page 291)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
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

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 282
- ":HARDcopy:STARt" on page 291
- ":HARDcopy:APRinter" on page 284
- ":HARDcopy:PRINter:LIST" on page 290
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:FFEed" on page 286
- ":HARDcopy:INKSaver" on page 287
- ":HARDcopy:LAYout" on page 288
- ":HARDcopy:PALette" on page 289


:Hardcopy:APRinter

(see page 788)

Command Syntax

:Hardcopy:APRinter <active_printer>

<active_printer> ::= {<index> | <name>}

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>

See Also

- "Introduction to :Hardcopy Commands" on page 282
- ":Hardcopy:PRINter:LIST" on page 290
- ":Hardcopy:STARt" on page 291
:HARDcopy:FACTors

(see page 788)

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 282
- ":HARDcopy:STARt" on page 291
- ":HARDcopy:FFEed" on page 286
- ":HARDcopy:INKSaver" on page 287
- ":HARDcopy:LAYout" on page 288
- ":HARDcopy:PALette" on page 289
:HARDcopy:FFEed

(see page 788)

Command Syntax

:HARDcopy:FFEed <ffeed>

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

The HARDcopy:FFEed command controls whether a formfeed is output between the screen image and factors of a hardcopy dump.

ON (or 1) is only valid when PRINter0 or PRINter1 is set as the :HARDcopy:FORMAT type.

Query Syntax

:HARDcopy:FFEed?

The :HARDcopy:FFEed? query returns a flag indicating whether a formfeed is output at the end of the hardcopy dump.

Return Format

<ffeed><NL>

<ffeed> ::= {0 | 1}

See Also

- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:START" on page 291
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:INKSaver" on page 287
- ":HARDcopy:LAYout" on page 288
- ":HARDcopy:PALette" on page 289
**:HARDcopy:INKSaver**

(see page 788)

**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 282
- ":HARDcopy:STARt" on page 291
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:FFEed" on page 286
- ":HARDcopy:LAYout" on page 288
- ":HARDcopy:PALette" on page 289
**:HARDcopy:LAYout**

(see page 788)

**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 282
- ":HARDcopy:STARt" on page 291
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:PALETTE" on page 289
- ":HARDcopy:FFEd" on page 286
- ":HARDcopy:INKSaver" on page 287
**:HARDcopy:PALette**

(see page 788)

**Command Syntax**

`:HARDcopy:PALette <palette>`

<palette> ::= {COLOR | GRAYscale | NONE}

The :HARDcopy:PALette command sets the hardcopy palette color.

**NOTE**

If no printer is connected, NONE is the only valid parameter.

**Query Syntax**

`:HARDcopy:PALette?`

The :HARDcopy:PALette? query returns the selected hardcopy palette color.

**Return Format**

<palette><NL>

<palette> ::= {COL | GRAY | NONE}

**See Also**

- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:STARt" on page 291
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:LAYout" on page 288
- ":HARDcopy:FFEed" on page 286
- ":HARDcopy:INKSaver" on page 287
:HARDCopy:PRINTER:LIST

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 282
- ":HARDCopy:APRINTER" on page 284
- ":HARDCopy:START" on page 291
**:HARDcopy:STARt**

(see page 788)

**Command Syntax**

**:HARDcopy:STARt**

The :HARDcopy:STARt command starts a print job.

**See Also**

- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:APRinter" on page 284
- ":HARDcopy:PRINter:LIST" on page 290
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:FFEed" on page 286
- ":HARDcopy:INKSaver" on page 287
- ":HARDcopy:LAYout" on page 288
- ":HARDcopy:PALette" on page 289
\section*{LISTer Commands}

\subsection*{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.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{Command} & \textbf{Query} & \textbf{Options and Query Returns} \\ \hline
n/a & :LISTer:DATA? (see page 293) & \textless{}binary\_block\textgreater{} ::= comma-separated data with newlines at the end of each row \\
\hline
:LISTer:DISPlay \\{{0 | OFF} | {1 | ON}\} & :LISTer:DISPlay? (see page 294) & \{0 | 1\} \\
\hline
\end{tabular}
\caption{:LISTer Commands Summary}
\end{table}
:LISTer:DATA

(see page 788)

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 292
- ":LISTer:DISPlay" on page 294
- "Definite-Length Block Response Data" on page 109
Commands by Subsystem

:LISTer:DISPlay

(see page 788)

Command Syntax

:LISTer:DISPlay <value>

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

The :LISTer:DISPlay command turns on or off the on-screen lister display.

Query Syntax

:LISTer:DISPlay?

The :LISTer:DISPlay? query returns lister display setting.

Return Format

<value><NL>

<value> ::= {0 | 1}

See Also

- "Introduction to :LISTer Commands" on page 292
- ":LISTer:DATA" on page 293
: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 296.

Table 68  :MARKer Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MARKer:MODE &lt;mode&gt; (see page 297)</td>
<td>:MARKer:MODE? (see page 297)</td>
<td>&lt;mode&gt; ::= {OFF</td>
</tr>
<tr>
<td>:MARKer:X1Position &lt;position&gt;[suffix] (see page 298)</td>
<td>:MARKer:X1Position? (see page 298)</td>
<td>&lt;position&gt; ::= X1 cursor position value in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:MARKer:X1Y1source &lt;source&gt; (see page 299)</td>
<td>:MARKer:X1Y1source? (see page 299)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MARKer:X2Position &lt;position&gt;[suffix] (see page 300)</td>
<td>:MARKer:X2Position? (see page 300)</td>
<td>&lt;position&gt; ::= X2 cursor position value in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:MARKer:X2Y2source &lt;source&gt; (see page 301)</td>
<td>:MARKer:X2Y2source? (see page 301)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td>:MARKer:XDELta? (see page 302)</td>
<td>&lt;return_value&gt; ::= X cursors delta value in NR3 format</td>
</tr>
<tr>
<td>:MARKer:Y1Position &lt;position&gt;[suffix] (see page 303)</td>
<td>:MARKer:Y1Position? (see page 303)</td>
<td>&lt;position&gt; ::= Y1 cursor position value in NR3 format [suffix] ::= {V</td>
</tr>
</tbody>
</table>
**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.

```
:MARKer:X1Y1 NONE;X2Y2 NONE;MODE OFF
```
Commands by Subsystem

:MARKer:MODE

(see page 788)

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 | WAVE)

See Also

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:X1Y1source" on page 299
- ":MARKer:X2Y2source" on page 301
- ":MEASure:SOURce" on page 337
- ":MARKer:X1Position" on page 298
- ":MARKer:X2Position" on page 300
- ":MARKer:Y1Position" on page 303
- ":MARKer:Y2Position" on page 304
Commands by Subsystem

:MARKer:X1Position

(see page 788)

Command Syntax

:MARKer:X1Position <position> [suffix]

<position> ::= X1 cursor position in NR3 format

<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}

The :MARKer:X1Position command:

- Sets :MARKer:MODE to MANual if it is not currently set to WAVEform (see ":MARKer:MODE" on page 297).
- Sets the X1 cursor position to the specified value.

Query Syntax

:MARKer:X1Position?

The :MARKer:X1Position? query returns the current X1 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

NOTE

If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVEform to put the cursors in the front-panel Normal mode.

Return Format

<position><NL>

<position> ::= X1 cursor position in NR3 format

See Also

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:MODE" on page 297
- ":MARKer:X2Position" on page 300
- ":MARKer:X1Y1source" on page 299
- ":MARKer:X2Y2source" on page 301
- ":MEASure:TSTArt" on page 721
**:MARKer:X1Y1source**

(see page 788)

**Command Syntax**

`:MARKer:X1Y1source <source>`

- `<source> ::= {CHAN<n> | FUNCtion | MATH}`
- `<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models`
- `<n> ::= (1 | 2) for the two channel oscilloscope models`

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVeform (see ":MARKer:MODE" on page 297):

- 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, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

| NOTE |
| ~~~ |

MATH is an alias for FUNCTION. The query will return FUNC if the source is FUNCTION or MATH.

**Query Syntax**

`:MARKer:X1Y1source?`

The :MARKer:X1Y1source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

**Return Format**

`<source><NL>`

- `<source> ::= {CHAN<n> | FUNC | NONE}`

See Also

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:MODE" on page 297
- ":MARKer:X2Y2source" on page 301
- ":MEASure:SOURce" on page 337
**:MARKer:X2Position**

(see page 788)

**Command Syntax**

`:MARKer:X2Position <position> [suffix]`

*<position>* ::= X2 cursor position in NR3 format

*<suffix>* ::= {s | ms | us | ns | ps | Hz | kHz | MHz}

The :MARKer:X2Position command:

- Sets :MARKer:MODE to MANual if it is not currently set to WAVeform (see ":MARKer:MODE" on page 297).
- Sets the X2 cursor position to the specified value.

**Query Syntax**

`:MARKer:X2Position?`

The :MARKer:X2Position? query returns current X2 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTOp command/query.

**NOTE**

If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

**Return Format**

*<position>*<NL>

*<position>* ::= X2 cursor position in NR3 format

**See Also**

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:MODE" on page 297
- ":MARKer:X1Position" on page 298
- ":MARKer:X2Y2source" on page 301
- "MEASure:TSTOp" on page 722
**:MARKer:X2Y2source**

(see page 788)

**Command Syntax**

`:MARKer:X2Y2source <source>`

```
<source> ::= (CHANnel<n> | FUNCtion | MATH)
<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models
```

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVeform (see ":MARKer:MODE" on page 297):

- 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, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

**NOTE**

MATH is an alias for FUNCTION. The query will return FUNC if the source is FUNCTION or MATH.

**Query Syntax**

`:MARKer:X2Y2source?`

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

**Return Format**

```
<source><NL>
<source> ::= (CHAN<n> | FUNC | NONE)
```

**See Also**

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:MODE" on page 297
- ":MARKer:X1Y1source" on page 299
- ":MEASure:SOURce" on page 337
5  Commands by Subsystem

:MARKer:XDELta

(see page 788)

Query Syntax  :MARKer:XDELta?

The MARKer:XDELta? query returns the value difference between the current X1 and X2 cursor positions.

\[
X_{\text{delta}} = (\text{Value at X2 cursor}) - (\text{Value at X1 cursor})
\]

NOTE  If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format  <value><NL>

<value> ::= difference value in NR3 format.

See Also  • "Introduction to :MARKer Commands" on page 296
  • ":MARKer:MODE" on page 297
  • ":MARKer:X1Position" on page 298
  • ":MARKer:X2Position" on page 300
  • ":MARKer:X1Ysource" on page 299
  • ":MARKer:X2Ysource" on page 301
:MARKer:Y1Position

(see page 788)

**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 297), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y1 cursor position to the specified value.

When the :MARKer:MODE is set to WAVeform, Y positions cannot be set.

**Query Syntax**

:MARKer:Y1Position?

The :MARKer:Y1Position? query returns current Y1 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTArt command/query.

**NOTE**

If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

**Return Format**

<position><NL>

<position> ::= Y1 cursor position in NR3 format

**See Also**

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:MODE" on page 297
- ":MARKer:X1Y1source" on page 299
- ":MARKer:X2Y2source" on page 301
- ":MARKer:Y2Position" on page 304
- ":MEASure:VSTArt" on page 727
**:MARKer:Y2Position**

(see page 788)

**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 297), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y2 cursor position to the specified value.

When the :MARKer:MODE is set to WAVeform, Y positions cannot be set.

**Query Syntax**

:MARKer:Y2Position?

The :MARKer:Y2Position? query returns current Y2 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

**NOTE**

If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

**Return Format**

<position><NL>

<position> ::= Y2 cursor position in NR3 format

**See Also**

- "Introduction to :MARKer Commands" on page 296
- ":MARKer:MODE" on page 297
- ":MARKer:X1Y1source" on page 299
- ":MARKer:X2Y2source" on page 301
- ":MARKer:Y1Position" on page 303
- ":MEASure:VSTOp" on page 728
**:MARKer:YDELta**

(see page 788)

**Query Syntax**

**:MARKer:YDELta?**

The **:MARKer:YDELta?** query returns the value difference between the current Y1 and Y2 cursor positions.

\[
\text{Ydelta} = (\text{Value at Y2 cursor}) - (\text{Value at Y1 cursor})
\]

**NOTE**

If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set **:MARKer:MODE** to MANual or WAVeform to put the cursors in the front-panel Normal mode.

**Return Format**

\[
<\text{value}><\text{NL}>
\]

<value> ::= difference value in NR3 format

**See Also**

- "Introduction to **:MARKer Commands**" on page 296
- "**:MARKer:MODE**" on page 297
- "**:MARKer:X1Y1source**" on page 299
- "**:MARKer:X2Y2source**" on page 301
- "**:MARKer:Y1Position**" on page 303
- "**:MARKer:Y2Position**" on page 304
5 Commands by Subsystem

:MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 312.

Table 69 :MEASure Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:CLEAR (see page 314)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:MEASure:COUNter [&lt;source&gt;] (see page 315)</td>
<td>:MEASure:COUNter? [&lt;source&gt;] (see page 315)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:DEFine DELay, &lt;delay spec&gt; (see page 316)</td>
<td>:MEASure:DEFine? DELay (see page 317)</td>
<td>&lt;delay spec&gt; ::= &lt;edge_spec1&gt;,&lt;edge_spec2&gt; edge_spec1 ::= [&lt;slope&gt;]&lt;occurrence&gt; edge_spec2 ::= [&lt;slope&gt;]&lt;occurrence&gt; &lt;slope&gt; ::= {+</td>
</tr>
<tr>
<td>:MEASure:DEFine THResholds, &lt;threshold spec&gt; (see page 316)</td>
<td>:MEASure:DEFine? THResholds (see page 317)</td>
<td>&lt;threshold spec&gt; ::= {STANdard}</td>
</tr>
<tr>
<td>:MEASure:DELay [&lt;source1&gt;] [,&lt;source2&gt;] (see page 319)</td>
<td>:MEASure:DELay? [&lt;source1&gt;] [,&lt;source2&gt;] (see page 319)</td>
<td>&lt;source1,2&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:DUTYcycle [&lt;source&gt;] (see page 321)</td>
<td>:MEASure:DUTYcycle? [&lt;source&gt;] (see page 321)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
Table 69: :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:FA\lltime [&lt;source&gt;] (see page 322)</td>
<td>:MEASure:FA\lltime? [&lt;source&gt;] (see page 322)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:FE\quency [&lt;source&gt;] (see page 323)</td>
<td>:MEASure:FE\quency? [&lt;source&gt;] (see page 323)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:N\width [&lt;source&gt;] (see page 324)</td>
<td>:MEASure:N\width? [&lt;source&gt;] (see page 324)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:O\veR\vveat [&lt;source&gt;] (see page 325)</td>
<td>:MEASure:O\veR\vveat? [&lt;source&gt;] (see page 325)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:PER\iod [&lt;source&gt;] (see page 327)</td>
<td>:MEASure:PER\iod? [&lt;source&gt;] (see page 327)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
Table 69 :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:PHASe [&lt;source1&gt;] [,&lt;source2&gt;] (see page 328)</td>
<td>:MEASure:PHASe? [&lt;source1&gt;] [,&lt;source2&gt;] (see page 328)</td>
<td>&lt;source1,2&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:PREShoot [&lt;source&gt;] (see page 329)</td>
<td>:MEASure:PREShoot? [&lt;source&gt;] (see page 329)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:PWIDth [&lt;source&gt;] (see page 330)</td>
<td>:MEASure:PWIDth? [&lt;source&gt;] (see page 330)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td>:MEASure:RESults? &lt;result_list&gt; (see page 331)</td>
<td>&lt;result_list&gt; ::= comma-separated list of measurement results</td>
</tr>
<tr>
<td>:MEASure:RISeetime [&lt;source&gt;] (see page 334)</td>
<td>:MEASure:RISeetime? [&lt;source&gt;] (see page 334)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:SDEViation [&lt;source&gt;] (see page 335)</td>
<td>:MEASure:SDEViation? [&lt;source&gt;] (see page 335)</td>
<td>&lt;source&gt; ::= (CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:SHOW (1</td>
<td>ON) (see page 336)</td>
<td>:MEASure:SHOW? (see page 336)</td>
</tr>
</tbody>
</table>
### Table 69 :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:SOURce &lt;source1&gt;,&lt;source2&gt; (see page 337)</td>
<td>:MEASure:SOURce? (see page 337)</td>
<td>&lt;source1,2&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASure:STATistics &lt;type&gt; (see page 339)</td>
<td>:MEASure:STATistics? (see page 339)</td>
<td>&lt;type&gt; ::= {{ON</td>
</tr>
<tr>
<td>:MEASure:STATistics:INCReement (see page 340)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:MEASure:STATistics:RESET (see page 341)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:MEASure:TEDGe? &lt;slope&gt;&lt;occurrence&gt;[, &lt;source&gt;] (see page 342)</td>
<td>&lt;slope&gt; ::= direction of the waveform &lt;occurrence&gt; ::= the transition to be reported &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
### Table 69 :MEASure Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| n/a      | :MEASure:TVALue? </br> <value>,  </br> [<slope>]<occurrence>  </br> [,<source>] (see page 344) | <value> ::= voltage level that the waveform must cross.  
<slope> ::= direction of the waveform when <value> is crossed.  
<occurrence> ::= transitions reported.  
<return_value> ::= time in seconds of specified voltage crossing in NR3 format  
<source> ::= (CHANnel<n> | FUNCTION | MATH) for DSO models  
<source> ::= (CHANnel<n> | DIGital0,...,DIGital15 | FUNCTION | MATH) for MSO models  
<n> ::= 1-2 or 1-4 in NR1 format |
| :MEASure:VAMPlitude | :MEASure:VAMPlitude? </br> [<source>] (see page 346) | <source> ::= (CHANnel<n> | FUNCTION | MATH)  
<n> ::= 1-2 or 1-4 in NR1 format  
<return_value> ::= the amplitude of the selected waveform in volts in NR3 format |
| :MEASure:VAVerage | :MEASure:VAVerage? </br> [<interval>][,]<source>] (see page 347) | <interval> ::= {CYCLE | DISplay | AUTO}  
<source> ::= (CHANnel<n> | FUNCTION | MATH)  
<n> ::= 1-2 or 1-4 in NR1 format  
<return_value> ::= calculated average voltage in NR3 format |
| :MEASure:VBASe | :MEASure:VBASe? </br> [<source>] (see page 348) | <source> ::= (CHANnel<n> | FUNCTION | MATH)  
<n> ::= 1-2 or 1-4 in NR1 format  
<base_voltage> ::= voltage at the base of the selected waveform in NR3 format |
| :MEASure:VMAX | :MEASure:VMAX? </br> [<source>] (see page 349) | <source> ::= (CHANnel<n> | FUNCTION | MATH)  
<n> ::= 1-2 or 1-4 in NR1 format  
<return_value> ::= maximum voltage of the selected waveform in NR3 format |
### Table 69: MEASURE Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASURE:VMIN [&lt;source&gt;] (see page 350)</td>
<td>:MEASURE:VMIN? [&lt;source&gt;] (see page 350)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:VPP [&lt;source&gt;] (see page 351)</td>
<td>:MEASURE:VPP? [&lt;source&gt;] (see page 351)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:VRATIO [&lt;source1&gt;,&lt;source2&gt;] (see page 328)</td>
<td>:MEASURE:VRATIO? [&lt;source1&gt;,&lt;source2&gt;] (see page 352)</td>
<td>&lt;source1,2&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:VRMS [&lt;interval&gt;][,&lt;source&gt;] (see page 353)</td>
<td>:MEASURE:VRMS? [&lt;interval&gt;][,&lt;source&gt;] (see page 353)</td>
<td>&lt;interval&gt; ::= {CYCLE</td>
</tr>
<tr>
<td>n/a</td>
<td>:MEASURE:VTIME? [vtime][,&lt;source&gt;] (see page 354)</td>
<td>&lt;vtime&gt; ::= displayed time from trigger in seconds in NR3 format &lt;return_value&gt; ::= voltage at the specified time in NR3 format &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:VTOP [&lt;source&gt;] (see page 355)</td>
<td>:MEASURE:VTOP? [&lt;source&gt;] (see page 355)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:MEASURE:WINDow &lt;window&gt; (see page 356)</td>
<td>:MEASURE:WINDow? (see page 356)</td>
<td>&lt;window&gt; ::= {MAIN</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Measurement Type</th>
<th>Portion of waveform that must be displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>period, duty cycle, or frequency</td>
<td>at least one complete cycle</td>
</tr>
<tr>
<td>pulse width</td>
<td>the entire pulse</td>
</tr>
<tr>
<td>rise time</td>
<td>rising edge, top and bottom of pulse</td>
</tr>
<tr>
<td>fall time</td>
<td>falling edge, top and bottom of pulse</td>
</tr>
</tbody>
</table>

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
Commands by Subsystem

:MEASure:CLEar

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

See Also

- "Introduction to :MEASure Commands" on page 312
**:MEASure:COUNter**

(see page 788)

**Command Syntax**

```
:MEASure:COUNter [<source>]
```

- `<source>` ::= ( `<digital channels>` | CHANnel<n> | EXTernal )
- `<digital channels>` ::= DIGital0,...,DIGital15 for the MSO models
- `<n>` ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
- `<n>` ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:COUNter command installs a screen measurement and starts a counter measurement. If the optional source parameter is specified, the current source is modified. Any channel except Math may be selected for the source.

The counter measurement counts trigger level crossings at the selected trigger slope and displays the results in Hz. The gate time for the measurement is automatically adjusted to be 100 ms or twice the current time window, whichever is longer, up to 1 second. The counter measurement can measure frequencies up to 125 MHz. The minimum frequency supported is 1/(2 X gate time).

The Y cursor shows the the edge threshold level used in the measurement. Only one counter measurement may be displayed at a time.

This command is not available if the source is MATH.

**Query Syntax**

```
:MEASure:COUNter? [<source>]
```

The :MEASure:COUNter? query measures and outputs the counter frequency of the specified source.

The :MEASure:COUNter? query times out if the counter measurement is installed on the front panel. Use :MEASure:CLEar to remove the front-panel measurement before executing the :MEASure:COUNter? query.

**Return Format**

```
<source><NL>
```

- `<source>` ::= count in Hertz in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:FREQuency" on page 323
- ":MEASure:CLEar" on page 314
5  Commands by Subsystem

:MEASure:DEFine

(see page 788)

Command Syntax

:MEASure:DEFine <meas_spec>

<meas_spec> ::= (DELay | THResholds)

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DELay specification or the THResholds values. For example, changing the THResholds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

<table>
<thead>
<tr>
<th>MEASure Command</th>
<th>DELay</th>
<th>THResholds</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUTYcycle</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>DELay</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>FALLtime</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>FREQuency</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>NWIDth</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>OVERshoot</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>PERiod</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>PHASE</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>PREShoot</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>PWIDth</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>RISEtime</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>VAVerage</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>VRMS</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

:MEASure:DEFine DELay Command Syntax

:MEASure:DEFine DELay,<delay spec>

<delay spec> ::= <edge_spec1>,<edge_spec2>

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

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

<slope> ::= (+ | -)

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

\[ \text{delay} = t(\text{edge_spec2}) - t(\text{edge_spec1}) \]

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.

**NOTE**

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

**MEASure:DEFine**

**THResholds**

**Command Syntax**

:MEASure:DEFine THResholds,<threshold_spec>

<threshold_spec> ::= {STANdard} |

| {<threshold_mode>,<upper>,<middle>,<lower>} |

<threshold_mode> ::= {PERCent | ABSolute}

for <threshold_mode> = PERCent:

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

for <threshold_mode> = ABSolute:

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

- STANdard threshold specification sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.

- Threshold mode PERCent sets the measurement thresholds to any user-defined percentages between 5% and 95% of values between Vbase and Vtop.

- Threshold mode ABSolute sets the measurement thresholds to absolute values. ABSolute thresholds are dependent on channel scaling (:CHANnel<n>:RANGE or "CHANnel<n>:SCALe" on page 234:CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

**Query Syntax**

:MEASure:DEFine? <meas_spec>

<meas_spec> ::= (DELay | THResholds)

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

for `<meas_spec> = DELay`:

\{( <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2> ) \} <NL>

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

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

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

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

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

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

for `<threshold spec> = STANdard`:

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

See Also

- "Introduction to :MEASure Commands" on page 312
- "MEASure:DELay" on page 319
- "MEASure:SOURce" on page 337
- "CHANnel<n>:RANGE" on page 233
- "CHANnel<n>:SCALe" on page 234
- "CHANnel<n>:PROBe" on page 227
- "CHANnel<n>:UNITs" on page 235
**:MEASure:DELay**

(see page 788)

**Command Syntax**

**:MEASure:DELay [<source1>][,<source2>]**

<source1>, <source2> ::= {CHANnel<n> | FUNCTION | MATH}

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:DELay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

\[ \text{delay} = t(\text{edge spec 2}) - t(\text{edge spec 1}) \]

where the <edge spec> definitions are set by the :MEASure:DEFine command

**NOTE**


The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, you can select the edge polarity, but the instrument will select the edges that will make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DELay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

**Query Syntax**

**:MEASure:DELay? [<source1>][,<source2>]**

The :MEASure:DELay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Also in the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. *It is the middle threshold value that is used when performing the delay query.* The standard upper, middle, and lower measurement thresholds are 90%, 50%, and 10% values between Vbase and
Vtop. If you want to move the delay measurement point nearer to Vtop or Vbase, you must change the threshold values with the :MEASure:DEFine THResholds command.

**Return Format**

<value><NL>

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

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:DEFine" on page 316
- ":MEASure:PHASe" on page 328
**:MEASure:DUTYcycle**

(see page 788)

**Command Syntax**

```
:MEASure:DUTYcycle [<source>]
```

- `<source>` ::= ( <digital channels> | CHANnel<n> | FUNCTION | MATH )
- `<digital channels>` ::= DIGital0,..,DIGital15 for the MSO models
- `<n>` ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
- `<n>` ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:DUTYcycle command installs a screen measurement and starts a duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

**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:

\[
\text{duty cycle} = \left( \frac{\text{positive pulse width}}{\text{period}} \right) \times 100
\]

**Return Format**

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

- `<value>` ::= ratio of positive pulse width to period in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:PERiod" on page 327
- ":MEASure:PWIDth" on page 330
- ":MEASure:SOURce" on page 337

**Example Code**

- "Example Code" on page 338

**NOTE**

The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).
**:MEASure:FALLtime**

(see page 788)

**Command Syntax**

`:MEASure:FALLtime [<source>]`

- `<source>` ::= {CHANnel<n> | FUNCtion | MATH}
- `<n>` ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
- `<n>` ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax**

`:MEASure:FALLtime? [<source>]`

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

\[
\text{fall time} = \text{time at lower threshold} - \text{time at upper threshold}
\]

**Return Format**

`<value><NL>`

- `<value>` ::= time in seconds between the lower threshold and upper threshold in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- "MEASure:RISetime" on page 334
- "MEASure:SOURce" on page 337
:MEASure:FREQuency

Command Syntax

:MEASure:FREQuency [<source>]

<source> ::= \(<\text{digital channels}> \mid \text{CHANnel}<n> \mid \text{FUNCtion} \mid \text{MATH}\)

<digital channels> ::= DIGital0,...,DIGital15 for the MSO models

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

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

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

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:FREQuency? [<source>]

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

Return Format

<source><NL>

<source> ::= frequency in Hertz in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:PERiod" on page 327

Example Code

- "Example Code" on page 338
5 Commands by Subsystem

:MEASure:NWIDth

(see page 788)

Command Syntax

:MEASure:NWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}
<digital channels> ::= DIGital0,..,DIGital15 for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:NWIDth? [<source>]

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

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

Return Format

<value><NL>

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

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:PWIDth" on page 330
- ":MEASure:PERiod" on page 327
Commands by Subsystem

:MEASure:OVERshoot

(see page 788)

Command Syntax

:MEASure:OVERshoot [<source>]

<source> ::= (CHANnel<n> | FUNCtion | MATH)

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

The :MEASure:OVERshoot command installs a screen measurement and
starts an overshoot measurement. If the optional source parameter is
specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:OVERshoot? [<source>]

The :MEASure:OVERshoot? query measures and returns the overshoot of
the edge closest to the trigger reference, displayed on the screen. The
method used to determine overshoot is to make three different vertical
value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on
whether the edge is rising or falling.

For a rising edge:

overshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

For a falling edge:

overshoot = ((Vbase-Vmin) / (Vtop-Vbase)) x 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 312
• ":MEASure:PREShoot" on page 329
• ":MEASure:SOURce" on page 337
• ":MEASure:VMAX" on page 349
• ":MEASure:VTOP" on page 355
• ":MEASure:VBASe" on page 348
• ":MEASure:VMIN" on page 350
**:MEASure:PERiod**

(see page 788)

**Command Syntax**

`:MEASure:PERiod [<source>]`

<source> ::= (<digital channels> | CHANnel<n> | FUNCTION | MATH)
<digital channels> ::= DIGital0,...,DIGital15 for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

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

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax**

`:MEASure:PERiod? [<source>]`

The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.

IF the edge closest to the trigger reference on screen is rising:

THEN period = (time at trailing rising edge - time at leading rising edge)

ELSE period = (time at trailing falling edge - time at leading falling edge)

**Return Format**

<value><NL>

<value> ::= waveform period in seconds in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- "MEASure:SOURce" on page 337
- "MEASure:NWIDth" on page 324
- "MEASure:PWIDth" on page 330
- "MEASure:FREQuency" on page 323

**Example Code**

- "Example Code" on page 338
Commands by Subsystem

:MEASure:PHASe

(see page 788)

Command Syntax

:MEASure:PHASe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCTION | MATH}

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

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

Query Syntax

:MEASure:PHASe? [<source1>][,<source2>]

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELay for more detail on selecting the 2nd edge.

The phase is calculated as follows:

\[
\text{phase} = \left(\frac{\text{delay}}{\text{period of input 1}}\right) \times 360
\]

Return Format

<value><NL>

<value> ::= the phase angle value in degrees in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:DELay" on page 319
- ":MEASure:PERiod" on page 327
- ":MEASure:SOURce" on page 337
:MEASure:PRESShoot

(see page 788)

Command Syntax

:MEASure:PRESShoot [<source>]

<source> ::= (CHANnel<n> | FUNCtion | MATH)

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:PRESShoot 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:PRESShoot? [<source>]

The :MEASure:PRESShoot? 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:

\[
\text{preshoot} = \left( \frac{(V_{\text{min}} - V_{\text{base}})}{(V_{\text{top}} - V_{\text{base}})} \right) \times 100
\]

For a falling edge:

\[
\text{preshoot} = \left( \frac{(V_{\text{max}} - V_{\text{top}})}{(V_{\text{top}} - V_{\text{base}})} \right) \times 100
\]

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 312
- ":MEASure:SOURce" on page 337
- ":MEASure:VMIN" on page 350
- ":MEASure:VMAX" on page 349
- ":MEASure:VTOP" on page 355
- ":MEASure:VBASE" on page 348
:MEASure:PWIDth

(see page 788)

Command Syntax

:MEASure:PWIDth [<source>]

<source> ::= ({digital channels} | CHANnel<n> | FUNCtion | MATH)
<digital channels> ::= DIGital0,...,DIGital15 for the MSO models
<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:PWIDth? [<source>]

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:
THEN width = (time at trailing falling edge - time at leading rising edge)
ELSE width = (time at leading falling edge - time at leading rising edge)

Return Format

/value/<NL>

/value> ::= width of positive pulse in seconds in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:NWIDth" on page 324
- ":MEASure:PERiod" on page 327
**Query Syntax**  
:MEASure:RESults?

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values.

If more than one measurement is running continuously, the :MEASure:RESults return values are duplicated for each continuous measurement from the first to last (left to right) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of four continuous measurements that can be continuously displayed at a time.

When no quick measurements are installed, the :MEASure:RESults? query returns nothing (empty string). When the count for any of the measurements is 0, the value of infinity (9.9E+37) is returned for the min, max, mean, and standard deviation.

**Return Format**

```
<result_list><NL>
<result_list> ::= comma-separated list of measurement results
```

The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

<table>
<thead>
<tr>
<th>Measurement label</th>
<th>current</th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>std dev</th>
<th>count</th>
</tr>
</thead>
</table>

Measurement label, current, min, max, mean, std dev, and count are only returned if :MEASure:STATistics is ON.

If :MEASure:STATistics is set to CURRent, MIN, MAX, MEAN, STDDev, or COUNT only that particular statistic value is returned for each measurement that is on.

**See Also**

- "Introduction to :MEASure Commands" on page 312
- "MEASure:STATistics" on page 339

**Example Code**

```
' This program shows the InfiniiVision oscilloscopes' measurement statistics commands.
' -------------------------------------------------------------------

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::130.29.70.228::inst0::INSTR")

' Initialize.
myScope.IO.Clear ' Clear the interface.
myScope.WriteString "*RST" ' Reset to the defaults.
myScope.WriteString "*CLS" ' Clear the status data structures.
myScope.WriteString "*AUToscale"

' Install some measurements.
myScope.WriteString ":MEASure:SOURce CHANnel1" ' Input source.

Dim MeasurementArray(3) As String
MeasurementArray(0) = "FREQuency"
MeasurementArray(1) = "DUTYcycle"
MeasurementArray(2) = "VAMPlitude"
MeasurementArray(3) = "VPP"
Dim Measurement As Variant

For Each Measurement In MeasurementArray
  myScope.WriteString ":MEASure:" + Measurement
  myScope.WriteString ":MEASure:" + Measurement + "?"
  varQueryResult = myScope.ReadNumber ' Read measurement value.
  Debug.Print Measurement + ": " + FormatNumber(varQueryResult, 4)
Next

myScope.WriteString ":MEASure:STATistics:RESet" ' Reset stats.
Sleep 5000 ' Wait for 5 seconds.

' Select the statistics results type.
Dim ResultsTypeArray(6) As String
ResultsTypeArray(0) = "CURRent"
ResultsTypeArray(1) = "MINimum"
ResultsTypeArray(2) = "MAXimum"
ResultsTypeArray(3) = "MEAN"
ResultsTypeArray(4) = "STDDev"
ResultsTypeArray(5) = "COUNT"
ResultsTypeArray(6) = "ON" ' All results.
Dim ResultType As Variant

Dim ResultsList() = 

Dim ValueColumnArray(6) As String
ValueColumnArray(0) = "Meas_Lbl"
ValueColumnArray(1) = "Current"
ValueColumnArray(2) = "Min"
ValueColumnArray(3) = "Max"
ValueColumnArray(4) = "Mean"
ValueColumnArray(5) = "Std_Dev"
ValueColumnArray(6) = "Count"
Dim ValueColumn As Variant

For Each ResultType In ResultsTypeArray
    myScope.WriteString ":MEASure:STATistics " + ResultType
    ' Get the statistics results.
    Dim intCounter As Integer
    intCounter = 0
    myScope.WriteString ":MEASure:RESults?"
    ResultsList() = myScope.ReadList

    For Each Measurement In MeasurementArray
        If ResultType = "ON" Then ' All statistics.
            For Each ValueColumn In ValueColumnArray
                If VarType(ResultsList(intCounter)) <> vbString Then
                    Debug.Print "Measure statistics result CH1, " + Measurement + ", " + ValueColumn + ": " + FormatNumber(ResultsList(intCounter), 4)
                Else ' Result is a string (e.g., measurement label).
                    Debug.Print "Measure statistics result CH1, " + Measurement + ", " + ValueColumn + ": " + ResultsList(intCounter)
                End If
                intCounter = intCounter + 1
            Next
        Else ' Specific statistic (e.g., Current, Max, Min, etc.).
            Debug.Print "Measure statistics result CH1, " + Measurement + ", " + ResultType + ": " + FormatNumber(ResultsList(intCounter), 4)
            intCounter = intCounter + 1
        End If
    Next
Else ' Specific statistic (e.g., Current, Max, Min, etc.).
    Debug.Print "Measure statistics result CH1, " + Measurement + ", " + ResultType + ": " + FormatNumber(ResultsList(intCounter), 4)
    intCounter = intCounter + 1
End If
Next
Next
Exit Sub
VisaComError:
    MsgBox "VISA COM Error:" + vbCrLf + Err.Description
End Sub
5 Commands by Subsystem

:MEASure:RISetime

(see page 788)

Command Syntax

:MEASure: RISetime [<source>]

<source> ::= (CHANnel<n> | FUNCtion | MATH)

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

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 312
- "MEASure:SOURce" on page 337
- "MEASure:FALLtime" on page 322
**:MEASure:SDEViation**

(see page 788)

**Command Syntax**

```
:MEASure:SDEViation [<source>]

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

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

The :MEASure:SDEViation command installs a screen measurement and starts std deviation measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax**

```
:MEASure:SDEViation? [<source>]
```

The :MEASure:SDEViation? query measures and outputs the std deviation of the selected waveform. The oscilloscope computes the std deviation on all displayed data points.

**Return Format**

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

<value> ::= calculated std deviation value in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
Commands by Subsystem

:MEASURE:SHOW

(see page 788)

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 312
:MEASure:SOURce

(see page 788)

Command Syntax

:MEASure:SOURce <source1>[,<source2>]

<source1>,<source2> ::= (<digital channels> | CHAnnel<n> | FUNCtion | MATH | EXTernal)

digital channels ::= DIGital0,...,DIGital15 for the MSO models

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command.

If a source is specified for any measurement, the current source is changed to this new value.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHAnnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

EXTernal is only a valid source for the counter measurement (and <source1>).

Query Syntax

:MEASure:SOURce?

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 only applies to :MEASure:DELay and :MEASure:PHASe measurements.

NOTE

MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Return Format

<source1>,<source2><NL>

<source1>,<source2> ::= (<digital channels> | CHAn<n> | FUNC | EXT | NONE)

See Also:

- "Introduction to :MEASure Commands" on page 312
- ":MARKer:MODE" on page 297
- ":MARKer:X1Y1source" on page 299
- ":MARKer:X2Y2source" on page 301
- ":MEASure:DELay" on page 319
- ":MEASure:PHASe" on page 328
Example Code

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.
myScope.WriteString " :MEASURE:SOURCE CHANNEL1"  ' Source to measure.
myScope.WriteString " :MEASURE:FREQUENCY?"  ' Query for frequency.
varQueryResult = myScope.ReadNumber  ' Read frequency.
MsgBox "Frequency:" + vbCrLf + FormatNumber(varQueryResult / 1000, 4) + " kHz"

myScope.WriteString " :MEASURE:DUTYCYCLE?"  ' Query for duty cycle.
varQueryResult = myScope.ReadNumber  ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(varQueryResult, 3) + "%"

myScope.WriteString " :MEASURE:RISETIME?"  ' Query for risetime.
varQueryResult = myScope.ReadNumber  ' Read risetime.
MsgBox "Risetime:" + vbCrLf + FormatNumber(varQueryResult * 1000000, 4) + " us"

myScope.WriteString " :MEASURE:VPP?"  ' Query for Pk to Pk voltage.
varQueryResult = myScope.ReadNumber  ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + FormatNumber(varQueryResult, 4) + " V"

myScope.WriteString " :MEASURE:VMAX?"  ' Query for Vmax.
varQueryResult = myScope.ReadNumber  ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + FormatNumber(varQueryResult, 4) + " V"

Example program from the start: "VISA COM Example in Visual Basic" on
page 814
:MEASure:STATistics

(see page 788)

Command Syntax
:MEASure:STATistics <type>

$type ::= \{ON \mid 1\} \mid CURRent \mid MINimum \mid MAXimum \mid MEAN \mid STDDev \mid COUNT\}

The :MEASure:STATistics command determines the type of information returned by the :MEASure:RESults? query. ON means all the statistics are on.

Query Syntax
:MEASure:STATistics?

The :MEASure:STATistics? query returns the current statistics mode.

Return Format
$type<\text{NL}>

$type ::= \{ON \mid CURR \mid MIN \mid MAX \mid MEAN \mid STDD \mid COUN\}

See Also
- "Introduction to :MEASure Commands" on page 312
- ":MEASure:RESults" on page 331
- ":MEASure:STATistics:RESet" on page 341
- ":MEASure:STATistics:INCRement" on page 340

Example Code
- "Example Code" on page 331
:MEASure:STATistics:INCRement

(see page 788)

Command Syntax :MEASure:STATistics:INCRement

This command updates the statistics once (incrementing the count by one) using the current measurement values. It corresponds to the front panel Increment Statistics softkey in the Measurement Statistics Menu. This command lets you, for example, gather statistics over multiple pulses captured in a single acquisition. To do this, change the horizontal position and enter the command for each new pulse that is measured.

This command is only allowed when the oscilloscope is stopped and quick measurements are on.

The command is allowed in segmented acquisition mode even though the corresponding front panel softkey is not available.

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:STATistics" on page 339
- ":MEASure:STATistics:RESet" on page 341
- ":MEASure:RESults" on page 331
:MEASURE:STATISTICS:RESET

Command Syntax: :MEASURE:STATISTICS:RESET

This command resets the measurement statistics, zeroing the counts.

Note that the measurement (statistics) configuration is not deleted.

See Also:
- "Introduction to :MEASURE Commands" on page 312
- "MEASURE:STATISTICS" on page 339
- "MEASURE:RESULTS" on page 331
- "MEASURE:STATISTICS:INCREMENT" on page 340

Example Code: "Example Code" on page 331
:MEASure:TEDGe

(see page 788)

Query Syntax

:MEASure:TEDGe? <slope><occurrence>[,<source>]

<slope> ::= direction of the waveform. A rising slope is indicated by a space or plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing from the left screen edge is reported. If the number is two, the second crossing is reported, etc.

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

<digital channels> ::= DIGital0,...,DIGital15 for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

When the :MEASure:TEDGe query is sent, the displayed signal is searched for the specified transition. The time interval between the trigger event and this occurrence is returned as the response to the query. The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the midpoint threshold in the positive direction. Once this crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified vertical value, or if the waveform does not cross the specified vertical value for the specific number of times in the direction specified.

You can make delay and phase measurements using the MEASure:TEDGe command:

    Delay = time at the nth rising or falling edge of the channel - time at the same edge of another channel

    Phase = (delay between channels / period of channel) x 360

For an example of making a delay and phase measurement, see ":MEASure:TEDGe Code" on page 343.

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

\(<value><NL>\)

\(<value> ::= \text{time in seconds of the specified transition in NR3 format}\)

:MEASURE:TEDGE Code

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString "MEASURE:TEDGE? +1, CHAN1"

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString "MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.
' Query time at 1st rising edge on ch1.
myScope.WriteString "MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

Example program from the start: "VISA COM Example in Visual Basic" on page 814

See Also

- "Introduction to :MEASURE Commands" on page 312
- "MEASURE:TVALUE" on page 344
- "MEASURE:VTIME" on page 354
:MEASure:TVALue

(see page 788)

Query Syntax

:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

/value> ::= the vertical value that the waveform must cross. The value can be volts or a math function value such as dB, Vs, or V/s.

/slope> ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

/occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

/source> ::= {CHANnel<n> | FUNCtion | MATH}

/n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models

/n> ::= (1 | 2) for the two channel oscilloscope models

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

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 312
- ":MEASure:TEDGe" on page 342
- ":MEASure:VTIMe" on page 354
5 Commands by Subsystem

:MEASURE:VAMPLITUDE

(see page 788)

Command Syntax
:MEASURE:VAMPLITUDE [<source>]

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

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

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

Query Syntax
:MEASURE:VAMPLITUDE? [<source>]

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

vertical amplitude = Vtop - Vbase

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

See Also
• "Introduction to :MEASURE Commands" on page 312
• "MEASURE:SOURce" on page 337
• "MEASURE:VBASE" on page 348
• "MEASURE:VTOP" on page 355
• "MEASURE:VPP" on page 351
**:MEASure:VAVerage**

(see page 788)

**Command Syntax**

:MEASure:VAVerage [<interval>] [,] [<source>]

<interval> ::= (CYCle | DISPlay | AUTO)

<source> ::= (CHANnel<n> | FUNCTION | MATH)

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

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax**


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.

The :MEASure:VRMS? query returns the average value of the selected waveform. How the average value is measured depends on the <interval> specification:

- If <interval> is CYCle, the average value is measured on an integral number of periods of the displayed signal. If less than three edges are present, the measurement fails, and +9.9E+37 is returned.
- If <interval> is DISPlay, the average value is measured on all displayed data points.
- If <interval> is AUTO or is not specified, the measurement attempts to compute a value using the CYCle interval. If less than three edges are present, the measurement is computed using the DISPlay interval.

**Return Format**

<value><NL>

<value> ::= calculated average value in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- "MEASure:SOURce" on page 337
5 Commands by Subsystem

:MEASure:VBASe

(see page 788)

Command Syntax

:MEASure:VBASe [<source>]

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

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

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

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:VBASe? [<source>]

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format

<base_voltage><NL>

<base_voltage> ::= value at the base of the selected waveform in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 312
- "MEASure:SOURce" on page 337
- "MEASure:VTOP" on page 355
- "MEASure:VAMPplitude" on page 346
- "MEASure:VMIN" on page 350
### :MEASure:VMAX

(see page 788)

**Command Syntax**

```
:MEASure:VMAX [<source>]
```

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

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

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

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

**Query Syntax**

```
:MEASure:VMAX? [<source>]
```

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

**Return Format**

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

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

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:VMIN" on page 350
- ":MEASure:VPP" on page 351
- ":MEASure:VTOP" on page 355
Commands by Subsystem

:MEASure:VMIN

(see page 788)

Command Syntax

:MEASure:VMIN [<source>]

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

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

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

Query Syntax

:MEASure:VMIN? [<source>]

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

Return Format

<value><NL>

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

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:VBASE" on page 348
- ":MEASure:VMAX" on page 349
- ":MEASure:VPP" on page 351
:MEASure:VPP

(see page 788)

Command Syntax

:MEASure:VPP [<source>]

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

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

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

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

:MEASure:VPP? [<source>]

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

\[ Vpp = V_{\text{max}} - V_{\text{min}} \]

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 312
- ":MEASure:SOURce" on page 337
- ":MEASure:VMAX" on page 349
- ":MEASure:VMIN" on page 350
- ":MEASure:VAMPlitude" on page 346
*MEASure:VRTatio*

(see page 788)

**Command Syntax**

```
:MEASure:VRTatio [<source1>][,<source2>]
```

<source1>, <source2> ::= {CHANnel<n> | FUNCTION | MATH}

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

The :MEASure:VRTatio command places the instrument in the continuous measurement mode and starts a ratio measurement.

**Query Syntax**

```
:MEASure:VRTatio? [<source1>][,<source2>]
```

The :MEASure:VRTatio? query measures and returns the ratio of AC RMS values of the specified sources expressed as dB.

**Return Format**

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

<value> ::= the ratio value in dB in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:VRMS" on page 353
- ":MEASure:SOURce" on page 337
Commands by Subsystem

:MEASure:VRMS

(see page 788)

Command Syntax

:MEASure:VRMS [<interval>] [,] [<source>]

<interval> ::= {CYCLe | DISPlay | AUTO}

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

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

The :MEASure:VRMS command installs a screen measurement and starts a dc RMS value measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax


The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. How the dc RMS value is measured depends on the <interval> specification:

- If <interval> is CYCLe, the dc RMS value is measured on an integral number of periods of the displayed signal. If less than three edges are present, the measurement fails, and +9.9E+37 is returned.
- If <interval> is DISPlay, the dc RMS value is measured on all displayed data points.
- If <interval> is AUTO or is not specified, the measurement attempts to compute a value using the CYCLe interval. If less than three edges are present, the measurement is computed using the DISPlay interval.

Return Format

<value><NL>

<value> ::= calculated dc RMS value in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
:MEASure:VTIMe

(see page 788)

Query Syntax
:MEASure:VTIMe? <vtime_argument>[,<source>]

<vtime_argument> ::= time from trigger in seconds
<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH}
<digital channels> ::= DIGital0,..,DIGital15 for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:VTIMe? query returns the value at a specified time on the source specified with :MEASure:SOURce. The specified time must be on the screen and is referenced to the trigger event. If the optional source parameter is specified, the current source is modified.

NOTE
This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

<value><NL>

<value> ::= value at the specified time in NR3 format

See Also
- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:T EDGE" on page 342
- ":MEASure:T VALue" on page 344
:MEASure:VTOP

(see page 788)

Command Syntax
:MEASure:VTOP [<source>]

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

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

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

NOTE
This query is not available if the source is FFT (Fast Fourier Transform).

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

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

Return Format
<value><NL>
<value> ::= vertical value at the top of the waveform in NR3 format

See Also
- "Introduction to :MEASure Commands" on page 312
- ":MEASure:SOURce" on page 337
- ":MEASure:VMAX" on page 349
- ":MEASure:VAMPlitude" on page 346
- ":MEASure:VBASe" on page 348
:MEASURE:WINDOW

Command Syntax
:MEASURE:WINDOW <window>

<window> ::= (MAIN | ZOOM | AUTO)

The :MEASURE:WINDOW command specifies, in the zoomed time base mode, which window is used as the measurement window:

- MAIN — the measurement window is the Main window.
- ZOOM — the measurement window is the Zoom window.
- AUTO — the measurement is attempted in the Zoom window; if it cannot be made there, the Main window is used.

Query Syntax
:MEASURE:WINDOW?

The :MEASURE:WINDOW? query returns the currently specified measurement window.

Return Format
<window><NL>

<window> ::= (MAIN | ZOOM | AUTO)

See Also
- "Introduction to :MEASURE Commands" on page 312
**:MEASure:XMAX**

(see page 788)

**Command Syntax**

:MEASure:XMAX [<source>]

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

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

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

The :MEASure:XMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

**NOTE**

:MEASure:XMAX is an alias for :MEASure:TMAX.

**Query Syntax**

:MEASure:XMAX? [<source>]

The :MEASure:XMAX? query measures and returns the horizontal axis value at which the maximum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

**Return Format**

/value/<NL>

/value> ::= horizontal value of the maximum in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 312
- ":MEASure:XMIN" on page 358
- ":MEASure:TMAX" on page 719
5 Commands by Subsystem

:MEASURE:XMIN

(see page 788)

Command Syntax

:MEASURE:XMIN [<source>]

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

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

The :MEASURE:XMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE

:MEASURE:XMIN is an alias for :MEASURE:TMIN.

Query Syntax

:MEASURE:XMIN? [<source>]

The :MEASURE:XMIN? query measures and returns the horizontal axis value at which the minimum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

<value><NL>

<value> ::= horizontal value of the minimum in NR3 format

See Also

- "Introduction to :MEASURE Commands" on page 312
- ":MEASURE:XMAX" on page 357
- ":MEASURE:TMIN" on page 720
## :MTESt Commands

The MTESt subsystem commands and queries control the mask test features. See "Introduction to :MTESt Commands" on page 361.

### Table 70 :MTESt Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MTESt:AMASK:CREate (see page 364)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
| :MTESt:AMASK:SOURce <source> (see page 365) | :MTESt:AMASK:SOURce? (see page 365) | <source> ::= CHANnel<n>  
<n> ::= {1 | 2 | 3 | 4} for 4ch models  
<n> ::= {1 | 2} for 2ch models  |
| :MTESt:AMASK:UNITs <units> (see page 366) | :MTESt:AMASK:UNITs? (see page 366) | <units> ::= {CURRent | DIVisions}  |
| :MTESt:AMASK:XDELta <value> (see page 367) | :MTESt:AMASK:XDELta? (see page 367) | <value> ::= X delta value in NR3 format  |
| :MTESt:AMASK:YDELta <value> (see page 368) | :MTESt:AMASK:YDELta? (see page 368) | <value> ::= Y delta value in NR3 format  |
| n/a | :MTESt:COUNT:FWAVeforms? [CHANnel<n>] (see page 369) | <failed> ::= number of failed waveforms in NR1 format  |
| :MTESt:COUNT:RESet (see page 370) | n/a | n/a  |
| n/a | :MTESt:COUNT:TIME? (see page 371) | <time> ::= elapsed seconds in NR3 format  |
| n/a | :MTESt:COUNT:WAVeform s? (see page 372) | <count> ::= number of waveforms in NR1 format  |
| :MTESt:DATA <mask> (see page 373) | :MTESt:DATA? (see page 373) | <mask> ::= data in IEEE 488.2 # format  |
| :MTESt:DELete (see page 374) | n/a | n/a  |
| :MTESt:ENABLE {{0 | OFF} | {1 | ON}} (see page 375) | :MTESt:ENABLE? (see page 375) | {0 | 1}  |
| :MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 376) | :MTESt:LOCK? (see page 376) | {0 | 1}  |
### Table 70 :MTESt Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MTESt:OUTPut &lt;signal&gt; (see page 377)</td>
<td>:MTESt:OUTPut? (see page 377)</td>
<td>&lt;signal&gt; ::= {FAIL</td>
</tr>
<tr>
<td>:MTESt:RMODe &lt;rmode&gt; (see page 378)</td>
<td>:MTESt:RMODe? (see page 378)</td>
<td>&lt;rmode&gt; ::= {FOREver</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:MEASure {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:PRINT {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:SAVE {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:FACTion:STOP {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:RMODe:SIGMa &lt;level&gt; (see page 383)</td>
<td>:MTESt:RMODe:SIGMa? (see page 383)</td>
<td>&lt;level&gt; ::= from 0.1 to 9.3 in NR3 format</td>
</tr>
<tr>
<td>:MTESt:RMODe:TIME &lt;seconds&gt; (see page 384)</td>
<td>:MTESt:RMODe:TIME? (see page 384)</td>
<td>&lt;seconds&gt; ::= from 1 to 86400 in NR3 format</td>
</tr>
<tr>
<td>:MTESt:RMODe:WAVeforms &lt;count&gt; (see page 385)</td>
<td>:MTESt:RMODe:WAVeforms? (see page 385)</td>
<td>&lt;count&gt; ::= number of waveforms in NR1 format</td>
</tr>
<tr>
<td>:MTESt:SCALe:BIND {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:MTESt:SCALe:X1 &lt;x1_value&gt; (see page 387)</td>
<td>:MTESt:SCALe:X1? (see page 387)</td>
<td>&lt;x1_value&gt; ::= X1 value in NR3 format</td>
</tr>
<tr>
<td>:MTESt:SCALe:XDELta &lt;xdelta_value&gt; (see page 388)</td>
<td>:MTESt:SCALe:XDELta? (see page 388)</td>
<td>&lt;xdelta_value&gt; ::= X delta value in NR3 format</td>
</tr>
<tr>
<td>:MTESt:SCALe:Y1 &lt;y1_value&gt; (see page 389)</td>
<td>:MTESt:SCALe:Y1? (see page 389)</td>
<td>&lt;y1_value&gt; ::= Y1 value in NR3 format</td>
</tr>
</tbody>
</table>
Introduction to \texttt{: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 \texttt{:MTESt?} to query setup information for the \texttt{MTESt} subsystem.

Return Format

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

\begin{verbatim}
:MTES:SOUR CHAN1;ENAB 0;LOCK 1;:MTES:AMAS:SOUR CHAN1;UNIT DIV;XDEL +2.50000000E-001;YDEL +2.50000000E-001;:MTES:SCAL:e;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
\end{verbatim}

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.

    Sleep 3000

    ' Set up the mask testing parameters...

    ' Record the results...

    ' End of example code.

End Sub
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488
Set myScope.IO = myMgr.Open("TCPIP0::130.29.70.228::inst0::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 "*:MTESTenable " + 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 " :OPERegister: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 " :OPERegister: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:AMASk:CREate

(see page 788)

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 361
- ":\MTESt:AMASk:XDELta" on page 367
- ":\MTESt:AMASk:YDELta" on page 368
- ":\MTESt:AMASk:UNITs" on page 366
- ":\MTESt:AMASk:SOURce" on page 365
- ":\MTESt:SOURce" on page 391

Example Code
- "Example Code" on page 361
:MTESt:AMASk:SOURce

(see page 788)

Command Syntax

: MTESt: AMASk: SOURce <source>

<source> ::= CHAnnel<n>

<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

The :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 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

See Also

- "Introduction to :MTESt Commands" on page 361
- ":MTESt:AMASk:XDELta" on page 367
- ":MTESt:AMASk:YDELta" on page 368
- ":MTESt:AMASk:UNITs" on page 366
- ":MTESt:SOURce" on page 391

Example Code

- "Example Code" on page 361
:MMEMory
(see page 788)

Command Syntax
:MMEMory <memory>

Query Syntax
:MMEMory?

Return Format
<memory>

See Also
- "Introduction to :MMEMory Commands" on page 361
**:MTESt:AMASk:XDELta**

(see page 788)

**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 361
- ":MTESt:AMASk:UNITs" on page 366
- "MTESt:AMASk:YDELta" on page 368
- "MTESt:AMASk:SOURce" on page 365
- "MTESt:SOURce" on page 391

**Example Code**

- "Example Code" on page 361
### :MTEST:AMASK:YDELta

#### Command Syntax

```plaintext
: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

```plaintext
: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

```plaintext
/value/<NL>
```

*<value> ::= Y delta value in NR3 format*

#### See Also

- "Introduction to :MTEST Commands" on page 361
- ":MTEST:AMASK:UNITs" on page 366
- ":MTEST:AMASK:XDELta" on page 367
- ":MTEST:AMASK:SOURce" on page 365
- ":MTEST:SOURce" on page 391

#### Example Code

- "Example Code" on page 361
:MTEST:COUNT:FWAVeforms

(see page 788)

Query Syntax

:MMEMS:COUNT:FWAVeforms? [CHANnel<n>]

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

The :MTEST:COUNT:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms.

Return Format

<failed><NL>

<failed> ::= number of failed waveforms in NR1 format.

See Also

- "Introduction to :MTEST Commands" on page 361
- ":MTEST:COUNT:WAVeforms" on page 372
- ":MTEST:COUNT:TIME" on page 371
- ":MTEST:COUNT:RESet" on page 370
- ":MTEST:SOURce" on page 391

Example Code

- "Example Code" on page 361
:**MTEST:COUNT:RES**

(see page 788)

**Command Syntax**

```
:MTEST:COUNT:RES
```

The :MTEST:COUNT:RES command resets the mask statistics.

**See Also**
- "Introduction to :MTEST Commands" on page 361
- ":MTEST:COUNT:WAVEforms" on page 372
- ":MTEST:COUNT:FWAVEforms" on page 369
- "MTEST:COUNT:TIME" on page 371
:MTES t:COUN t:TIME

(see page 788)

Query Syntax :MTES t:COUN t:TIME?

The :MTES t:COUN t: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 :MTES t Commands" on page 361
- " :MTES t:COUN t:WAveforms" on page 372
- " :MTES t:COUN t:FWAveforms" on page 369
- " :MTES t:COUN t:RESet" on page 370

Example Code
- "Example Code" on page 361
:MTESt:COUNt:WAVEforms

(see page 788)

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 361
- "MTESt:COUNt:FWAVEforms" on page 369
- "MTESt:COUNt:TIME" on page 371
- "MTESt:COUNt:RESet" on page 370

Example Code

- "Example Code" on page 361
:MTEST:DATA

(see page 788)

Command Syntax
: :MTEST:DATA <mask>

<mask> ::= binary block data in IEEE 488.2 # format.

The :MTEST:DATA command loads a mask from binary block data.

Query Syntax
: :MTEST:DATA?

The :MTEST:DATA? query returns a mask in binary block data format. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format
<mask><NL>

<mask> ::= binary block data in IEEE 488.2 # format

See Also
- "SAVE:MASK[:START]" on page 414
- "RECall:MASK[:START]" on page 401
Commands by Subsystem

:MTEST:DELETE

Command Syntax

:MAND:DELETE

The :MTEST:DELETE command clears the currently loaded mask.

See Also

- "Introduction to :MTEST Commands" on page 361
- ":MTEST:AMASK:CREATE" on page 364
:MTST:ENABLe

(see page 788)

Command Syntax
:MTST:ENABLe <on_off>

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

The :MTST:ENABLe command enables or disables the mask test features.

- ON — Enables the mask test features.
- OFF — Disables the mask test features.

Query Syntax
:MTST:ENABLe?

The :MTST:ENABLe? query returns the current state of mask test features.

Return Format
<on_off><NL>

<on_off> ::= {1 | 0}

See Also
- "Introduction to :MTST Commands" on page 361
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.

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 361
- ":MTESt:SOURce" on page 391
:MTEST:OUTPut

N (see page 788)

Command Syntax

: MTEST:OUTPut <signal>

<signal> ::= {FAIL | PASS}

The :MTEST:OUTPut command selects the mask test output condition:

• FAIL — the output occurs when there are mask test failures.
• PASS — the output occurs when the mask test passes.

You can place the mask test signal on the rear panel TRIG OUT BNC using the "CALibrate:OUTPut" on page 211 command.

Query Syntax

: MTEST:OUTPut?

The :MTEST:OUTPut? query returns the currently set output signal.

Return Format

<signal><NL>

<signal> ::= {FAIL | PASS}

See Also

• "Introduction to :MTEST Commands" on page 361
• "CALibrate:OUTPut" on page 211
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 383 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 384 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 385 command.

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 361
- ":MTESt:RMODe:SIGMa" on page 383
- ":MTESt:RMODe:TIME" on page 384
- ":MTESt:RMODe:WAVeforms" on page 385

**Example Code**

- "Example Code" on page 361
:MTESt:RMODe:FACTion:MEASure

(see page 788)

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 361
- ":MTESt:RMODe:FACTion:PRINt" on page 380
- ":MTESt:RMODe:FACTion:SAVE" on page 381
- ":MTESt:RMODe:FACTion:STOP" on page 382
:MTESt:RMODe:FACTion:PRINt

(see page 788)

**Command Syntax**

:MNTESt: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 ":HARDcopy Commands" on page 281 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 361
- ":MTESt:RMODe:FACTion:MEASure" on page 379
- ":MTESt:RMODe:FACTion:SAVE" on page 381
- ":MTESt:RMODe:FACTion:STOP" on page 382
:MTEST:RMODE:FACTion:SAVE

(see page 788)

Command Syntax

:MKST: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


See "SAVE Commands" on page 404 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 361
- "MTEST:RMODE:FACTion:MEASure" on page 379
- "MTEST:RMODE:FACTion:PRINT" on page 380
- "MTEST:RMODE:FACTion:STOP" on page 382
5 Commands by Subsystem

:MMEM:RMODe:FACTion:STOP

(see page 788)

Command Syntax

:MMEM:RMODe:FACTion:STOP <on_off>

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

The :MMEM: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

:MMEM:RMODe:FACTion:STOP?

The :MMEM:RMODe:FACTion:STOP? query returns the current mask failure stop setting.

Return Format

<on_off><NL>

<on_off> ::= {1 | 0}

See Also

- "Introduction to :MMEM Commands" on page 361
- ":MMEM:RMODe:FACTion:MEASure" on page 379
- ":MMEM:RMODe:FACTion:PRINT" on page 380
- ":MMEM:RMODe:FACTion:SAVE" on page 381
**:MTEST:RMODe:SIGMa**

(see page 788)

**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 361
- ":MTEST:RMODe" on page 378

**Example Code**

- "Example Code" on page 361
### :MTESt:RMODe:TIME

(see page 788)

**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 361
- ":MTESt:RMODe" on page 378
**:MTEST:RMODe:WAVEforms**

(see page 788)

**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 361
- ":MTEST:RMODe" on page 378
:MTEST:SCALE:BIND

(see page 788)

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 361
- ":MTEST:SCALE:X1" on page 387
- ":MTEST:SCALE:XDELta" on page 388
- ":MTEST:SCALE:Y1" on page 389
- ":MTEST:SCALE:Y2" on page 390
:MTESt:SCALe:X1

(see page 788)

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:

\[ X = (X \times \Delta X) + X1 \]

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

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 361
- "MTESt:SCALe:BIND" on page 386
- "MTESt:SCALe:XDELta" on page 388
- "MTESt:SCALe:Y1" on page 389
- "MTESt:SCALe:Y2" on page 390
Command Syntax

:MTES:SCAL:XDELta <xdelta_value>

<xdelta_value> ::= X delta value in NR3 format

The :MTES:SCAL:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then, a change in data rate without corresponding changes in the waveform can easily be handled by changing ΔX.

The X-coordinate of polygon vertices is normalized using this equation:

$$X = (X \times \Delta X) + X1$$

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

:MTES:SCAL:XDELta?

The :MTES:SCAL: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 :MTES Commands" on page 361
- ":MTES:SCAL:BIND" on page 386
- ":MTES:SCAL:X1" on page 387
- ":MTES:SCAL:Y1" on page 389
- ":MTES:SCAL:Y2" on page 390
:MTES:MST:SCAL:e:Y1

(see page 788)

Command Syntax

:MTES:MST:SCAL:e:Y1 <y1_value>

<y1_value> ::= Y1 value in NR3 format

The :MTES:MST:SCAL:e: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 SCAL:e:Y1 and SCAL:e: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

:MTES:MST:SCAL:e:Y1?

The :MTES:MST:SCAL:e:Y1? query returns the current setting of the Y1 marker.

Return Format

<y1_value>

<y1_value> ::= Y1 value in NR3 format

See Also

- "Introduction to :MTES Commands" on page 361
- ":MTES:MST:SCAL:e:BIND" on page 386
- ":MTES:MST:SCAL:e:X1" on page 387
- ":MTES:MST:SCAL:e:XDELta" on page 388
- ":MTES:MST:SCAL:e:Y2" on page 390
**:MTEST:SCALE:Y2**

(see page 788)

**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:

\[ Y = (Y \times (Y2 - Y1)) + Y1 \]

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

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 361
- ":MTEST:SCALE:BIND" on page 386
- ":MTEST:SCALE:X1" on page 387
- ":MTEST:SCALE:XDELta" on page 388
- ":MTEST:SCALE:Y1" on page 389
**:MTEST:SOURce**

(see page 788)

**Command Syntax**

: :MTEST:SOURce <source>

<source> ::= CHANnel<n>

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

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

The :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 | 2 | 3 | 4} for the four channel oscilloscope models

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

**See Also**

- "Introduction to :MTEST Commands" on page 361
- ":MTEST:AMASK:SOURce" on page 365
Commands by Subsystem

:MTEST:TITLE

(see page 788)

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 361
:POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "Introduction to :POD<n> Commands" on page 393.

Table 71 :POD<n> Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:POD&lt;n&gt;:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:POD&lt;n&gt;:SIZE &lt;value&gt; (see page 395)</td>
<td>:POD&lt;n&gt;:SIZE? (see page 395)</td>
<td>&lt;value&gt; ::= {SMAL1</td>
</tr>
<tr>
<td>:POD&lt;n&gt;:THReshold &lt;type&gt;[suffix] (see page 396)</td>
<td>:POD&lt;n&gt;:THReshold? (see page 396)</td>
<td>&lt;n&gt; ::= 1-2 in NR1 format &lt;type&gt; ::= {CMOS</td>
</tr>
</tbody>
</table>

Introduction to :POD<n> Commands

<n> ::= {1 | 2}.

The POD subsystem commands control the viewing and threshold of groups of digital channels.

POD1 ::= D0-D7
POD2 ::= D8-D15

These commands are only valid for the MSO models.

NOTE

Reporting the Setup

Use :POD1? or :POD2? to query setup information for the POD subsystem.

Return Format

The following is a sample response from the :POD1? query. In this case, the query was issued following a *RST command.

:POD1:DISP 0;THR +1.40E+00
**:POD<n>:DISPlay**

(see page 788)

**Command Syntax**

:POD<n>:DISPlay <display>

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

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

POD1 ::= D0-D7
POD2 ::= D8-D15

The :POD<n>:DISPlay command turns displaying of the specified group of channels on or off.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax**

:POD<n>:DISPlay?

The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

**Return Format**

<display><NL>

<display> ::= {0 | 1}

**See Also**

- "Introduction to :POD<n> Commands" on page 393
- "DIGital<n>:DISPlay" on page 239
- "CHANnel<n>:DISPlay" on page 222
- "VIEW" on page 180
- "BLANk" on page 148
- "STATus" on page 177
**:POD<n>:SIZE**

(see page 788)

**Command Syntax**

`:POD<n>:SIZE <value>`

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

POD1 ::= D0-D7
POD2 ::= D8-D15

<value> ::= {SMALL | MEDium | LARGe}

The :POD<n>:SIZE command specifies the size of digital channels on the display.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax**

`:POD<n>:SIZE?`

The :POD<n>:SIZE? query returns the size setting for the specified group of channels.

**Return Format**

<size_value><NL>

<size_value> ::= {SMAL | MED | LARG}

**See Also**

- "Introduction to :POD<n> Commands" on page 393
- ":DIGital<n>:SIZE" on page 242
- ":DIGital<n>:POSition" on page 241
5 Commands by Subsystem

:POD<n>:THReshold

N (see page 788)

Command Syntax

:POD<n>:THReshold <type>[<suffix>]

<n> ::= An integer, 1 or 2, is attached as a suffix to the command and defines the group of channels that are affected by the command.
<type> ::= {CMOS | ECL | TTL | <user defined value>}
<user defined value> ::= -8.00 to +8.00 in NR3 format
<suffix> ::= {V | mV | uV}
POD1 ::= D0-D7
POD2 ::= D8-D15
TTL ::= 1.4V
CMOS ::= 2.5V
ECL ::= -1.3V

The :POD<n>:THReshold command sets the threshold for the specified group of channels. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE

This command is only valid for the MSO models.

Query Syntax

:POD<n>:THReshold?

The :POD<n>:THReshold? query returns the threshold value for the specified group of channels.

Return Format

<threshold><NL>

<threshold> ::= Floating point number in NR3 format

See Also

- "Introduction to :POD<n> Commands" on page 393
- ":DIGital<n>:THReshold" on page 243
- ":TRIGger[:EDGE]:LEVel" on page 504

Example Code

' THRESHOLD - This command is used to set the voltage threshold for the waveforms. There are three preset values (TTL, CMOS, and ECL) and you can also set a user-defined threshold value between -8.0 volts and +8.0 volts.
'
' In this example, we set channels 0-7 to CMOS, then set channels 8-15 to a user-defined 2.0 volts, and then set the external trigger to TTL. Of course, you only need to set the thresholds for the channels you will be using in your program.'
' Set channels 0-7 to CMOS threshold.
myScope.WriteString "POD1:THRESHOLD CMOS"

' Set channels 8-15 to 2.0 volts.
myScope.WriteString "POD2:THRESHOLD 2.0"

' Set external channel to TTL threshold (short form).
myScope.WriteString "TRIG:LEV TTL,EXT"

Example program from the start: "VISA COM Example in Visual Basic" on page 814
### :RECall Commands

Recall previously saved oscilloscope setups and traces. See "Introduction to :RECall Commands" on page 398.

#### Table 72 :RECall Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:RECall:FILename &lt;base_name&gt; (see page 399)</td>
<td>:RECall:FILename? (see page 399)</td>
<td>&lt;base_name&gt;: := quoted ASCII string</td>
</tr>
<tr>
<td>:RECall:IMAGe[:STARt] [&lt;file_spec&gt;] (see page 400)</td>
<td>n/a</td>
<td>&lt;file_spec&gt;: := {&lt;internal_loc&gt;</td>
</tr>
<tr>
<td>:RECall:MASK[:STARt] [&lt;file_spec&gt;] (see page 401)</td>
<td>n/a</td>
<td>&lt;file_spec&gt;: := {&lt;internal_loc&gt;</td>
</tr>
<tr>
<td>:RECall:PWD &lt;path_name&gt; (see page 402)</td>
<td>:RECall:PWD? (see page 402)</td>
<td>&lt;path_name&gt;: := quoted ASCII string</td>
</tr>
<tr>
<td>:RECall:SETup[:STARt] [&lt;file_spec&gt;] (see page 403)</td>
<td>n/a</td>
<td>&lt;file_spec&gt;: := {&lt;internal_loc&gt;</td>
</tr>
</tbody>
</table>

#### Introduction to :RECall Commands

The :RECall subsystem provides commands to recall previously saved oscilloscope setups and traces.

**Reporting the Setup**

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

**Return Format**

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

:REC:FIL "scope_0"
**:RECall:FILename**

(see page 788)

**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 398
- ":RECall:IMAGe[:STARt]" on page 400
- ":RECall:SETup[:STARt]" on page 403
- ":SAVE:FILename" on page 406
Commands by Subsystem

:RECall:IMAGE[:STARt]

(see page 788)

Command Syntax

:RECall:IMAGE[:STARt] [<file_spec>]

<file_spec> ::= (<internal_loc> | <file_name>)

<internal_loc> ::= 0-9; an integer in NR1 format

<file_name> ::= quoted ASCII string

The :RECall:IMAGE[:STARt] command recalls a trace (TIFF) image.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".tif".

See Also

- "Introduction to :RECall Commands" on page 398
- ":RECall:FILENAME" on page 399
- ":SAVE:IMAGE[:STARt]" on page 407
** :RECall:MASK[:STArt] **

(see page 788)

**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 398
- ":RECall:FILENAME" on page 399
- ":SAVE:MASK[:STArt]" on page 414
- ":MTESt:DATA" on page 373
5 Commands by Subsystem

:RECall:PWD

(see page 788)

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 398
- "SAVE:PWD" on page 415
**:RECall:SETup[:STARt]**

(see page 788)

**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 398
- ":RECall:FILename" on page 399
- ":SAVE:SETup[:STARt]" on page 416
### :SAVE Commands

Save oscilloscope setups and traces, screen images, and data. See "Introduction to :SAVE Commands" on page 405.

**Table 73** :SAVE Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SAVE:FILename &lt;base_name&gt;</td>
<td>:SAVE:FILename? (see page 406)</td>
<td>&lt;base_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>:SAVE:IMAGe[:STARt] [&lt;file_spec&gt;]</td>
<td>n/a</td>
<td>&lt;file_spec&gt; ::= {&lt;internal_loc&gt;</td>
</tr>
<tr>
<td>n/a</td>
<td>:SAVE:IMAGe:AREA? (see page 408)</td>
<td>&lt;area&gt; ::= {GRAT</td>
</tr>
<tr>
<td>:SAVE:IMAGe:FACTors {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:SAVE:IMAGe:FORMAT &lt;format&gt;</td>
<td>:SAVE:IMAGe:FORMAT? (see page 410)</td>
<td>&lt;format&gt; ::= {TIFF</td>
</tr>
<tr>
<td>:SAVE:IMAGe:INKSaver {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:SAVE:IMAGe:PALette &lt;palette&gt;</td>
<td>:SAVE:IMAGe:PALette? (see page 412)</td>
<td>&lt;palette&gt; ::= {COlor</td>
</tr>
<tr>
<td>n/a</td>
<td>:SAVE:LISTer[:STARt] [&lt;file_name&gt;]</td>
<td>&lt;file_name&gt; ::= quoted ASCII string</td>
</tr>
<tr>
<td>n/a</td>
<td>:SAVE:MASK[:STARt] [&lt;file_spec&gt;]</td>
<td>&lt;file_spec&gt; ::= {&lt;internal_loc&gt;</td>
</tr>
<tr>
<td>:SAVE:PWD &lt;path_name&gt;</td>
<td>:SAVE:PWD? (see page 415)</td>
<td>&lt;path_name&gt; ::= quoted ASCII string</td>
</tr>
</tbody>
</table>
Introduction to :SAVE Commands

The :SAVE subsystem provides commands to save oscilloscope setups and traces, screen images, and data.

:SAV is an acceptable short form for :SAVE.

Reporting the Setup

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

Return Format

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

:SAVE:FIL ""; :SAVE:IMAG:AREA GRAT; FACT 0; FORM TIFF; INKS 0; PAL
MON; :SAVE:PWD "C:/setups/"; :SAVE:WAV:FORM NONE; LENG 1000; SEGM CURR

---

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :SAVE:SETup[:STARt] [<file_spec>] (see page 416) | 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 417) | n/a | <file_name> ::= quoted ASCII string |
| :SAVE:WAVeform:FORMat <format> (see page 418) | :SAVE:WAVeform:FORMat ? (see page 418) | <format> ::= {ALB | ASCIIxy | CSV | BINary | NONE} |
| :SAVE:WAVeform:LENGth <length> (see page 419) | :SAVE:WAVeform:LENGth ? (see page 419) | <length> ::= 100 to max. length; an integer in NR1 format |
| :SAVE:WAVeform:SEGMented <option> (see page 420) | :SAVE:WAVeform:SEGMented? (see page 420) | <option> ::= {ALL | CURRent} |
### :SAVE:FILename

(see page 788)

**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 405
- ":SAVE:IMAGe[:STARt]" on page 407
- ":SAVE:SETup[:STARt]" on page 416
- ":SAVE:WAVEform[:STARt]" on page 417
- ":SAVE:PWD" on page 415
- ":RECall:FILename" on page 399
:SAVE:IMAGe[:STARt]

(see page 788)

Command Syntax

:SAVE:IMAGe[:STARt] [<file_spec>]

<file_spec> ::= (<internal_loc> | <file_name>)

<internal_loc> ::= 0-9; an integer in NR1 format

<file_name> ::= quoted ASCII string

The :SAVE:IMAGe[:STARt] command saves an image.

**NOTE**

If a file extension is provided as part of a specified <file_name>, 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.

**NOTE**

When the <internal_loc> option is used, the :SAVE:IMAGe:FORMat will be changed to TIFF.

See Also

- "Introduction to :SAVE Commands" on page 405
- ":SAVE:IMAGe:AREA" on page 408
- ":SAVE:IMAGe:FACTors" on page 409
- ":SAVE:IMAGe:FORMat" on page 410
- ":SAVE:IMAGe:INKSaver" on page 411
- ":SAVE:IMAGe:PAlette" on page 412
- ":SAVE:FILename" on page 406
- ":RECall:IMAGe[:STARt]" on page 400
**:SAVE:IMAGE:AREA**

(see page 788)

**Query Syntax**

`:SAVE:IMAGE:AREA?`

The `:SAVE:IMAGE:AREA?` query returns the selected image area. If the `:SAVE:IMAGE:FORMAT` is TIFF, the area is GRAT (graticule). Otherwise, it is SCR (screen).

**Return Format**

```
<area><NL>
<area> ::= {GRAT | SCR}
```

**See Also**

- "Introduction to :SAVE Commands" on page 405
- ":SAVE:IMAGE[:START]" on page 407
- ":SAVE:IMAGE:FACTors" on page 409
- ":SAVE:IMAGE:FORMAT" on page 410
- ":SAVE:IMAGE:INKSaver" on page 411
- ":SAVE:IMAGE:PALETTE" on page 412
:SAVE:IMAGE:FACTors

(see page 788)

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 405
- ":SAVE:IMAGE[:STARt]" on page 407
- ":SAVE:IMAGE:AREA" on page 408
- ":SAVE:IMAGE:FORMat" on page 410
- ":SAVE:IMAGE:INKSaver" on page 411
- ":SAVE:IMAGE:PALette" on page 412
**SAVE:IMAGe:FORMat**

![Image](image.png) (see page 788)

**Command Syntax**

`:SAVE:IMAGe:FORMat <format>`

<format> ::= {TIFF | {BMP | BMP24bit} | BMP8bit | PNG}

The `:SAVE:IMAGe:FORMat` command sets the image format type.

**Query Syntax**

`:SAVE:IMAGe:FORMat?`

The `:SAVE:IMAGe:FORMat?` query returns the selected image format type.

**Return Format**

<format><NL>

<format> ::= {TIFF | BMP | BMP8 | PNG | NONE}

When NONE is returned, it indicates that a waveform data file format is currently selected.

**See Also**

- "Introduction to :SAVE Commands" on page 405
- "SAVE:IMAgE[:STArt]" on page 407
- "SAVE:IMAgE:AREA" on page 408
- "SAVE:IMAgE:FACTors" on page 409
- "SAVE:IMAgE:INKSaver" on page 411
- "SAVE:IMAgE:PALette" on page 412
- "SAVE:WAVEform:FORMat" on page 418
**:SAVE:IMAGe:INKSaver**

(see page 788)

**Command Syntax**

```plaintext
: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**

```plaintext
: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 405
- ":SAVE:IMAGe[:STARt]" on page 407
- ":SAVE:IMAGe:AREA" on page 408
- ":SAVE:IMAGe:FACTors" on page 409
- ":SAVE:IMAGe:FORMAT" on page 410
- ":SAVE:IMAGe:PAlette" on page 412
:SAVE:IMAGE:PAlette

(see page 788)

Command Syntax

:SAVE:IMAGE:PAlette <palette>

<palette> ::= {COlor | GRAYscale | MONochrome}

The :SAVE:IMAGE:PAlette command sets the image palette color.

NOTE

MONochrome is the only valid choice when the :SAVE:IMAGE:FORMat is TIFF. COLor and GRAYscale are the only valid choices when the format is not TIFF.

Query Syntax

:SAVE:IMAGE:PAlette?

The :SAVE:IMAGE:PAlette? query returns the selected image palette color.

Return Format

<palette><NL>

<palette> ::= {COL | GRAY | MON}

See Also

- "Introduction to :SAVE Commands" on page 405
- ":SAVE:IMAGE[:STARt]" on page 407
- ":SAVE:IMAGE:AREA" on page 408
- ":SAVE:IMAGE:FACTors" on page 409
- ":SAVE:IMAGE:FORMat" on page 410
- ":SAVE:IMAGE:INKSaver" on page 411
**:SAVE:LISTer[:STARt]**

(see page 788)

**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 405
- ":SAVE:FILename" on page 406
- ":LISTer Commands" on page 292

**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 405
- ":SAVE:FILENAME" on page 406
- ":RECall:MASK[:START]" on page 401
- ":MTESt:DATA" on page 373
:SAVE:PWD

(see page 788)

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 405
- ":SAVE:FILENAME" on page 406
- ":RECALL:PWD" on page 402
5 Commands by Subsystem

:SAVE:SETup[:STARt]

(see page 788)

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 405
- ":SAVE:FILename" on page 406
- ":RECall:SETup[:STARt]" on page 403
**:SAVE:WAVeform[:STARt]**

(see page 788)

**Command Syntax**

`:SAVE:WAVeform[:STARt] [<file_name>]

<file_name> ::= quoted ASCII string

The :SAVE:WAVeform[:STARt] command saves oscilloscope waveform data to a file.

**NOTE**

If a file extension is provided as part of a specified <file_name>, 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 405
- ":SAVE:WAVeform:FORMat" on page 418
- ":SAVE:WAVeform:LENGth" on page 419
- ":SAVE:FILename" on page 406
- ":RECall:SETup[:STARt]" on page 403
:SAVE:WAVEform:FORMat

(see page 788)

Command Syntax

:SAVE:WAVEform:FORMat <format>

=format> ::= (ALB | ASCiixy | CSV | BINary)

The :SAVE:WAVEform:FORMat command sets the waveform data format type:

- ALB — creates an Agilent module binary format file. These files can be viewed offline by the Agilent Logic Analyzer application software. The proper file extension for this format is "alb".
- ASCiixy — creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is "csv".
- CSV — creates one comma-separated value file that contains information for all analog channels that are displayed (turned on). The proper file extension for this format is "csv".
- BINary — creates an oscilloscope binary data format file. See the User's Guide for a description of this format. The proper file extension for this format is "bin".

Query Syntax

:SAVE:WAVEform:FORMat?

The :SAVE:WAVEform:FORMat? query returns the selected waveform data format type.

Return Format

=format><NL

=format> ::= (ALB | ASC | CSV | BIN | NONE)

When NONE is returned, it indicates that an image file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 405
- ":SAVE:WAVEform[:STARt]" on page 417
- ":SAVE:WAVEform:LENGTH" on page 419
- ":SAVE:IMAGe:FORMat" on page 410
The :SAVE:WAVeform:LENGth command sets the waveform data length (that is, the number of points saved).

The :SAVE:WAVeform:LENGth? query returns the specified waveform data length.

See Also
- "Introduction to :SAVE Commands" on page 405
- ":SAVE:WAVeform[:START]" on page 417
- ":WAVeform:POINts" on page 632
- ":SAVE:WAVeform:FORMat" on page 418
**Command Syntax**

`:SAVE:WAVeform:SEGmented <option>`

```
<option> ::= {ALL | CURRent}
```

When segmented memory is used for acquisitions, the `:SAVE:WAVeform:SEGメントed` command specifies which segments are included when the waveform is saved:

- **ALL** — all acquired segments are saved.
- **CURRent** — only the currently selected segment is saved.

**Query Syntax**

`:SAVE:WAVeform:SEGmented?`

The `:SAVE:WAVeform:SEGメントed?` query returns the current segmented waveform save option setting.

**Return Format**

```
<option><NL>
```

```
<option> ::= {ALL | CURR}
```

**See Also**

- "Introduction to :SAVE Commands" on page 405
- "`:SAVE:WAVeform[:START]" on page 417
- "`:SAVE:WAVeform:FORMat" on page 418
- "`:SAVE:WAVeform:LENGth" on page 419
### :SBUS Commands

Control oscilloscope functions associated with the serial decode bus. See "Introduction to :SBUS Commands" on page 422.

**Table 74 :SBUS Commands Summary**

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNt:ERRor? (see page 423)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNt:OVERl oad? (see page 424)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:COUNt:RESet (see page 425)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNt:TOTal? (see page 426)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:CAN:COUNt:UTILi zation? (see page 427)</td>
<td>&lt;percent&gt; ::= floating-point in NR3 format</td>
</tr>
<tr>
<td>:SBUS:DISPlay {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:FLEXray:COUNt:N ULL? (see page 429)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:FLEXray:COUNt:RESet (see page 430)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:FLEXray:COUNt:SYNC? (see page 431)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:FLEXray:COUNt:TOTal? (see page 432)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:I2S:BASE &lt;base&gt; (see page 433)</td>
<td>:SBUS:I2S:BASE? (see page 433)</td>
<td>&lt;base&gt; ::= {DECimal</td>
</tr>
<tr>
<td>:SBUS:IIC:ASIZe &lt;size&gt; (see page 434)</td>
<td>:SBUS:IIC:ASIZe? (see page 434)</td>
<td>&lt;size&gt; ::= {BIT7</td>
</tr>
<tr>
<td>:SBUS:LIN:PARity {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:SBUS:M1553:BASE &lt;base&gt; (see page 436)</td>
<td>:SBUS:M1553:BASE? (see page 436)</td>
<td>&lt;base&gt; ::= {DECimal</td>
</tr>
<tr>
<td>:SBUS:MODE &lt;mode&gt; (see page 437)</td>
<td>:SBUS:MODE? (see page 437)</td>
<td>&lt;mode&gt; ::= {CAN</td>
</tr>
</tbody>
</table>
### Introduction to :SBUS Commands

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

These commands are only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

### Table 74: :SBUS Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SBUS:SPI:BITorder &lt;order&gt; (see page 438)</td>
<td>:SBUS:SPI:BITorder? (see page 438)</td>
<td>&lt;order&gt; ::= {LSBFirst</td>
</tr>
<tr>
<td>:SBUS:SPI:WIDTH &lt;word_width&gt; (see page 439)</td>
<td>:SBUS:SPI:WIDTH? (see page 439)</td>
<td>&lt;word_width&gt; ::= integer 4-16 in NR1 format</td>
</tr>
<tr>
<td>:SBUS:UART:BASE &lt;base&gt; (see page 440)</td>
<td>:SBUS:UART:BASE? (see page 440)</td>
<td>&lt;base&gt; ::= {ASCii</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:UART:COUNT:ERROR? (see page 441)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:UART:COUNT:RESET (see page 442)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:UART:COUNT:RXFRAMES? (see page 443)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:SBUS:UART:COUNT:TXFRAMES? (see page 444)</td>
<td>&lt;frame_count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:SBUS:UART:FRAMING &lt;value&gt; (see page 445)</td>
<td>:SBUS:UART:FRAMING? (see page 445)</td>
<td>&lt;value&gt; ::= {OFF</td>
</tr>
</tbody>
</table>

### Reporting the Setup

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

### Return Format

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

:SBUS:DISP 0;MODE IIC
**:SBUS:CAN:COUNt:ERRor**

(see page 788)

**Query Syntax**

**:SBUS:CAN:COUNt:ERRor?**

Returns the error frame count.

**Return Format**

\[<\text{frame\_count}><\text{NL}>\]

\[<\text{frame\_count}> ::= \text{integer in NR1 format}\]

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- ":SBUS:CAN:COUNt:RESet" on page 425
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:CAN Commands" on page 480
**:SBUS:CAN:COUNT:OVERload**

(see page 788)

**Query Syntax**

:SBUS:CAN:COUNT:OVERload?

Returns the overload frame count.

**Return Format**

<frame_count><NL>

<frame_count> ::= integer in NR1 format

**Errors**

- `-241, Hardware missing` on page 747

**See Also**

- ":SBUS:CAN:COUNT:RESet" on page 425
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:CAN Commands" on page 480
:SBUS:CAN:COUNt:RESet

Resets the frame counters.

Command Syntax

:SBUS:CAN:COUNt:RESet

Errors

- "-241, Hardware missing" on page 747

See Also

- ":SBUS:CAN:COUNt:ERRor" on page 423
- ":SBUS:CAN:COUNt:OVERload" on page 424
- ":SBUS:CAN:COUNt:TOTal" on page 426
- ":SBUS:CAN:COUNt:UTILization" on page 427
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:CAN Commands" on page 480
:\SBUS:CAN:COUNt:TOTal

N (see page 788)

Query Syntax
:\SBUS:CAN:COUNt:TOTal?

Returns the total frame count.

Return Format
<frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors
• "-241, Hardware missing" on page 747

See Also
• ":SBUS:CAN:COUNt:RESet" on page 425
• "Introduction to :SBUS Commands" on page 422
• ":SBUS:MODE" on page 437
• ":TRIGger:CAN Commands" on page 480
:SBUS:CAN:COUNt:UTILization

(see page 788)

**Query Syntax**

:SBUS:CAN:COUNt:UTILization?

Returns the percent utilization.

**Return Format**

<percent><NL>

<percent> ::= floating-point in NR3 format

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- ":SBUS:CAN:COUNt:RESet" on page 425
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:CAN Commands" on page 480
**:SBUS:DISPlay**

(see page 788)

**Command Syntax**

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

The :SBUS:DISPlay command turns displaying of the serial decode bus on or off.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

**Query Syntax**

```
:SBUS:DISPlay?
```

The :SBUS:DISPlay? query returns the current display setting of the serial decode bus.

**Return Format**

```
<display><NL>
<display> ::= {0 | 1}
```

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- "Introduction to :SBUS Commands" on page 422
- ":CHANnel<n>:DISPlay" on page 222
- ":DIGital<n>:DISPlay" on page 239
- ":POD<n>:DISPlay" on page 394
- ":VIEW" on page 180
- ":BLANk" on page 148
- ":STATus" on page 177
:SBUS:FLEXray:COUNt:NULL

(see page 788)

Query Syntax
:SBUS:FLEXray:COUNt:NULL?

Returns the FlexRay null frame count.

Return Format
<frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors
- "-241, Hardware missing" on page 747

See Also
- ":SBUS:FLEXray:COUNt:RESet" on page 430
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:FLEXray Commands" on page 508
:SBUS:FLEXray:COUNt:RESet

Resets the FlexRay frame counters.

Errors

- "-241, Hardware missing" on page 747

See Also

- "SBUS:FLEXray:COUNt:NULL" on page 429
- "SBUS:FLEXray:COUNt:SYNC" on page 431
- "SBUS:FLEXray:COUNt:TOTal" on page 432
- "Introduction to :SBUS Commands" on page 422
- "SBUS:MODE" on page 437
- "TRIGger:FLEXray Commands" on page 508
**:SBUS:FLEXray:COUNt:SYNC**

N (see page 788)

**Query Syntax**

**:SBUS:FLEXray:COUNt:SYNC?**

Returns the FlexRay sync frame count.

**Return Format**

<frame_count><NL>

<frame_count> ::= integer in NR1 format

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- ":SBUS:FLEXray:COUNt:RESet" on page 430
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:FLEXray Commands" on page 508
:SBUS:FLEXray:COUNt:TOTal

N (see page 788)

Query Syntax

:SBUS:FLEXray:COUNt:TOTal?

Returns the FlexRay total frame count.

Return Format

<frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors

• "-241, Hardware missing" on page 747

See Also

• ":SBUS:FLEXray:COUNt:RESet" on page 430
• "Introduction to :SBUS Commands" on page 422
• ":SBUS:MODE" on page 437
• ":TRIGger:FLEXray Commands" on page 508
**:SBUS:I2S:BASE**

N  (see page 788)

**Command Syntax**

:SBUS:I2S:BASE <base>

<base> ::= {DECimal | HEX}

The :SBUS:I2S:BASE command determines the base to use for the I2S decode display.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the I2S serial decode option (Option SND) has been licensed.

---

**Query Syntax**

:SBUS:I2S:BASE?

The :SBUS:I2S:BASE? query returns the current I2S display decode base.

**Return Format**

<base><NL>

<base> ::= {DECimal | HEX}

**Errors**

- "241, Hardware missing" on page 747

**See Also**

- "Introduction to :SBUS Commands" on page 422
- "TRIGger:I2S Commands" on page 529
Commands by Subsystem

:SBUS: IIC: ASIZe

(see page 788)

Command Syntax :

:SBUS: IIC: ASIZe <size>

<size> ::= {BIT7 | BIT8}

The :SBUS: IIC: ASIZe command determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.

NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax :

:SBUS: IIC: ASIZe?

The :SBUS: IIC: ASIZe? query returns the current IIC address width setting.

Return Format <mode><NL>

<mode> ::= {BIT7 | BIT8}

Errors

- "-241, Hardware missing" on page 747

See Also

- "Introduction to :SBUS Commands" on page 422
- ":TRIGger: IIC Commands" on page 547
Command Syntax

:SBUS:LIN:PARity <display>

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

The :SBUS:LIN:PARity command determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.

Note

This command is only valid on 4 (analog) channel oscilloscope models when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

:SBUS:LIN:PARity?

The :SBUS:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

Return Format

<display><NL>

<display> ::= {0 | 1}

Errors

- "241, Hardware missing" on page 747

See Also

- "Introduction to :SBUS Commands" on page 422
- "TRIGger:LIN Commands" on page 556
Commands by Subsystem

:SBUS:M1553:BASE

Command Syntax
:SBUS:M1553:BASE <base>

<base> ::= {BINary | HEX}

The :SBUS:M1553:BASE command determines the base to use for the MIL-STD 1553 decode display.

NOTE
This command is only valid on 4 (analog) channel oscilloscope models when the MIL-STD 1553 serial decode option (Option 553) has been licensed.

Query Syntax
:SBUS:M1553:BASE?


Return Format
<base><NL>

<base> ::= {BIN | HEX}

Errors
- "-241, Hardware missing" on page 747

See Also
- "Introduction to :SBUS Commands" on page 422
- ":TRIGger:M1553 Commands" on page 569
:**SBUS:MODE**

(see page 788)

**Command Syntax**

:SBUS:MODE <mode>

<mode> ::= {CAN | FLEXray | I2S | IIC | LIN | M1553 | SPI | UART}

The :SBUS:MODE command determines the decode mode for the serial bus.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

**Query Syntax**

:SBUS:MODE?

The :SBUS:MODE? query returns the current serial bus decode mode setting.

**Return Format**

<mode><NL>

<mode> ::= {CAN | FLEX | I2S | IIC | LIN | M1553 | SPI | UART | NONE}

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- "Introduction to :SBUS Commands" on page 422
- ".TRIGger:MODE" on page 475
- ".TRIGger:CAN Commands" on page 480
- ".TRIGger:FLEXray Commands" on page 508
- ".TRIGger:I2S Commands" on page 529
- ".TRIGger:IIC Commands" on page 547
- ".TRIGger:LIN Commands" on page 556
- ".TRIGger:M1553 Commands" on page 569
- ".TRIGger:SPI Commands" on page 584
- ".TRIGger:UART Commands" on page 599
:SBUS:SPI:BITorder

Command Syntax

:SBUS:SPI:BITorder <order>

<order> ::= {LSBFirst | MSBFirst}

The :SBUS: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.

NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax

:SBUS:SPI:BITorder?

The :SBUS:SPI:BITorder? query returns the current SPI decode bit order.

Return Format

<order><NL>

<order> ::= {LSBF | MSBF}

Errors

- "-241, Hardware missing" on page 747

See Also

- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:SPI Commands" on page 584
:SBUS:SPI:WIDTh

(see page 788)

Command Syntax

:SBUS:SPI:WIDTh <word_width>

<word_width> ::= integer 4-16 in NR1 format

The :SBUS:SPI:WIDTh command determines the number of bits in a word of data for SPI.

NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax

:SBUS:SPI:WIDTh?

The :SBUS:SPI:WIDTh? query returns the current SPI decode word width.

Return Format

<word_width><NL>

<word_width> ::= integer 4-16 in NR1 format

Errors

- "-241, Hardware missing" on page 747

See Also

- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:SPI Commands" on page 584
**:SBUS:UART:BASE**

(see page 788)

**Command Syntax**

`:SBUS:UART:BASE <base>`

<base> ::= {ASCii | BINary | HEX}

The :SBUS:UART:BASE command determines the base to use for the UART decode display.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Query Syntax**

`:SBUS:UART:BASE?`

The :SBUS:UART:BASE? query returns the current UART decode base setting.

**Return Format**

<base><NL>

<base> ::= {ASCii | BINary | HEX}

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- "Introduction to :SBUS Commands" on page 422
- ":.TRIGger:UART Commands" on page 599
**:SBUS:UART:COUNT:ERROR**

Query Syntax

```
:SBUS:UART:COUNT:ERROR?
```

Returns the UART error frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Return Format

```
<frame_count><NL>
```

<frame_count> ::= integer in NR1 format

Errors

- "-241, Hardware missing" on page 747

See Also

- ":SBUS:UART:COUNT:RESet" on page 442
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:UART Commands" on page 599
### :SBUS:UART:COUNT:RESet

#### Command Syntax

```plaintext
:SBUS:UART:COUNT:RESet
```

Resets the UART frame counters.

#### NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

#### Errors

- "-241, Hardware missing" on page 747

#### See Also

- ":SBUS:UART:COUNT:ERROR" on page 441
- ":SBUS:UART:COUNT:RXFRAMES" on page 443
- ":SBUS:UART:COUNT:TXFRAMES" on page 444
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGGER:UART Commands" on page 599
:SBUS:UART:COUNt:RXFRames

(see page 788)

Query Syntax

:SBUS:UART:COUNt:RXFRames?

Returns the UART Rx frame count.

NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Return Format

<frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors

- "-241, Hardware missing" on page 747

See Also

- ":SBUS:UART:COUNt:RESet" on page 442
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:UART Commands" on page 599
**:SBUS:UART:COUNt:TXFRI\text{rames}**

(see page 788)

**Query Syntax**

**:SBUS:UART:COUNt:TXFRI\text{rames}?**

Returns the UART Tx frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format**

<frame_count><NL>

<frame_count> ::= integer in NR1 format

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- ":SBUS:UART:COUNt:RE\text{Set}" on page 442
- "Introduction to :SBUS Commands" on page 422
- ":SBUS:MODE" on page 437
- ":TRIGger:UART Commands" on page 599
:**SBUS:UART:FRAMing**

(see page 788)

**Command Syntax**

:SBUS:UART:FRAMing <value>

<value> ::= (OFF | <decimal> | <nondecimal>)

<decimal> ::= 8-bit integer in decimal from 0-255 (0x00-0xff)

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

<nondecimal> ::= #Bnn...n where n ::= (0 | 1) for binary

The :SBUS:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.

**Query Syntax**

:SBUS:UART:FRAMing?

The :SBUS:UART:FRAMing? query returns the current UART decode base setting.

**Return Format**

<value><NL>

<value> ::= (OFF | <decimal>)

<decimal> ::= 8-bit integer in decimal from 0-255

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- "Introduction to :SBUS Commands" on page 422
- ":TRIGger:UART Commands" on page 599

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.
Commands by Subsystem

:SYSTem Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTem Commands" on page 446.

Table 75 :SYSTem Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SYSTem:DATE &lt;date&gt; (see page 447)</td>
<td>:SYSTem:DATE? (see page 447)</td>
<td>&lt;date&gt; ::= &lt;year&gt;,&lt;month&gt;,&lt;day&gt; &lt;year&gt; ::= 4-digit year in NR1 format &lt;month&gt; ::= {1,...,12</td>
</tr>
<tr>
<td>:SYSTem:DSP &lt;string&gt; (see page 448)</td>
<td>n/a</td>
<td>&lt;string&gt; ::= up to 254 characters as a quoted ASCII string</td>
</tr>
<tr>
<td>:SYSTem:LOCK &lt;value&gt; (see page 450)</td>
<td>:SYSTem:LOCK? (see page 450)</td>
<td>&lt;value&gt; ::= {{1</td>
</tr>
<tr>
<td>:SYSTem:PRECision &lt;value&gt; (see page 451)</td>
<td>:SYSTem:PRECision? (see page 451)</td>
<td>&lt;value&gt; ::= {{1</td>
</tr>
<tr>
<td>:SYSTem:PROTection:LOCK &lt;value&gt; (see page 452)</td>
<td>:SYSTem:PROTection:LOCK? (see page 452)</td>
<td>&lt;value&gt; ::= {{1</td>
</tr>
<tr>
<td>:SYSTem:SETup &lt;setup_data&gt; (see page 453)</td>
<td>:SYSTem:SETup? (see page 453)</td>
<td>&lt;setup_data&gt; ::= data in IEEE 488.2 # format.</td>
</tr>
<tr>
<td>:SYSTem:TIME &lt;time&gt; (see page 455)</td>
<td>:SYSTem:TIME? (see page 455)</td>
<td>&lt;time&gt; ::= hours,minutes,seconds in NR1 format</td>
</tr>
</tbody>
</table>

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**

(see page 788)

**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 446
- ":SYSTem:TIME" on page 455
Commands by Subsystem

:SYSTem:DSP

(see page 788)

Command Syntax

:SYSTem:DSP <string>

<string> ::= quoted ASCII string (up to 254 characters)

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYSTem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

See Also

- "Introduction to :SYSTem Commands" on page 446
**:SYStem:ERRor**

(see page 788)

**Query Syntax**

`:SYSTem:ERRor?`

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

**Return Format**

```
<error number>,<error string><NL>
```

<error number> ::= an integer error code in NR1 format
<error string> ::= quoted ASCII string containing the error message

Error messages are listed in Chapter 8, “Error Messages,” starting on page 745.

**See Also**

- "Introduction to :SYStem Commands" on page 446
- "*ESR (Standard Event Status Register)" on page 120
- "*CLS (Clear Status)" on page 117
5  Commands by Subsystem

:SYSTem:LOCK

(see page 788)

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 446
:SYSTem:PRECision

(see page 788)

Command Syntax

:SYSTem:PRECision <value>

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

The :SYSTem:PRECision command turns the oscilloscope's precision analysis setting on or off.

- OFF (0) — provides the maximum oscilloscope waveform update rate by performing measurements and math functions on a 1000-point measurement record.
- ON (1) — at the expense of oscilloscope waveform update rate, this setting allows measurements and math functions to be performed on a precision analysis record (see ":WAVEform:POINts:MODE" on page 634).

The precision analysis setting is OFF after a *RST command.

Precision analysis is not available when:
- Realtime sampling mode is off.
- Averaging or High Resolution acquisition modes are selected.
- XY or Roll time modes are selected.

Query Syntax

:SYSTem:PRECision?

The :SYSTem:PRECision? query returns the current precision analysis setting.

Return Format

<value><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 446
- ":WAVEform:POINts:MODE" on page 634
- "*RST (Reset)" on page 128
:SYSTem:PROTection:LOCK

(see page 788)

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 446
:SYSTem:SETup

(see page 788)

Command Syntax

:SYSTem:SETup <setup_data>

<setup_data> ::= binary block data in IEEE 488.2 # format.

The :SYSTem:SETup command sets the oscilloscope as defined by the data in the setup (learn) string sent from the controller. The setup string does not change the interface mode or interface address.

Query Syntax

:SYSTem:SETup?

The :SYSTem:SETup? query operates the same as the *LRN? query. It outputs the current oscilloscope setup in the form of a learn string to the controller. The setup (learn) string is sent and received as a binary block of data. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format

<setup_data><NL>

<setup_data> ::= binary block data in IEEE 488.2 # format

See Also

- "Introduction to :SYSTem Commands" on page 446
- "*LRN (Learn Device Setup)" on page 123

Example Code

' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message that contains the current state of the instrument. Its format is a definite-length binary block, for example, #800002204<setup string><NL> where the setup string is 2204 bytes in length.
myScope.WriteString " :SYSTEM:SETUP?"
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckForInstrumentErrors ' After reading query results.

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"

' Open file for output.
Close #1 ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult ' Write data.
Close #1 ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"

' Open file for input.
Open strPath For Binary Access Read As #1
Get #1, , varSetupString ' Read data.
Close #1 ' Close file.
Commands by Subsystem

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

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:SYSTem:TIME**

(see page 788)

**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 446
- ":SYSTem:DATE" on page 447
### TIMebase Commands

Control all horizontal sweep functions. See "Introduction to TIMebase Commands" on page 457.

#### Table 76 TIMebase Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TIMebase:MODE &lt;value&gt;</td>
<td>:TIMebase:MODE? (see page 458)</td>
<td>&lt;value&gt; ::= {MAIN</td>
</tr>
<tr>
<td>:TIMebase:POSition &lt;pos&gt;</td>
<td>:TIMebase:POSition? (see page 459)</td>
<td>&lt;pos&gt; ::= time from the trigger event to the display reference point in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:RANGe &lt;range_value&gt;</td>
<td>:TIMebase:RANGe? (see page 460)</td>
<td>&lt;range_value&gt; ::= 5 ns through 500 s in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:REFClock {0</td>
<td>OFF}</td>
<td>:TIMebase:REFClock? (see page 461)</td>
</tr>
<tr>
<td>:TIMebase:REFERence {LEFT</td>
<td>CENTER</td>
<td>RIGHT}</td>
</tr>
<tr>
<td>:TIMebase:SCALE &lt;scale_value&gt;</td>
<td>:TIMebase:SCALE? (see page 463)</td>
<td>&lt;scale_value&gt; ::= scale value in seconds in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:VERNier {0</td>
<td>OFF}</td>
<td>:TIMebase:VERNier? (see page 464)</td>
</tr>
<tr>
<td>:TIMebase:WINDow:POSition &lt;pos&gt;</td>
<td>:TIMebase:WINDow:POSition? (see page 465)</td>
<td>&lt;pos&gt; ::= time from the trigger event to the zoomed view reference point in NR3 format</td>
</tr>
<tr>
<td>:TIMebase:WINDow:RANGe &lt;range_value&gt;</td>
<td>:TIMebase:WINDow:RANGe? (see page 466)</td>
<td>&lt;range_value&gt; ::= range value in seconds in NR3 format for the zoomed window</td>
</tr>
<tr>
<td>:TIMebase:WINDow:SCALE &lt;scale_value&gt;</td>
<td>:TIMebase:WINDow:SCALE? (see page 467)</td>
<td>&lt;scale_value&gt; ::= scale value in seconds in NR3 format for the zoomed window</td>
</tr>
</tbody>
</table>
Introduction to :TIMebase Commands

The TIMebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (zoomed) time bases.

Reporting the Setup

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

Return Format

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

:TIM:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
:TIMebase:MODE

(see page 788)

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.

NOTE

If a :DIGitize command is executed when the :TIMebase:MODE is not MAIN, the :TIMebase:MODE is set to MAIN.

Query Syntax

:TIMebase:MODE?

The :TIMebase:MODE query returns the current time base mode.

Return Format

<value><NL>

<value> ::= {MAIN | WIND | XY | ROLL}

See Also

- "Introduction to :TIMebase Commands" on page 457
- "*RST (Reset)" on page 128
- ":TIMebase:RANGe" on page 460
- ":TIMebase:POSition" on page 459
- ":TIMebase:REFerence" on page 462

Example Code

' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.
myScope.WriteString ":TIMEBASE:MODE MAIN"

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:TIMebase:POSition

(see page 788)

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 457
• ":TIMebase:REFerence" on page 462
• ":TIMebase:RANGe" on page 460
• ":TIMebase:SCALe" on page 463
• ":TIMebase:WINDow:POSition" on page 465
• ":TIMebase:DELay" on page 739
The \texttt{: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.

The \texttt{:TIMebase:RANGe?} query returns the current full-scale range value for the main window.

The value is returned in NR3 format.

Example Code

```plaintext
' TIME_RANGE - Sets the full scale horizontal time in seconds. The
' range value is 10 times the time per division.
myScope.WriteString ":TIM:RANG 2e-3" ' Set the time range to 0.002 seconds.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 814.
**:TIMebase:REFClock**

N (see page 788)

**Command Syntax**

`:TIMebase:REFClock <value>`

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

The :TIMebase:REFClock command enables or disables the 10 MHz REF BNC located on the rear panel of the oscilloscope.

The 10 MHz REF BNC can be used as an input for the oscilloscope's reference clock (instead of the internal 10 MHz reference), or it can be used to output the internal 10 MHz reference clock when synchronizing multiple instruments (see ":ACQuire:RSIGnal" on page 189).

The :TIMebase:REFClock ON command enables the 10 MHz REF BNC and sets the reference signal mode to IN. The :TIMebase:REFClock OFF command disables the 10 MHz REF BNC (the same as setting the reference signal mode to OFF).

**Query Syntax**

`:TIMebase:REFClock?`

The :TIMebase:REFClock? query returns the current state of the 10 MHz reference signal mode. A "1" indicates that the 10 MHz REF input is enabled (on), and a "0" indicates that either the 10 MHz REF BNC is disabled (off) or that it is set as an output (by the :ACQuire:RSIGnal command).

**Return Format**

<value><NL>

<value> ::= {0 | 1}

**See Also**

- ":ACQuire:RSIGnal" on page 189
:TIMebase:REFerence

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 | RIGHT)

See Also
• "Introduction to :TIMebase Commands" on page 457
• ":TIMebase:MODE" on page 458

Example Code
'TIME_REFERENCE - Possible values are LEFT and CENTER.
' - LEFT sets the display reference on time division from the left.
' - CENTER sets the display reference to the center of the screen.
myScope.WriteString ':TIMEBASE:REFERENCE CENTER' ' Set reference to center.

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:TIMebase:SCALe

(see page 788)

Command Syntax

:TIMebase:SCALe <scale_value>

*scale_value* ::= 500 ps through 50 s in NR3 format

The :TIMebase:SCALe command sets the horizontal scale or units per division for the main window.

Query Syntax

:TIMebase:SCALe?

The :TIMebase:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

Return Format

*scale_value*<NL>

*scale_value* ::= 500 ps through 50 s in NR3 format

See Also

- "Introduction to :TIMebase Commands" on page 457
- ":TIMebase:RANGe" on page 460
- ":TIMebase:WINDow:SCALe" on page 467
- ":TIMebase:WINDow:RANGe" on page 466
 Commands by Subsystem

:TIMebase:VERNier

(see page 788)

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 457
:TIMebase:WINDow:POSition

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 457
- ":TIMebase:MODE" on page 458
- ":TIMebase:POSition" on page 459
- ":TIMebase:RANGe" on page 460
- ":TIMebase:SCALe" on page 463
- ":TIMebase:WINDow:RANGe" on page 466
- ":TIMebase:WINDow:SCALe" on page 467
**:TIMebase:WINDow:RANGe**

(see page 788)

**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 457
- ":.TIMebase:RANGe" on page 460
- ":.TIMebase:POSition" on page 459
- ":.TIMebase:SCALE" on page 463
:TIMebase:WINDow:SCALe

Command Syntax

`:TIMebase:WINDow:SCALe <scale_value>`

`<scale_value> ::= scale value in seconds in NR3 format`

The :TIMebase:WINDow:SCALe command sets the zoomed (delayed) window horizontal scale (seconds/division). The main sweep scale determines the range for this command. The maximum value is one half of the :TIMebase:SCALe value.

Query Syntax

`:TIMebase:WINDow:SCALe?`

The :TIMebase:WINDow:SCALe? query returns the current zoomed window scale setting.

Return Format

`<scale_value><NL>`

`<scale_value> ::= current seconds per division for the zoomed window`

See Also

- "Introduction to :TIMebase Commands" on page 457
- ":TIMebase:RANGe" on page 460
- ":TIMebase:POSition" on page 459
- ":TIMebase:SCALe" on page 463
- ":TIMebase:WINDow:RANGe" on page 466
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 479) 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 475).

- **CAN (Controller Area Network) triggering**— will trigger on CAN version 2.0A and 2.0B signals. Setup consists of connecting the oscilloscope to a CAN signal. Baud rate, signal source, and signal polarity, and type of data to trigger on can be specified. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on CAN data and identifier patterns, set the bit sample point, and have the module send an acknowledge to the bus when it receives a valid message.

- **Edge triggering**— identifies a trigger by looking for a specified slope and voltage level on a waveform.

- **Nth Edge Burst triggering**— lets you trigger on the Nth edge of a burst that occurs after an idle time.

- **Pulse width triggering**— (:TRIGger:GLITch commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.

- **Pattern triggering**— identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels.

- **Duration triggering**— lets you define a pattern, then trigger on a specified time duration.

- **FlexRay triggering**— will, when used with a BusDoctor 2 protocol analyzer and a four-channel mixed-signal oscilloscope with Option FRS, trigger on FlexRay bus frames, times, or errors.

- **I2S (Inter-IC Sound or Integrated Interchip Sound bus) triggering**— consists of connecting the oscilloscope to the serial clock, word select, and serial data lines, then triggering on a data value.

- **IIC (Inter-IC bus) triggering**— consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.

- **LIN (Local Interconnect Network) triggering**— will trigger on LIN sync break at the beginning of a message frame. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on Frame IDs.

- **MIL-STD 1553 triggering** (with Option 553) – lets you trigger on MIL-STD 1553 serial data.

**NOTE**

The CAN and LIN serial decode option (Option ASM) replaces the functionality that was available with the N2758A CAN trigger module for the 54620/54640 Series oscilloscopes.
• **Sequence triggering**— allows you to trigger the oscilloscope after finding a sequence of events. Defining a sequence trigger requires three steps:
  a. Define the event to find before you trigger on the next event. This event can be a pattern, and edge from a single channel, or the combination of a pattern and a channel edge.
  b. Define the trigger event. This event can be a pattern, and edge from a single channel, the combination of a pattern and a channel edge, or the nth occurrence of an edge from a single channel.
  c. Set an optional reset event. This event can be a pattern, an edge from a single channel, the combination of a pattern and a channel edge, or a timeout value.

• **SPI (Serial Peripheral Interface) triggering**— consists of connecting the oscilloscope to a clock, data, and framing signal. You can then trigger on a data pattern during a specific framing period. The serial data string can be specified to be from 4 to 32 bits long.

• **TV triggering**— is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than a division of sync amplitude with any analog channel as the trigger source.

• **UART/RS-232 triggering** (with Option 232) — lets you trigger on RS-232 serial data.

• **USB (Universal Serial Bus) triggering**— will trigger on a Start of Packet (SOP), End of Packet (EOP), Reset Complete, Enter Suspend, or Exit Suspend signal on the differential USB data lines. USB Low Speed and Full Speed are supported by this trigger.

**Reporting the Setup**

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

**Return Format**

The return format for the TRIGGER? query varies depending on the current mode. The following is a sample response from the `:TRIGGER?` query. In this case, the query was issued following a `*RST` command.

```plaintext
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.0000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```
## General :TRIGger Commands

### Table 77  General :TRIGger Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:HFReject {{0</td>
<td>OFF}</td>
<td>(1</td>
</tr>
<tr>
<td>:TRIGger:HOLDoff &lt;holdoff_time&gt; (see page 473)</td>
<td>:TRIGger:HOLDoff? (see page 473)</td>
<td>&lt;holdoff_time&gt; ::= 60 ns to 10 s in NR3 format</td>
</tr>
<tr>
<td>:TRIGger:LPIFty (see page 474)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:TRIGger:MODE &lt;mode&gt; (see page 475)</td>
<td>:TRIGger:MODE? (see page 475)</td>
<td>&lt;mode&gt; ::= {EDGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;none&gt; ::= query returns &quot;NONE&quot; if the :TIMebase:MODE is ROLL or XY</td>
</tr>
<tr>
<td>:TRIGger:NREJect {{0</td>
<td>OFF}</td>
<td>(1</td>
</tr>
<tr>
<td>:TRIGger:PATTern &lt;value&gt;, &lt;mask&gt; [,&lt;edge source&gt;,&lt;edge&gt;] (see page 477)</td>
<td>:TRIGger:PATTern? (see page 478)</td>
<td>&lt;value&gt; ::= integer in NR1 format or &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;edge source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;edge&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger:SWEep &lt;sweep&gt; (see page 479)</td>
<td>:TRIGger:SWEep? (see page 479)</td>
<td>&lt;sweep&gt; ::= {AUTO</td>
</tr>
</tbody>
</table>
:TRIGger:HFReject

(see page 788)

**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 468
- ":TRIGger[:EDGE]:REJect" on page 505
**:TRIGger:HZLDoff**

(see page 788)

**Command Syntax**

```plaintext
: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**

```plaintext
: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 468
:TRIGger:LFIfty

(see page 788)

Command Syntax

:TRIGger:LFIfty

The :TRIGger:LFIfty command sets the trigger level of a displayed analog channel trigger source to the waveform's 50% value.

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger[:EDGE]:SOURce" on page 507
- ":TRIGger[:EDGE]:LEVel" on page 504
**:TRIGger:MODE**

(see page 788)

Command Syntax

```
:TRIGger:MODE <mode>
```

- `<mode>` ::= {EDGE | GLITch | PATTern | CAN | DURation | I2S | IIC |
  - EBURst | LIN | M1553 | SEQuence | SPI | TV | UART |
  - USB | FLEXray}

The `:TRIGger:MODE` command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The `:TRIGger:MODE?` query returns the current trigger mode. If the `:TIMebase:MODE` is ROLL or XY, the query returns "NONE".

Return Format

```
<mode><NL>
```

- `<mode>` ::= {NONE | EDGE | GLIT | PATT | CAN | DUR | I2S |
  - IIC | EBUR | LIN | M1553 | SEQ | SPI | TV | UART |
  - USB | FLEX}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SWEep" on page 479
- ":TIMebase:MODE" on page 458

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE.
myScope.WriteString "::TRIGGER:MODE EDGE"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 814
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 468
**:TRIGger:PATTern**

(see page 788)

**Command Syntax**

**:TRIGger:PATTern <pattern>**

- `<pattern> ::= <value>, <mask> [, <edge source>, <edge>]`
- `<value> ::= integer in NR1 format or <string>`
- `<mask> ::= integer in NR1 format or <string>`
- `<string> ::= "0xnnnnn"; n ::= {0,..,9 | A,..,F}`
  (# bits = # channels, see following table)
- `<edge source> ::= (CHANnel<n> | EXTernal | NONE) for DSO models`
- `<edge source> ::= (CHANnel<n> | DIGital0,..,DIGital15 | NONE) for MSO models`
- `<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models`
- `<n> ::= {1 | 2} for the two channel oscilloscope models`
- `<edge> ::= (POSitive | NEGative)`

The :TRIGger:PATTern command defines the specified pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

<table>
<thead>
<tr>
<th>Oscilloscope Models</th>
<th>Value and Mask Bit Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 analog + 16 digital channels (mixed-signal)</td>
<td>Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.</td>
</tr>
<tr>
<td>2 analog + 16 digital channels (mixed-signal)</td>
<td>Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.</td>
</tr>
<tr>
<td>4 analog channels only</td>
<td>Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.</td>
</tr>
<tr>
<td>2 analog channels only</td>
<td>Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.</td>
</tr>
</tbody>
</table>

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

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

**NOTE**

The optional source and the optional edge should be sent together or not at all. The edge will be set in the simple pattern if it is included. If the edge source is also specified in the mask, the edge takes precedence.
Query Syntax

:TRIGger:PA TTern?

The :TRIGger:PA TTern? query returns the pattern value, the mask, and the edge of interest in the simple pattern.

Return Format

<pattern><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
:TRIGger:SWEep

(see page 788)

**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 468
### :TRIGger:CAN Commands

#### Table 78 :TRIGger:CAN Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:CAN:PATTern: DATA &lt;value&gt;, &lt;mask&gt; (see page 482)</td>
<td>:TRIGger:CAN:PATTern: DATA? (see page 482)</td>
<td>&lt;value&gt; ::= 64-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; (with Option AMS) &lt;mask&gt; ::= 64-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; &lt;nondecimal&gt; ::= #Hnn...n where n ::= {0,..,9</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTern: DATA:LENGTH &lt;length&gt; (see page 483)</td>
<td>:TRIGger:CAN:PATTern: DATA:LENGTH? (see page 483)</td>
<td>&lt;length&gt; ::= integer from 1 to 8 in NR1 format (with Option AMS)</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTern: ID &lt;value&gt;, &lt;mask&gt; (see page 484)</td>
<td>:TRIGger:CAN:PATTern: ID? (see page 484)</td>
<td>&lt;value&gt; ::= 32-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; (with Option AMS) &lt;mask&gt; ::= 32-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; &lt;nondecimal&gt; ::= #Hnn...n where n ::= {0,..,9</td>
</tr>
<tr>
<td>:TRIGger:CAN:PATTern: ID:MODE &lt;value&gt; (see page 485)</td>
<td>:TRIGger:CAN:PATTern: ID:MODE? (see page 485)</td>
<td>&lt;value&gt; ::= {STANdard</td>
</tr>
<tr>
<td>:TRIGger:CAN:SIGNal:BAUDrate &lt;baudrate&gt; (see page 487)</td>
<td>:TRIGger:CAN:SIGNal:BAUDrate? (see page 487)</td>
<td>&lt;baudrate&gt; ::= integer from 10000 to 1000000 in 100 b/s increments</td>
</tr>
</tbody>
</table>
Table 78  :TRIGger:CAN Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:CAN:SIGNal:DEFinition &lt;value&gt; (see page 488)</td>
<td>:TRIGger:CAN:SIGNal:DEFinition? (see page 488)</td>
<td>&lt;value&gt; ::= {CANH</td>
</tr>
<tr>
<td>:TRIGger:CAN:SOURce &lt;source&gt; (see page 489)</td>
<td>:TRIGger:CAN:SOURce? (see page 489)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:CAN:TRIGger &lt;condition&gt; (see page 490)</td>
<td>:TRIGger:CAN:TRIGger? (see page 491)</td>
<td>&lt;condition&gt; ::= {SOF} (without Option AMS) &lt;condition&gt; ::= {SOF</td>
</tr>
</tbody>
</table>
:TRIGger:CAN:PATTern:DATA

(see page 788)

Command Syntax

:TRIGger:CAN:PATTern:DATA <value>,<mask>

<value> ::= 64-bit integer in decimal, <nondecimal>, or <string>

<mask> ::= 64-bit integer in decimal, <nondecimal>, or <string>

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

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

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

The :TRIGger:CAN:PATTern:DATA command defines the CAN data pattern resource according to the value and the mask. This pattern, along with the data length (set by the :TRIGger:CAN:PATTern:DATA:LENGth command), control the data pattern searched for in each CAN message.

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

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

NOTE

If more bytes are sent for <value> or <mask> than specified by the :TRIGger:CAN:PATTern:DATA:LENGth command, the most significant bytes will be truncated. If the data length is changed after the <value> and <mask> are programmed, the added or deleted bytes will be added to or deleted from the least significant bytes.

NOTE

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

:TRIGger:CAN:PATTern:DATA?


Return Format

<value>, <mask><NL> in nondecimal format

Errors

- "-241, Hardware missing" on page 747

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 483
- ":TRIGger:CAN:PATTern:ID" on page 484
:TRIgger:CAN:PATTern:DATA:LENGth

(see page 788)

Command Syntax

:TRIgger:CAN:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format

The :TRIgger:CAN:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string. The number of bytes in the string can be anywhere from 0 bytes to 8 bytes (64 bits). The value for these bytes is set by the :TRIgger:CAN:PATTern:DATA command.

NOTE

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

:TRIgger:CAN:PATTern:DATA:LENGth?

The :TRIgger:CAN:PATTern:DATA:LENGth? query returns the current CAN data pattern length setting.

Return Format

<count><NL>

<count> ::= integer from 1 to 8 in NR1 format

Errors

- "-241, Hardware missing" on page 747

See Also

- "Introduction to :TRIgger Commands" on page 468
- ":TRIgger:CAN:PATTern:DATA" on page 482
- ":TRIgger:CAN:SOURce" on page 489
:TRIGger:CAN:PATTern:ID

(see page 788)

Command Syntax

:TRIGger:CAN:PATTern:ID <value>, <mask>

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

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

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

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

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

The :TRIGger:CAN:PATTern:ID command defines the CAN identifier pattern resource according to the value and the mask. This pattern, along with the identifier mode (set by the :TRIGger:CAN:PATTern:ID:MODE command), control the identifier pattern searched for in each CAN message.

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

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

NOTE
If more bits are sent than allowed (11 bits in standard mode, 29 bits in extended mode) by the :TRIGger:CAN:PATTern:ID:MODE command, the most significant bytes will be truncated. If the ID mode is changed after the <value> and <mask> are programmed, the added or deleted bits will be added to or deleted from the most significant bits.

NOTE
This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

:TRIGger:CAN:PATTern:ID?


Return Format

<value>, <mask><NL> in nondecimal format

Errors

• "-241, Hardware missing" on page 747

See Also

• "Introduction to :TRIGger Commands" on page 468
• ":TRIGger:CAN:PATTern:ID:MODE" on page 485
• ":TRIGger:CAN:PATTern:DATA" on page 482
:TRIGger:CAN:PATTERn:ID:MODE

Command Syntax
:TRIGger:CAN:PATTERn:ID:MODE <value>

<value> ::= {STANdard | EXTended}


Query Syntax
:TRIGger:CAN:PATTERn:ID:MODE?

The :TRIGger:CAN:PATTERn:ID:MODE? query returns the current setting of the CAN identifier mode.

Return Format
<value><NL>

<value> ::= {STAN | EXT}

Errors
- "-241, Hardware missing" on page 747

See Also
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:PATTERn:DATA" on page 482
- ":TRIGger:CAN:PATTERn:DATA:LENGTH" on page 483
- ":TRIGger:CAN:PATTERn:ID" on page 484

NOTE
This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.
**:TRIGger:CAN:SAMPlepoint**

(see page 788)

**Command Syntax**

:TRIGger:CAN:SAMPlepoint <value>

<value><NL>

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

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

**Query Syntax**

:TRIGger:CAN:SAMPlepoint?

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

**Return Format**

<value><NL>

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

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:TRIGger" on page 490
:TRIGger:CAN:SIGNal:BAUDrate

(see page 788)

Command Syntax

:TRIGger:CAN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer from 10000 to 1000000 in 100 b/s increments

The :TRIGger:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 1 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.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

:TRIGger:CAN:SIGNal:BAUDrate?

The :TRIGger:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Return Format

<baudrate><NL>

<baudrate> ::= integer from 10000 to 1000000 in 100 b/s increments

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:TRIGger" on page 490
- ":TRIGger:CAN:SIGNal:DEFinition" on page 488
- ":TRIGger:CAN:SOURce" on page 489
:TRIGger:CAN:SIGNal:DEFinition

(see page 788)

Command Syntax

:TRIGger:CAN:SIGNal:DEFinition <value>

<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}

The :TRIGger:CAN:SIGNal:DEFinition command sets the CAN signal type when :TRIGger:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signal:
- CANH — the actual CAN_H differential bus signal.
- 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.
- DIFFerential — the CAN differential bus signal connected to an analog source channel using a differential probe.
- DIFL — the CAN differential (L-H) bus signal connected to an analog source channel using a differential probe. This is the same as DIFFerential.

Query Syntax

:TRIGger:CAN:SIGNal:DEFinition?


Return Format

<value><NL>

<value> ::= {CANH | CANL | RX | TX | DIFF | DIFH}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:SIGNal:BAUDrate" on page 487
- ":TRIGger:CAN:SOURce" on page 489
- ":TRIGger:CAN:TRIGger" on page 490
Commands by Subsystem

:TRIGger:CAN:SOURce

(see page 788)

Command Syntax

:TRIGger:CAN:SOURce <source>

<source> ::= (CHANnel<n> | EXTernal) for the DSO models
<source> ::= (CHANnel<n> | DIGital0,...,DIGital15) for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:CAN:SOURce command sets the source for the CAN signal. The source setting is only valid when :TRIGger:CAN:TRIGger is set to SOF (start of frame).

Query Syntax

:TRIGger:CAN:SOURce?

The :TRIGger:CAN:SOURce? query returns the current source for the CAN signal.

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:TRIGger" on page 490
- ":TRIGger:CAN:SIGNal:DEFinition" on page 488
Commands by Subsystem

:TRIGger:CAN:TRIGger

(see page 788)

Command Syntax

:TRIGger:CAN:TRIGger <condition>

<condition> ::= (SOF | DATA | ERROR | IDData | IDEither | IDRemote | ALLerrors | OVERload | ACKerror)

The :TRIGger:CAN:TRIGger command sets the CAN trigger on condition:

- **SOF** - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- **DATA** - will trigger on CAN Data frames matching the specified Id, Data, and the DLC (Data length code).
- **ERROR** - will trigger on CAN Error frame.
- **IDData** - will trigger on CAN frames matching the specified Id of a Data frame.
- **IDEither** - will trigger on the specified Id, regardless if it is a Remote frame or a Data frame.
- **IDRemote** - will trigger on CAN frames matching the specified Id of a Remote frame.
- **ALLerrors** - will trigger on CAN active error frames and unknown bus conditions.
- **OVERload** - will trigger on CAN overload frames.
- **ACKerror** - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

<table>
<thead>
<tr>
<th>Remote &lt;condition&gt; parameter</th>
<th>Front-panel Trigger on: softkey selection (softkey text - softkey popup text)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOF</td>
<td>SOF - Start of Frame</td>
</tr>
<tr>
<td>DATA</td>
<td>Id &amp; Data - Data Frame Id and Data</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error - Error frame</td>
</tr>
<tr>
<td>IDData</td>
<td>Id &amp; ~RTR - Data Frame Id (~RTR)</td>
</tr>
<tr>
<td>IDEither</td>
<td>Id - Remote or Data Frame Id</td>
</tr>
<tr>
<td>IDRemote</td>
<td>Id &amp; RTR - Remote Frame Id (RTR)</td>
</tr>
<tr>
<td>ALLerrors</td>
<td>All Errors - All Errors</td>
</tr>
<tr>
<td>OVERload</td>
<td>Overload - Overload Frame</td>
</tr>
<tr>
<td>ACKerror</td>
<td>Ack Error - Acknowledge Error</td>
</tr>
</tbody>
</table>

CAN Data specification is set by the :TRIGger:CAN:PATTern:DATA command.

CAN Data Length Code is set by the :TRIGger:CAN:PATTern:DATA:LENGth command.

**NOTE**

SOF is the only valid selection for analog oscilloscopes. If the automotive CAN and LIN serial decode option (Option AMS) has not been licensed, SOF is the only valid selection.

**Query Syntax**

:TRIGger:CAN:TRIG?er?

The :TRIGger:CAN:TRIGger? query returns the current CAN trigger on condition.

**Return Format**

<condition><NL>
<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}

**Errors**

- "-241, Hardware missing" on page 747

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:PATTern:DATA" on page 482
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 483
- ":TRIGger:CAN:PATTern:ID" on page 484
- ":TRIGger:CAN:PATTern:ID:MODE" on page 485
- ":TRIGger:CAN:SIGNal:DEFinition" on page 488
- ":TRIGger:CAN:SOURce" on page 489
### :TRIGger:DURation Commands

**Table 79  :TRIGger:DURation Commands Summary**

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:DURation:GREaterthan &lt;greater_than_time&gt;[suffix] (see page 493)</td>
<td>:TRIGger:DURation:GREaterthan? (see page 493)</td>
<td>&lt;greater_than_time&gt; ::= floating-point number in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:TRIGger:DURation:LESSthan &lt;less_than_time&gt;[suffix] (see page 494)</td>
<td>:TRIGger:DURation:LESSthan? (see page 494)</td>
<td>&lt;less_than_time&gt; ::= floating-point number from in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:TRIGger:DURation:PATTERN &lt;value&gt;, &lt;mask&gt; (see page 495)</td>
<td>:TRIGger:DURation:PATTERN? (see page 495)</td>
<td>&lt;value&gt; ::= integer or &lt;string&gt; &lt;mask&gt; ::= integer or &lt;string&gt; &lt;string&gt; ::= &quot;0xnnnnnn&quot; n ::= {0,..,9</td>
</tr>
<tr>
<td>:TRIGger:DURation:QUALifier &lt;qualifier&gt; (see page 496)</td>
<td>:TRIGger:DURation:QUALifier? (see page 496)</td>
<td>&lt;qualifier&gt; ::= {GREaterthan</td>
</tr>
<tr>
<td>:TRIGger:DURation:RANGE &lt;less_than_time&gt;[suffix], &lt;greater_than_time&gt;[suffix] (see page 497)</td>
<td>:TRIGger:DURation:RANGE? (see page 497)</td>
<td>&lt;less_than_time&gt; ::= 15 ns to 10 seconds in NR3 format &lt;greater_than_time&gt; ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s</td>
</tr>
</tbody>
</table>
**:TRIGger:DURation:GREaterthan**

(see page 788)

**Command Syntax**

```
:TTRIGger:DURation:GREaterthan <greater_than_time>[:<suffix>]
```

<greater_than_time> ::= minimum trigger duration in seconds in NR3 format

<suffix> ::= \{s | ms | us | ns | ps \}

The :TRIgger:DURation:GREaterthan command sets the minimum duration for the defined pattern when :TRIgger:DURation:QUALifier is set to GREaterthan. The command also sets the timeout value when the :TRIgger:DURation:QUALifier is set to TIMeout.

**Query Syntax**

```
:TTRIGger:DURation:GREaterthan?
```

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

**Return Format**

```
<greater_than_time><NL>
```

**See Also**

- "Introduction to :TRIgger Commands" on page 468
- "::TRIgger:DURation:PATTer" on page 495
- "::TRIgger:DURation:QUALifier" on page 496
- "::TRIgger:MODE" on page 475
Commands by Subsystem

:TRIGger:DURation:LESSthan

N (see page 788)

Command Syntax

:TRIGger:DURation:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= maximum trigger duration in seconds in NR3 format

<suffix> ::= {s | ms | us | ns | ps}

The :TRIGger:DURation:LESSthan command sets the maximum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to LESSthan.

Query Syntax

:TRIGger:DURation:LESSthan?

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

Return Format

<less_than_time><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:DURation:PATTern" on page 495
- ":TRIGger:DURation:QUALifier" on page 496
- ":TRIGger:MODE" on page 475
**:TRIGger:DURation:PATTern**

![image](image1.png)  
(see page 788)

**Command Syntax**

`:TRIGger:DURation:PATTern <value>, <mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

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

The :TRIGger:DURation:PATTern command defines the specified duration pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

<table>
<thead>
<tr>
<th>Oscilloscope Models</th>
<th>Value and Mask Bit Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 analog + 16 digital channels (mixed-signal)</td>
<td>Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.</td>
</tr>
<tr>
<td>2 analog + 16 digital channels (mixed-signal)</td>
<td>Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.</td>
</tr>
<tr>
<td>4 analog channels only</td>
<td>Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.</td>
</tr>
<tr>
<td>2 analog channels only</td>
<td>Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.</td>
</tr>
</tbody>
</table>

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

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

**Query Syntax**

`:TRIGger:DURation:PATTern?`

The :TRIGger:DURation:PATTern? query returns the pattern value.

**Return Format**

<value>, <mask><NL>

<value> ::= a 32-bit integer in NR1 format.

<mask> ::= a 32-bit integer in NR1 format.

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:PATTern" on page 477
**:TRIGger:DURation:QUALifier**

N (see page 788)

**Command Syntax**

:TRIGger:DURation:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRate | TIMeout}

The :TRIGger:DURation:QUALifier command qualifies the trigger duration.

Set the GREaterthan qualifier value with the :TRIGger:DURation:GREaterthan command.

Set the LESSthan qualifier value with the :TRIGger:DURation:LESSthan command.

Set the INRange and OUTRange qualifier values with the :TRIGger:DURation:RANGe command.

Set the TIMeout qualifier value with the :TRIGger:DURation:GREaterthan command.

**Query Syntax**

:TRIGger:DURation:QUALifier?

The :TRIGger:DURation:QUALifier? query returns the trigger duration qualifier.

**Return Format**

<qualifier><NL>

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:DURation:GREaterthan" on page 493
- ":TRIGger:DURation:LESSthan" on page 494
- ":TRIGger:DURation:RANGe" on page 497
:TRIGger:DURation:RANGe

(see page 788)

Command Syntax

:TRIGger:DURation:RANGe <less_than_time>[<suffix>],
<greater_than_time>[<suffix>]

<greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format
<less_than_time> ::= 15 ns to 10 seconds in NR3 format

<suffix> ::= {s | ms | us | ns | ps}

The :TRIGger:DURation:RANGe command sets the duration for the defined pattern when the :TRIGger:DURation:QUALifier command is set to INRange or OUTRange. 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:DURation:RANGe?

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

Return Format

<less_than_time>,<greater_than_time><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:DURation:PArrTern" on page 495
- ":TRIGger:DURation:QUALifier" on page 496
- ":TRIGger:MODE" on page 475
The :TRIGger:EDGE:SOURce command is used to specify the source channel for the Nth Edge Burst trigger. If an analog channel is selected as the source, the :TRIGger:EDGE:LEVel command is used to set the Nth Edge Burst trigger level. If a digital channel is selected as the source, the :DIGital<n>:THReshold or :POD<n>:THReshold command is used to set the Nth Edge Burst trigger level.
:TRIGger:EBURst:COUNt

(see page 788)

Command Syntax
:TRIGger:EBURst:COUNt <count>
<count> ::= integer in NR1 format

The :TRIGger:EBURst:COUNt command sets the Nth edge at burst counter resource. The edge counter is used in the trigger stage to determine which edge in a burst will generate a trigger.

Query Syntax
:TRIGger:EBURst:COUNt?

The :TRIGger:EBURst:COUNt? query returns the current Nth edge of burst edge counter setting.

Return Format
<count><NL>
<count> ::= integer in NR1 format

See Also
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:EBURst:SLOPe" on page 501
- ":TRIGger:EBURst:IDLE" on page 500
**:TRIGger:EBURst:IDLE**

(see page 788)

**Command Syntax**

:TRIGger:EBURst:IDLE <time_value>

<time_value> ::= time in seconds in NR3 format

The :TRIGger:EBURst:IDLE command sets the Nth edge in a burst idle resource in seconds from 10 ns to 10 s. The timer is used to set the minimum time before the next burst.

**Query Syntax**

:TRIGger:EBURst:IDLE?


**Return Format**

<time value><NL>

<time_value> ::= time in seconds in NR3 format

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:EBURst:SLOPe" on page 501
- ":TRIGger:EBURst:COUNT" on page 499
:TRIGger:EBURst:SLOPe

(see page 788)

**Command Syntax**

:TRIGger:EBURst:SLOPe <slope>

<slope> ::= (NEGative | POSitive)

The :TRIGger:EBURst:SLOPe command specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.

**Query Syntax**

:TRIGger:EBURst:SLOPe?

The :TRIGger:EBURst:SLOPe? query returns the current Nth edge in a burst slope.

**Return Format**

<slope><NL>

<slope> ::= (NEG | POS)

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ".:TRIGger:EBURst:IDLE" on page 500
- ".:TRIGger:EBURst:COUNt" on page 499
### Table 81 :TRIGger[:EDGE] Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger[:EDGE]:COUPling {AC</td>
<td>DC</td>
<td>LF} (see page 503)</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:LEVel &lt;level&gt; [,&lt;source&gt;] (see page 504)</td>
<td>:TRIGger[:EDGE]:LEVel? [&lt;source&gt;] (see page 504)</td>
<td>For internal triggers, &lt;level&gt; ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, &lt;level&gt; ::= ±(external range setting) in NR3 format. For digital channels (MSO models), &lt;level&gt; ::= ±8 V. &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:REJect {OFF</td>
<td>LF</td>
<td>HF} (see page 505)</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:SLOPe &lt;polarity&gt; (see page 506)</td>
<td>:TRIGger[:EDGE]:SLOPe? (see page 506)</td>
<td>&lt;polarity&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger[:EDGE]:SOURce &lt;source&gt; (see page 507)</td>
<td>:TRIGger[:EDGE]:SOURce? (see page 507)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
**:TRIGger[:EDGE]:COUPling**

(see page 788)

**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 468
- ".TRIGger:MODE" on page 475
- ".TRIGger[:EDGE]:REJect" on page 505
Commands by Subsystem

:TRIGger[:EDGE]:LEVel

(see page 788)

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> | DIGital0,...,DIGital15 | EXTernal} for the MSO models

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

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

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

NOTE

If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

Query Syntax

:TRIGger[:EDGE]:LEVel? [<source>]

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

Return Format

<level><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger[:EDGE]:SOURce" on page 507
- ":EXTernal:RANGe" on page 262
- ":POD<n>:THReshold" on page 396
- ":DIGital<n>:THReshold" on page 243
:TRIgger[:EDGE]:REJect

(see page 788)

Command Syntax

:TRIgger[:EDGE]:REJect <reject>

<reject> ::= (OFF | LFR| LFR | HFR)

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 468
• ":TRIgger:HFR| Reject" on page 472
• ":TRIgger[:EDGE]:COUpling" on page 503
**:TRIGger[:EDGE]:SLOPe**

(see page 788)

**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 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:TV:POLarity" on page 596

**Example Code**

' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:TRIGger[:EDGE]:SOURce**

(see page 788)

**Command Syntax**

:TRIGger[:EDGE]:SOURce <source>

<source> ::= (CHANnel<n> | EXTERNAL | LINE) for the DSO models

<source> ::= (CHANnel<n> | DIGital0,...,DIGital15 | EXTERNAL | LINE) for the MSO models

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :TRIGger[:EDGE]:SOURce command selects the channel that produces the trigger.

**Query Syntax**

:TRIGger[:EDGE]:SOURce?

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

**Return Format**

<source><NL>

<source> ::= (CHAN<n> | EXT | LINE | NONE) for the DSO models

<source> ::= (CHAN<n> | DIG0,...,DIG15 | EXTERNAL | LINE | NONE) for the MSO models

**See Also**

- "Introduction to :TRIGGER Commands" on page 468
- ":TRIGGER:MODE" on page 475

**Example Code**

```
' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces the edge trigger. Any channel can be selected.
myScope.WriteString "':TRIGGER:EDGE:SOURCE CHANNEL1"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 814
### :TRIGger:FLEXray Commands

#### Table 82 :TRIGger:FLEXray Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:FLEXray:AUTOsetup (see page 509)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:BAUDrate &lt;baudrate&gt; (see page 510)</td>
<td>:TRIGger:FLEXray:BAUDrate? (see page 510)</td>
<td>&lt;baudrate&gt; ::= {2500000</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:CHANnel &lt;channel&gt; (see page 511)</td>
<td>:TRIGger:FLEXray:CHANnel? (see page 511)</td>
<td>&lt;channel&gt; ::= {A</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:ERRor:TYPE &lt;error_type&gt; (see page 512)</td>
<td>:TRIGger:FLEXray:ERRor:TYPE? (see page 512)</td>
<td>&lt;error_type&gt; ::= {ALL</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:EVENt:TYPE &lt;event&gt; (see page 513)</td>
<td>:TRIGger:FLEXray:EVENt:TYPE? (see page 513)</td>
<td>&lt;event&gt; ::= {WAKeup</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:CCBase &lt;cycle_count_base&gt; (see page 514)</td>
<td>:TRIGger:FLEXray:FRAMe:CCBase? (see page 514)</td>
<td>&lt;cycle_count_base&gt; ::= integer from 0-63</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:CCRepetition &lt;cycle_count_repetition&gt; (see page 515)</td>
<td>:TRIGger:FLEXray:FRAMe:CCRepetition? (see page 515)</td>
<td>&lt;cycle_count_repetition&gt; ::= {ALL</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:ID &lt;frame_id&gt; (see page 516)</td>
<td>:TRIGger:FLEXray:FRAMe:ID? (see page 516)</td>
<td>&lt;frame_id&gt; ::= {ALL</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:FRAMe:TYPE &lt;frame_type&gt; (see page 517)</td>
<td>:TRIGger:FLEXray:FRAMe:TYPE? (see page 517)</td>
<td>&lt;frame_type&gt; ::= {NORMal</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:SOURce &lt;source&gt; (see page 518)</td>
<td>:TRIGger:FLEXray:SOURce? (see page 518)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;}</td>
</tr>
<tr>
<td>:TRIGger:FLEXray:TRIGger &lt;condition&gt; (see page 519)</td>
<td>:TRIGger:FLEXray:TRIGger? (see page 519)</td>
<td>&lt;condition&gt; ::= {FRAME</td>
</tr>
</tbody>
</table>
**:TRIGger:FLEXray:AUToSetup**

(see page 788)

**Command Syntax**

```
:TRIGger:FLEXray:AUToSetup
```

The :TRIGger:FLEXray:AUToSetup command automatically configures oscilloscope settings to facilitate FlexRay triggering and serial decode.

- Sets the selected source channel's impedance to 50 Ohms.
- Sets the selected source channel's probe attenuation to 10:1.
- Sets the trigger level (on the selected source channel) to -300 mV.
- Turns on trigger Noise Reject.
- Turns on Serial Decode.
- Sets the trigger type to FlexRay.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray:TRIGger" on page 519
- ":TRIGger:FLEXray:BAUDrate" on page 510
- ":TRIGger[:EDGE]:LEVel" on page 504
- ":TRIGger:FLEXray:SOURce" on page 518
:TRIGger:FLEXray:BAUDrate

(see page 788)

**Command Syntax**

:TRIGger:FLEXray:BAUDrate <baudrate>

<baudrate> ::= {2500000 | 5000000 | 10000000}

The :TRIGger:FLEXray:BAUDrate command specifies the baud rate as 2.5 Mb/s, 5 Mb/s, or 10 Mb/s.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the FlexRay triggering and serial decode option (Option FLX) has been licensed.

**Query Syntax**

:TRIGger:FLEXray:BAUDrate?

The :TRIGger:FLEXray:BAUDrate? query returns the current baud rate setting.

**Return Format**

<baudrate><NL>

<baudrate> ::= {2500000 | 5000000 | 10000000}

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray Commands" on page 508
Commands by Subsystem 5

:TRIGger:FLEXray:CHANnel

N  (see page 788)

Command Syntax  :TRIGger:FLEXray:CHANnel <channel>

<channel> ::= {A | B}

The :TRIGger:FLEXray:CHANnel command specifies the bus channel, A or B, of the FlexRay signal.

NOTE  This command is only valid on 4 (analog) channel oscilloscope models when the FlexRay triggering and serial decode option (Option FLX) has been licensed.

Query Syntax  :TRIGger:FLEXray:CHANnel?

The :TRIGger:FLEXray:CHANnel? query returns the current bus channel setting.

Return Format  <channel><NL>

<channel> ::= {A | B}

See Also  • "Introduction to :TRIGger Commands" on page 468
  • ":TRIGger:FLEXray Commands" on page 508
:TRIGger:FLEXray:ERRor:TYPE

(see page 788)

Command Syntax
:TRIGger:FLEXray:ERRor:TYPE <error_type>

<error_type> ::= {ALL | HCRC | FCRC}

Selects the FlexRay error type to trigger on. The error type setting is only valid when the FlexRay trigger mode is set to ERRor.

- ALL — triggers on ALL errors.
- HCRC — triggers on only Header CRC errors.
- FCRC — triggers on only Frame CRC errors.

NOTE
This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

Query Syntax
:TRIGger:FLEXray:ERRor:TYPE?

The :TRIGger:FLEXray:ERRor:TYPE? query returns the currently selected FLEXray error type.

Return Format
<error_type><NL>

<error_type> ::= {ALL | HCRC | FCRC}

See Also
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray:TRIGger" on page 519
:**TRIGger:FLEXray:EVENt:TYPE**

(see page 788)

**Command Syntax**

:TRIGger:FLEXray:EVENt:TYPE <event>

<event> ::= (WAKeup | TSS | (FES | DTS) | BSS)

Selects the FlexRay event to trigger on. The event setting is only valid when the FlexRay trigger mode is set to EVENt.

- WAKeup — triggers on Wake-Up event.
- TSS — triggers on Transmission Start Sequence event.
- FES — triggers on Frame End Sequence event.
- DTS — triggers on Dynamic Trailing Sequence event.
- BSS — triggers on Byte Start Sequence event.

**NOTE**

FES and DTS are equivalent.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

**Query Syntax**

:TRIGger:FLEXray:EVENt:TYPE?

The :TRIGger:FLEXray:EVENt:TYPE? query returns the currently selected FLEXray event.

**Return Format**

<event><NL>

<event> ::= (WAK | TSS | (FES | DTS) | BSS)

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray:TRIGger" on page 519
- ":TRIGger:FLEXray:AUTosetup" on page 509
- ":TRIGger:FLEXray:SOURce" on page 518
**Command Syntax**

:TRIGger:FLEXray:FRAME:CCBase <cycle_count_base>

<cycle_count_base> ::= integer from 0-63

The :TRIGger:FLEXray:FRAME:CCBase command sets the base of the FlexRay cycle count (in the frame header) to trigger on. The cycle count base setting is only valid when the FlexRay trigger mode is set to FRAME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

**Query Syntax**

:TRIGger:FLEXray:FRAME:CCBase?

The :TRIGger:FLEXray:FRAME:CCBase? query returns the current cycle count base setting for the FlexRay frame trigger setup.

**Return Format**

<cycle_count_base><NL>

<cycle_count_base> ::= integer from 0-63

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray:TRIGger" on page 519
**:TRIGger:FLEXray:FRAME:CCRepetition**

(see page 788)

**Command Syntax**

`:TRIGger:FLEXray:FRAME:CCRepetition <cycle_count_repetition>

<cycle_count_repetition> ::= {ALL | <rep #>}

<rep #> ::= integer from 2-64

The :TRIGger:FLEXray:FRAME:CCRepetition command sets the repetition number of the FlexRay cycle count (in the frame header) to trigger on. The cycle count repetition setting is only valid when the FlexRay trigger mode is set to FRAME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

**Query Syntax**

`:TRIGger:FLEXray:FRAME:CCRepetition?`

The :TRIGger:FLEXray:FRAME:CCRepetition? query returns the current cycle count repetition setting for the FlexRay frame trigger setup.

**Return Format**

<cycle_count_repetition><NL>

<cycle_count_repetition> ::= {ALL | <rep #>}

<rep #> ::= integer from 2-64

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray:TRIGger" on page 519
**:TRIGger:FLEXray:FRAME:ID**

(see page 788)

**Command Syntax**

:TRIGger:FLEXray:FRAME:ID <frame_id>

[frame_id] ::= {ALL | <frame #>}

[frame #] ::= integer from 1-2047

The :TRIGger:FLEXray:FRAME:ID command sets the FlexRay frame ID to trigger on. The frame IF setting is only valid when the FlexRay trigger mode is set to FRAME.

**NOTE**

This command is only valid when the FlexRay triggering and serial decode option (Option FLX) has been licensed.

**Query Syntax**

:TRIGger:FLEXray:FRAME:ID?

The :TRIGger:FLEXray:FRAME:ID? query returns the current frame ID setting for the FlexRay frame trigger setup.

**Return Format**

<frame_id><NL>

[frame_id] ::= {ALL | <frame #>}

[frame #] ::= integer from 1-2047

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:FLEXray:TRIGger" on page 519
:TRIGger:FLEXray:FRAME:TYPE

(see page 788)

Command Syntax
:TRIGger:FLEXray:FRAME:TYPE <frame_type>

[frame_type] ::= {NORMal | STARTup | NULL | SYNC | NSTArtup | NNULl | NSYNc | ALL}

The :TRIGger:FLEXray:FRAME:TYPE command sets the FlexRay frame type to trigger on. The frame type setting is only valid when the FlexRay trigger mode is set to FRAME.

- NORMal — will trigger on only normal (NSTArtup & NNULl & NSYNc) frames.
- STARTup — will trigger on only startup frames.
- NULL — will trigger on only null frames.
- SYNC — will trigger on only sync frames.
- NSTArtup — will trigger on frames other than startup frames.
- NNULl — will trigger on frames other than null frames.
- NSYNc — will trigger on frames other than sync frames.
- ALL — will trigger on all FlexRay frame types.

NOTE
This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

Query Syntax
:TRIGger:FLEXray:FRAME:TYPE?

The :TRIGger:FLEXray:FRAME:TYPE? query returns the current frame type setting for the FlexRay frame trigger setup.

Return Format
[frame_type]<NL>

[frame_type] ::= {NORM | STAR | NULL | SYNC | NSTA | NNUL | NSYN | ALL}

See Also
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:FLEXray:TRIGger" on page 519
Commands by Subsystem

:TRIGger:FLEXray:SOURce

Command Syntax

:TRIGger:FLEXray:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= (1 | 2 | 3 | 4)

The :TRIGger:FLEXray:SOURce command specifies the input source for the FlexRay signal.

NOTE

This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

Query Syntax

:TRIGger:FLEXray:SOURce?

The :TRIGger:FLEXray:SOURce? query returns the current source for the FlexRay signal.

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:FLEXray:TRIGger" on page 519
- ":TRIGger:FLEXray:EVENt:TYPE" on page 513
- ":TRIGger:FLEXray:AUTosetup" on page 509
:TRIGger:FLEXray:TRIGger

Command Syntax

:TRIGger:FLEXray:TRIGger <condition>

<condition> ::= (FRAMe | ERRor | EVENt)

The :TRIGger:FLEXray:TRIGger command sets the FLEXray trigger on condition:

- FRAMe — triggers on specified frames (without errors).
- ERRor — triggers on selected active error frames and unknown bus conditions.
- EVENt — triggers on specified FlexRay event/symbol.

NOTE

This command is only valid when the FLEXray triggering and serial decode option (Option FLX) has been licensed.

Query Syntax

:TRIGger:FLEXray:TRIGger?

The :TRIGger:FLEXray:TRIGger? query returns the current FLEXray trigger on condition.

Return Format

<condition><NL>

<condition> ::= (FRAM | ERR | EVEN)

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:FLEXray:ERRor:TYPE" on page 512
- ":TRIGger:FLEXray:EVENt:TYPE" on page 513
- ":TRIGger:FLEXray:FRAMe:CCBase" on page 514
- ":TRIGger:FLEXray:FRAMe:CCRepetition" on page 515
- ":TRIGger:FLEXray:FRAMe:ID" on page 516
- ":TRIGger:FLEXray:FRAMe:TYPE" on page 517
### :TRIGger:GLITch Commands

#### Table 83 :TRIGger:GLITch Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:GLITch:GREater than &lt;greater_than_time&gt;[suffix] (see page 522)</td>
<td>:TRIGger:GLITch:GREater than? (see page 522)</td>
<td>&lt;greater_than_time&gt; ::= floating-point number in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:TRIGger:GLITch:LESSthan &lt;less_than_time&gt;[suffix] (see page 523)</td>
<td>:TRIGger:GLITch:LESSthan? (see page 523)</td>
<td>&lt;less_than_time&gt; ::= floating-point number in NR3 format [suffix] ::= {s</td>
</tr>
<tr>
<td>:TRIGger:GLITch:LEVel &lt;level&gt; [source] (see page 524)</td>
<td>:TRIGger:GLITch:LEVel? (see page 524)</td>
<td>For internal triggers, &lt;level&gt; ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), &lt;level&gt; ::= ±(external range setting) in NR3 format. For digital channels (MSO models), &lt;level&gt; ::= ±8 V. &lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:GLITch:POLarity &lt;polarity&gt; (see page 525)</td>
<td>:TRIGger:GLITch:POLarity? (see page 525)</td>
<td>&lt;polarity&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger:GLITch:QUALifier &lt;qualifier&gt; (see page 526)</td>
<td>:TRIGger:GLITch:QUALifier? (see page 526)</td>
<td>&lt;qualifier&gt; ::= {GREaterthan</td>
</tr>
</tbody>
</table>
### Table 83: TRIGger:GLITch Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| :TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 527) | :TRIGger:GLITch:RANGE? (see page 527) | - `<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 528) | :TRIGger:GLITch:SOURce? (see page 528) | - `<source> ::= {CHANnel<n> | EXTernal} for DSO models`  
- `<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 } for MSO models`  
- `<n> ::= 1-2 or 1-4 in NR1 format` |
522 Agilent InfiniiVision 7000A Series Oscilloscopes Programmer’s Guide

Commands by Subsystem

:TRIGger:GLITch:GREaterthan

(see page 788)

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 468
- ":TRIGger:GLITch:SOURce" on page 528
- ":TRIGger:GLITch:QUALifier" on page 526
- ":TRIGger:MODE" on page 475
:TRIGGER:GLITCH:LESSthan

(see page 788)

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 468
- ":TRIGGER:GLITCH:SOURCE" on page 528
- ":TRIGGER:GLITCH:QUALifier" on page 526
- ":TRIGGER:MODE" on page 475
Commands by Subsystem

:TRIGger:GLITch:LEVel

(see page 788)

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> | DIGital0,...,DIGital15} for MSO models

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the  
active pulse width trigger.

Query Syntax  
:TRIGger:GLITch:LEVel?

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current  
pulse width trigger mode. If all channels are off, the query returns  
"NONE."

Return Format  
<level_argument><NL>

See Also  
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475  
- ":TRIGger:GLITch:SOURce" on page 528
- ":EXTernal:RANGe" on page 262
"TRIGger:GLITch:POLarity"

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 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:GLITch:SOURce" on page 528
Commands by Subsystem

:TRIGger:GLITch:QUALifier

(see page 788)

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 468
• "TRIGger:GLITch:SOURce" on page 528
• "TRIGger:MODE" on page 475
:TRIGger:GLITch:RANGe

(see page 788)

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 468
- ":TRIGger:GLITch:SOURce" on page 528
- ":TRIGger:GLITch:QUALifier" on page 526
- ":TRIGger:MODE" on page 475
Commands by Subsystem

:TRIGger:GLITch:SOURce

(see page 788)

Command Syntax

:TRIGger:GLITch:SOURce <source>

<source> ::= (CHANnel<n> | EXTernal) for the DSO models
<source> ::= {DIGital0,..,DIGital15 | CHANnel<n>} for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:GLITch:SOURce command selects the channel that produces the pulse width trigger.

Query Syntax

:TRIGger:GLITch:SOURce?

The :TRIGger:GLITch:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE."

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:GLITch:LEVel" on page 524
- ":TRIGger:GLITch:POLarity" on page 525
- ":TRIGger:GLITch:QUALifier" on page 526
- ":TRIGger:GLITch:RANGe" on page 527

Example Code

- "Example Code" on page 507
## :TRIGger:I2S Commands

### Table 84 :TRIGger:I2S Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:I2S:ALIGNment</td>
<td>:TRIGger:I2S:ALIGNmen</td>
<td>&lt;setting&gt; ::= {I2S</td>
</tr>
<tr>
<td>&lt;setting&gt; (see page 531)</td>
<td>t? (see page 531)</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:AUDIO</td>
<td>:TRIGger:I2S:AUDio?</td>
<td>&lt;audio_ch&gt; ::= {RIGHT</td>
</tr>
<tr>
<td>&lt;audio_ch&gt; (see page 532)</td>
<td>(see page 532)</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:CLOCK:SL</td>
<td>:TRIGger:I2S:CLOCK:SL</td>
<td>&lt;slope&gt; ::= {NEGative</td>
</tr>
<tr>
<td>OPe &lt;slope&gt; (see page 533)</td>
<td>OPe? (see page 533)</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:PATTERN:DATA</td>
<td>:TRIGger:I2S:PATTern:</td>
<td>&lt;string&gt; ::= &quot;n&quot; where n ::=</td>
</tr>
<tr>
<td>&lt;string&gt; (see page 534)</td>
<td>DATA? (see page 535)</td>
<td>32-bit integer in signed decimal when &lt;base&gt; = DECIMAL</td>
</tr>
<tr>
<td>:TRIGger:I2S:PATTERN:</td>
<td>&lt;string&gt; ::= &quot;nn...n&quot; where n ::=</td>
<td></td>
</tr>
<tr>
<td>FORMAT &lt;base&gt; (see page 536)</td>
<td>FORMAT? (see page 536)</td>
<td>(0</td>
</tr>
<tr>
<td>&lt;base&gt; ::= {BINary</td>
<td>HEX</td>
<td>DECimal}</td>
</tr>
<tr>
<td>:TRIGger:I2S:RANGE</td>
<td>:TRIGger:I2S:RANGE?</td>
<td>&lt;upper&gt; ::= 32-bit integer in signed decimal, &lt;nondecimal&gt;, or</td>
</tr>
<tr>
<td>&lt;upper&gt;,&lt;lower&gt; (see page 537)</td>
<td>(see page 537)</td>
<td>&lt;string&gt;</td>
</tr>
<tr>
<td>&lt;lower&gt; ::= 32-bit integer in signed decimal, &lt;nondecimal&gt;, or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;string&gt; &lt;nondecimal&gt; ::= #Hnn...n where n ::= (0,..,9</td>
<td>A,..,F) for hexadecimal</td>
<td></td>
</tr>
<tr>
<td>&lt;nondecimal&gt; ::= #Bnn...n where n ::= (0</td>
<td>1) for binary</td>
<td></td>
</tr>
<tr>
<td>&lt;string&gt; ::= &quot;0xnn...n&quot; where n ::= (0,..,9</td>
<td>A,..,F) for hexadecimal</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:I2S:RWIDTH</td>
<td>:TRIGger:I2S:RWIDTH?</td>
<td>&lt;receiver&gt; ::= 4-32 in NR1 format</td>
</tr>
<tr>
<td>&lt;receiver&gt; (see page 539)</td>
<td>(see page 539)</td>
<td></td>
</tr>
</tbody>
</table>
### Table 84: :TRIGger:I2S Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:I2S:SOURce:LOCK &lt;source&gt; (see page 540)</td>
<td>:TRIGger:I2S:SOURce:LOCK? (see page 540)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:I2S:SOURce:DATA &lt;source&gt; (see page 541)</td>
<td>:TRIGger:I2S:SOURce:DATA? (see page 541)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:I2S:SOURce:WSELect &lt;source&gt; (see page 542)</td>
<td>:TRIGger:I2S:SOURce:WSELect? (see page 542)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:I2S:TRIGger &lt;operator&gt; (see page 543)</td>
<td>:TRIGger:I2S:TRIGger? (see page 543)</td>
<td>&lt;operator&gt; ::= {EQUal</td>
</tr>
<tr>
<td>:TRIGger:I2S:TWIDth &lt;word_size&gt; (see page 545)</td>
<td>:TRIGger:I2S:TWIDth? (see page 545)</td>
<td>&lt;word_size&gt; ::= 4-32 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:I2S:WSLow &lt;low_def&gt; (see page 546)</td>
<td>:TRIGger:I2S:WSLow? (see page 546)</td>
<td>&lt;low_def&gt; ::= {LEFT</td>
</tr>
</tbody>
</table>
**:TRIGger:I2S:ALIGNment**

(see page 788)

### Command Syntax

```
:TRIGger:I2S:ALIGNment <setting>
```

<setting> ::= (I2S | LJ | RJ)

The :TRIGger:I2S:ALIGNment command selects the data alignment of the I2S bus for the serial decoder and/or trigger when in I2S mode:

- I2S — standard.
- LJ — left justified.
- RJ — right justified.

Note that the word select (WS) polarity is specified separately with the :TRIGger:I2S:WSHigh command.

### Query Syntax

```
:TRIGger:I2S:ALIGNment?
```

The :TRIGger:I2S:ALIGNment? query returns the currently selected I2S data alignment.

### Return Format

```
<setting><NL>
```

<setting> ::= (I2S | LJ | RJ)

### See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:CLOCk:SLOPe" on page 533
- ":TRIGger:I2S:RWIDth" on page 539
- ":TRIGger:I2S:TWIDth" on page 545
- ":TRIGger:I2S:WSLow" on page 546
:TRIGger:I2S:AUDio

(see page 788)

Command Syntax :TRIGger:I2S:AUDio <audio_ch>

<audio_ch> ::= (RIGHt | LEFT | EITHer)

The :TRIGger:I2S:AUDio command specifies the audio channel to trigger on:

- RIGHt – right channel.
- LEFT – left channel.
- EITHer – right channel.

Query Syntax :TRIGger:I2S:AUDio?

The :TRIGger:I2S:AUDio? query returns the current audio channel for the I2S trigger.

Return Format <audio_ch><NL>

<audio_ch> ::= (RIGH | LEFT | EITH)

See Also

- "Introduction to :TRIGger Commands" on page 468
- "::TRIGger:I2S::TRIGger" on page 543
**:TRIGger:I2S:CLoCk:SLOPe**

(see page 788)

**Command Syntax**

`:TRIGger:I2S:CLoCk:SLOPe <slope>`

<slope> ::= (NEGative | POSitive)

The :TRIGger:I2S:CLoCk:SLOPe command specifies which edge of the I2S serial clock signal clocks in data.

- NEGative — Falling edge.
- POSitive — Rising edge.

**Query Syntax**

`:TRIGger:I2S:CLoCk:SLOPe?`

The :TRIGger:I2S:CLoCk:SLOPe? query returns the current I2S clock slope setting.

**Return Format**

<slope><NL>

<slope> ::= (NEG | POS)

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:ALIGNment" on page 531
- ":TRIGger:I2S:RWIDTH" on page 539
- ":TRIGger:I2S:TWIDTH" on page 545
- ":TRIGger:I2S:WSLow" on page 546
:TRIGger:I2S:PATTern:DATA

(see page 788)

Command Syntax

:TRIGger:I2S:PATTern:DATA <string>

<string> ::= "n" where n ::= 32-bit integer in signed 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

<string> is specified with the :TRIGger:I2S:PATTern:FORMat command. The default <base> is DECimal.

The :TRIGger:I2S:PATTern:DATA command specifies the I2S trigger data pattern searched for in each I2S message.

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 :TRIGger:I2S:PATTern:DATA command specifies the I2S trigger data pattern used by the EQUal, NOTEqual, GREaterthan, and LESSthan trigger conditions. If the GREaterthan or LESSthan trigger condition is selected, the bits specified to be masked off ("X") will be interpreted as 0’s.

NOTE

The length of the trigger data value is determined by the :TRIGger:I2S:RWIDth and :TRIGger:I2S:TWIDth commands. When the receiver word size is less than the transmitter word size, the data length is equal to the receiver word size. When the receiver word size is greater than the transmitter word size, the data length is equal to the transmitter word size.
NOTE
If more bits are sent for <string> than the specified trigger data length, the most significant bits will be truncated. If the word 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
:TRIGger:I2S:PATTern:DATA?

The :TRIGger:I2S:PATTern:DATA? query returns the currently specified I2S trigger data pattern.

Return Format
<string><NL>

See Also
• "Introduction to :TRIGger Commands" on page 468
• ":TRIGger:I2S:PATTern:FORMat" on page 536
• ":TRIGger:I2S:TRIGger" on page 543
• ":TRIGger:I2S:RWIDth" on page 539
• ":TRIGger:I2S:TWIDth" on page 545
• ":TRIGger:I2S:AUDio" on page 532
:TRIGger:I2S:PATTern:FORMat

(see page 788)

Command Syntax

:TRIGger:I2S:PATTern:FORMat <base>

<base> ::= {BINary | HEX | DECimal}

The :TRIGger:I2S:PATTern:FORMat command sets the entry (and query) number base used by the :TRIGger:I2S:PATTern:DATA command. The default <base> is DECimal.

Query Syntax

:TRIGger:I2S:PATTern:FORMat?

The :TRIGger:I2S:PATTern:FORMat? query returns the currently set number base for I2S pattern data.

Return Format

<base><NL>

<base> ::= {BIN | HEX | DEC}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:AUDio" on page 532
- ":TRIGger:I2S:TRIGger" on page 543
:TRIGger:I2S:RANGe

Command Syntax

:TRIGger:I2S:RANGe <upper>,<lower>

<upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string>
<lower> ::= 32-bit integer in signed decimal, <nondecimal> or <string>

<nondecimal> ::= #Hnn...n where n ::= (0,...,9 | A,...,F) for hexadecimal
<nondecimal> ::= #Bnn...n where n ::= (0 | 1) for binary
<string> ::= "0xnn...n" where n ::= (0,...,9 | A,...,F) for hexadecimal

The :TRIGger:I2S:RANGe command sets the upper and lower range boundaries used by the INRange, OUTRange, INCreasing, and DECreasing trigger conditions. You can enter the parameters in any order — the smaller value becomes the <lower> and the larger value becomes the <upper>.

Note that for INCreasing and DECreasing, the <upper> and <lower> values correspond to the "Armed" and "Trigger" softkeys.

Query Syntax

:TRIGger:I2S:RANGe?

The :TRIGger:I2S:RANGe? query returns the currently set upper and lower range boundaries.

Return Format

<upper>,<lower><NL>

<upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string>
<lower> ::= 32-bit integer in signed 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

See Also

• "Introduction to :TRIGger Commands" on page 468
Commands by Subsystem

- ".:TRIGger:I2S:TRIGger" on page 543
- ".:TRIGger:I2S:RWIDTH" on page 539
- ".:TRIGger:I2S:TWIDTH" on page 545
- ".:TRIGger:I2S:WSLOW" on page 546
:TRIGger:I2S:RWIDth

N (see page 788)

Command Syntax

:TRIGger:I2S:RWIDth <receiver>

<receiver> ::= 4-32 in NR1 format

The :TRIGger:I2S:RWIDth command sets the width of the receiver (decoded) data word in I2S anywhere from 4 bits to 32 bits.

Query Syntax

:TRIGger:I2S:RWIDth?

The :TRIGger:I2S:RWIDth? query returns the currently set I2S receiver data word width.

Return Format

<receiver><NL>

<receiver> ::= 4-32 in NR1 format

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:ALIGNment" on page 531
- ":TRIGger:I2S:CLOCk:SLOPe" on page 533
- ":TRIGger:I2S:TWIDth" on page 545
- ":TRIGger:I2S:WSLow" on page 546
:TRIGger:I2S:SOURce:CLOCk

(see page 788)

Command Syntax

:TRIGger:I2S:SOURce:CLOCk <source>

<source> ::= {CHANnel<n> | EXternal} for the DSO models
<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:I2S:SOURce:CLOCk controls which signal is used as the serial clock (SCLK) source by the serial decoder and/or trigger when in I2S mode.

Query Syntax

:TRIGger:I2S:SOURce:CLOCk?

The :TRIGger:I2S:SOURce:CLOCk? query returns the current source for the I2S serial clock (SCLK).

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:SOURce:DATA" on page 541
- ":TRIGger:I2S:SOURce:WSELeCt" on page 542
Commands by Subsystem 5

:TRIGger:I2S:SOURce:DATA

(see page 788)

**Command Syntax**

:TRIGger:I2S:SOURce:DATA <source>

<source> ::= {CHANnel<n> | EXternal} for the DSO models
<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:I2S:SOURce:DATA command controls which signal is used as the serial data (SDATA) source by the serial decoder and/or trigger when in I2S mode.

**Query Syntax**

:TRIGger:I2S:SOURce:DATA?


**Return Format**

<source><NL>

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:SOURce:CLOCk" on page 540
- ":TRIGger:I2S:SOURce:WSELect" on page 542
Command Syntax

:TRIGger:I2S:SOURce:WSELect <source>

<source> ::= \{CHANnel<n> | EXTernal\} for the DSO models
<source> ::= \{CHANnel<n> | DIGital0,...,DIGital15\} for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:I2S:SOURce:WSELect command controls which signal is used as the word select (WS) source by the serial decoder and/or trigger when in I2S mode.

Query Syntax

:TRIGger:I2S:SOURce:WSELect?

The :TRIGger:I2S:SOURce:WSELect? query returns the current source for I2S word select (WS).

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:SOURce:CLOCK" on page 540
- ":TRIGger:I2S:SOURce:DATA" on page 541
**Command Syntax**

```
:TRIGger:I2S:TRIGger <operator>

<operator> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange
| OUTRange | INCreasing | DECreasing}
```

The :TRIGger:I2S:TRIGger command sets the I2S trigger operator:

- **EQUal**— triggers on the specified audio channel's data word when it equals the specified word.
- **NOTequal** — triggers on any word other than the specified word.
- **LESSthan** — triggers when the channel's data word is less than the specified value.
- **GREaterthan** — triggers when the channel's data word is greater than the specified value.
- **INRange** — enter upper and lower values to specify the range in which to trigger.
- **OUTRange** — enter upper and lower values to specify range in which trigger will not occur.
- **INCreasing** — triggers when the data value makes a certain increase over time and the specified value is met or exceeded. Use the :TRIGger:I2S:RANGe command to set "Trigger" and "Armed" values. The "Trigger" value is the value that must be met or exceeded to cause the trigger. The "Armed" value is the value the data must go below in order to re-arm the oscilloscope (ready it to trigger again).
- **DECreasing** — similar to INCreasing except the trigger occurs on a certain decrease over time and the "Trigger" data value is less than the "Armed" data value.

**Query Syntax**

```
:TRIGger:I2S:TRIGger?
```
The :TRIGger:I2S:TRIgger? query returns the current I2S trigger operator.

**Return Format**

```
<operator><NL>
```

<operator> ::= (EQU | NOT | LESS | GRE | INR | OUTR | INCR | DECR)

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- "TRIGger:I2S:AUDio" on page 532
- "TRIGger:I2S:RANGe" on page 537
- "TRIGger:I2S:PARTtern:FORMat" on page 536
:TRIGger:I2S:TWIDth

(see page 788)

Command Syntax

:TRIGger:I2S:TWIDth <word_size>

<word_size> ::= 4-32 in NR1 format

The :TRIGger:I2S:TWIDth command sets the width of the transmitted data word in I2S anywhere from 4 bits to 32 bits.

Query Syntax

:TRIGger:I2S:TWIDth?

The :TRIGger:I2S:TWIDth? query returns the currently set I2S transmitted data word width.

Return Format

<word_size><NL>

<word_size> ::= 4-32 in NR1 format

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:I2S:ALIGnment" on page 531
- ":TRIGger:I2S:CLOck:SLOPe" on page 533
- ":TRIGger:I2S:RWIDth" on page 539
- ":TRIGger:I2S:WSLow" on page 546
### :TRIGger:I2S:WSLow

(see page 788)

**Command Syntax**

```plaintext
:TRIGger:I2S:WSLow <low_def>

<low_def> ::= \{LEFT | RIGHT\}
```

The :TRIGger:I2S:WSLow command selects the polarity of the word select (WS) signal:

- **LEFT**— a word select (WS) state of low indicates left channel data is active on the I2S bus, and a WS state of high indicates right channel data is active on the bus.
- **RIGHT**— a word select (WS) state of low indicates right channel data is active on the I2S bus, and a WS state of high indicates left channel data is active on the bus.

**Query Syntax**

```plaintext
:TRIGger:I2S:WSLow?
```

The :TRIGger:I2S:WSLow? query returns the currently selected I2S word select (WS) polarity.

**Return Format**

```plaintext
<low_def><NL>
<low_def> ::= \{LEFT | RIGHT\}
```

See Also

- "Introduction to :TRIGger Commands" on page 468
- ".:TRIGger:I2S:ALIGNment" on page 531
- ".:TRIGger:I2S:CLOCk:SLOPe" on page 533
- ".:TRIGger:I2S:RWIDth" on page 539
- ".:TRIGger:I2S:TWIDth" on page 545
### :TRIGger:IIC Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:IIC:PATTern:ADDRess &lt;value&gt; (see page 548)</td>
<td>:TRIGger:IIC:PATTern:ADDRess? (see page 548)</td>
<td>:&lt;value&gt; ::= integer or &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;string&gt; ::= &quot;0xnn&quot; n ::= [0,\ldots,9</td>
</tr>
<tr>
<td>:TRIGger:IIC:PATTern:DATA &lt;value&gt; (see page 549)</td>
<td>:TRIGger:IIC:PATTern:DATA? (see page 549)</td>
<td>:&lt;value&gt; ::= integer or &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;string&gt; ::= &quot;0xnn&quot; n ::= [0,\ldots,9</td>
</tr>
<tr>
<td>:TRIGger:IIC:PATTern:DATa2 &lt;value&gt; (see page 550)</td>
<td>:TRIGger:IIC:PATTern:DATa2? (see page 550)</td>
<td>:&lt;value&gt; ::= integer or &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;string&gt; ::= &quot;0xnn&quot; n ::= [0,\ldots,9</td>
</tr>
<tr>
<td>:TRIGger:IIC[:SOURce]:CLOCk &lt;source&gt; (see page 551)</td>
<td>:TRIGger:IIC[:SOURce]:CLOCk? (see page 551)</td>
<td>:&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:IIC[:SOURce]:DATA &lt;source&gt; (see page 552)</td>
<td>:TRIGger:IIC[:SOURce]:DATA? (see page 552)</td>
<td>:&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:IIC:TRIGger:QUALifier &lt;value&gt; (see page 553)</td>
<td>:TRIGger:IIC:TRIGger:QUALifier? (see page 553)</td>
<td>:&lt;value&gt; ::= {EQUal</td>
</tr>
<tr>
<td>:TRIGger:IIC:TRIGger[:TYPE] &lt;type&gt; (see page 554)</td>
<td>:TRIGger:IIC:TRIGger[:TYPE]? (see page 554)</td>
<td>:&lt;type&gt; ::= {STARt</td>
</tr>
</tbody>
</table>
548 Agilent InfiniiVision 7000A Series Oscilloscopes Programmer's Guide

Commands by Subsystem

:TRIGger:IIC:PATTern:ADDRess

(see page 788)

Command Syntax

:TRIGger:IIC:PATTern:ADDRess <value>

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

The :TRIGger:IIC:PATTern:ADDRess command sets the address for IIC data. The address can range from 0x00 to 0x7F (7-bit) or 0x3FF (10-bit) hexadecimal. Use the don't care address (-1 or 0xFFFFFFFF) to ignore the address value.

Query Syntax

:TRIGger:IIC:PATTern:ADDRess?

The :TRIGger:IIC:PATTern:ADDRess? query returns the current address for IIC data.

Return Format

<value><NL>

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

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:IIC:PATTern:DATA" on page 549
- ":TRIGger:IIC:PATTern:DATa2" on page 550
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 554
:TRIGger:IIC:PATTern:DATA

(see page 788)

Command Syntax

:TRIGger:IIC:PATTern:DATA <value>

<value> ::= integer or <string>

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

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

Query Syntax

:TRIGger:IIC:PATTern:DATA?

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

Return Format

<value><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:IIC:PATTern:ADDRess" on page 548
- ":TRIGger:IIC:PATTern:DATa2" on page 550
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 554
:TRIGger:IIC:PATTern:DATa2

(see page 788)

Command Syntax  
:TRIGger:IIC:PATTern:DATa2 <value>

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

The :TRIGger:IIC:PATTern:DATa2 command sets IIC data 2. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax  
:TRIGger:IIC:PATTern:DATa2?


Return Format  
<value><NL>

See Also  
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:IIC:PATTern:ADDRess" on page 548
- ":TRIGger:IIC:PATTern:DATA" on page 549
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 554
**:TRIGger:IIC[:SOURce]:CLOCk**

(see page 788)

**Command Syntax**

```plaintext
:TRIGger:IIC:[SOURce:]CLOCk <source>
```

- `<source>` ::= `(CHANnel<n> | EXTernal)` for the DSO models
- `<source>` ::= `(CHANnel<n> | DIGital0,...,DIGital15)` for the MSO models
- `<n>` ::= `{ 1 | 2 | 3 | 4 }` for the four channel oscilloscope models
- `<n>` ::= `{ 1 | 2 }` for the two channel oscilloscope models

The **:TRIGger:IIC:[SOURce:]CLOCk** command sets the source for the IIC serial clock (SCL).

**Query Syntax**

```plaintext
:TRIGger:IIC:[SOURce:]CLOCk?
```

The **:TRIGger:IIC:[SOURce:]CLOCk?** query returns the current source for the IIC serial clock.

**Return Format**

```
<source><NL>
```

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:IIC[:SOURce]:DATA" on page 552
**:TRIGger:IIC[:SOURce]:DATA**

(see page 788)

**Command Syntax**

`:TRIGger:IIC:[SOURce:]DATA <source>  

<source> ::= (CHANnel<n> | EXTernal) for the DSO models  
<source> ::= (CHANnel<n> | DIGital0,...,DIGital15) for the MSO models  
<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models  
<n> ::= (1 | 2) for the two channel oscilloscope models  

The :TRIGger:IIC:[SOURce:]DATA command sets the source for IIC serial data (SDA).

**Query Syntax**

`:TRIGger:IIC:[SOURce:]DATA?  

The :TRIGger:IIC:[SOURce:]DATA? query returns the current source for IIC serial data.

**Return Format**

<source><NL>

**See Also**

- "Introduction to :TRIGger Commands" on page 468  
- ":TRIGger:IIC[:SOURce]:CLOCk" on page 551
:TRIGger:IIC:TRIGger:QUALifier

(see page 788)

Command Syntax

:TRIGger:IIC:TRIGger:QUALifier <value>

<value> ::= (EQUal | NOTequal | LESSthan | GREaterthan)

The :TRIGger:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEngom.

Query Syntax

:TRIGger:IIC:TRIGger:QUALifier?

The :TRIGger:IIC:TRIGger:QUALifier? query returns the current IIC data qualifier value.

Return Format

<value><NL>

<value> ::= (EQUal | NOTequal | LESSthan | GREaterthan)

See Also

• "Introduction to :TRIGger Commands" on page 468
• ":TRIGger:MODE" on page 475
• ":TRIGger:IIC:TRIGger[:TYPE]" on page 554
**:TRIGger:IIC:TRIGger[:TYPE]**

(see page 788)

**Command Syntax**

:TRIGger:IIC:TRIGger[:TYPE] <value>

<value> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10 | NACKnowledge | ANACKnowledge | R7Data2 | W7Data2 | RESTart}

The :TRIGger:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- **STARt** — Start condition.
- **STOP** — Stop condition.
- **READ7** — 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- **R7Data2** — 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- **READEprom** — EEPROM data read.
- **WRITe7** — 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- **W7Data2** — 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).
- **WRITe10** — 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).
- **NACKnowledge** — Missing acknowledge.
- **ANACKnowledge** — Address with no acknowledge.
- **RESTart** — Another start condition occurs before a stop condition.

**NOTE**

The short form of READ7 (READ7), READEprom (READe), WRITe7 (WRITe7), and WRITe10 (WRITe10) do not follow the defined Long Form to Short Form Truncation Rules (see page 790).

**Query Syntax**

:TRIGger:IIC:TRIGger[:TYPE]?

The :TRIGger:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

**Return Format**

<value><NL>

<value> ::= {STAR | STOP | READ7 | READe | WRITe | WRITe10 | NACK | ANAC | R7D2 | W7D2 | REST}

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
• ".TRIGger:IIC:PATTern:ADDRess" on page 548
• ".TRIGger:IIC:PATTern:DATA" on page 549
• ".TRIGger:IIC:PATTern:DATa2" on page 550
• ".TRIGger:IIC:TRIGger:QUALifier" on page 553
• "Long Form to Short Form Truncation Rules" on page 790
# :TRIGger:LIN Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:LIN:ID &lt;value&gt; (see page 558)</td>
<td>:TRIGger:LIN:ID? (see page 558)</td>
<td>&lt;value&gt; ::= 7-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; from 0-63 or 0x00-0x3f (with Option AMS) &lt;nondecimal&gt; ::= #Hnn where n ::= {0,...,9</td>
</tr>
<tr>
<td>:TRIGger:LIN:PATTERN:DATA &lt;string&gt; (see page 559)</td>
<td>:TRIGger:LIN:PATTERN:DATA? (see page 560)</td>
<td>&lt;string&gt; ::= &quot;n&quot; where n ::= 32-bit integer in signed decimal when &lt;base&gt; = DECimal &lt;string&gt; ::= &quot;nn...n&quot; where n ::= {0</td>
</tr>
<tr>
<td>:TRIGger:LIN:PATTERN:DATA:LENGTH &lt;length&gt; (see page 561)</td>
<td>:TRIGger:LIN:PATTERN:DATA:LENGTH? (see page 561)</td>
<td>&lt;length&gt; ::= integer from 1 to 8 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:LIN:PATTERN:FORMAT &lt;base&gt; (see page 562)</td>
<td>:TRIGger:LIN:PATTERN:FORMAT? (see page 562)</td>
<td>&lt;base&gt; ::= {BINary</td>
</tr>
<tr>
<td>:TRIGger:LIN:SAMPLEpoint:INT &lt;value&gt; (see page 563)</td>
<td>:TRIGger:LIN:SAMPLEpoint:INT? (see page 563)</td>
<td>&lt;value&gt; ::= {60</td>
</tr>
<tr>
<td>:TRIGger:LIN:SIGNal:BAUDrate &lt;baudrate&gt; (see page 564)</td>
<td>:TRIGger:LIN:SIGNal:BAUDrate? (see page 564)</td>
<td>&lt;baudrate&gt; ::= integer from 2400 to 625000 in 100 b/s increments</td>
</tr>
<tr>
<td>:TRIGger:LIN:SOURce &lt;source&gt; (see page 565)</td>
<td>:TRIGger:LIN:SOURce? (see page 565)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:LIN:STANdard &lt;std&gt; (see page 566)</td>
<td>:TRIGger:LIN:STANdard? (see page 566)</td>
<td>&lt;std&gt; ::= {LIN13</td>
</tr>
</tbody>
</table>
Table 86 :TRIGger:LIN Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:LIN:SYNCbreak &lt;value&gt; (see page 567)</td>
<td>:TRIGger:LIN:SYNCbreak? (see page 567)</td>
<td>&lt;value&gt; ::= integer = {11</td>
</tr>
<tr>
<td>:TRIGger:LIN:TRIGger &lt;condition&gt; (see page 568)</td>
<td>:TRIGger:LIN:TRIGger? (see page 568)</td>
<td>&lt;condition&gt; ::= {SYNCbreak} (without Option AMS) &lt;condition&gt; ::= {SYNCbreak</td>
</tr>
</tbody>
</table>
Commands by Subsystem

:TRIGger:LIN:ID

(see page 788)

Command Syntax

:TRIGger:LIN:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>
from 0-63 or 0x00-0x3f

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

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

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

The :TRIGger:LIN:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

NOTE

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Setting the ID to a value of "-1" results in "0XX" which is equivalent to all IDs.

Query Syntax

:TRIGger:LIN:ID?

The :TRIGger:LIN:ID? query returns the current LIN identifier setting.

Return Format

<value><NL>

<value> ::= integer in decimal

Errors

- 

See Also

- "Introduction to :TRIGger Commands" on page 468
- 
- "TRIGger:MODE" on page 475
- "TRIGger:LIN:TRIGger" on page 568
- "TRIGger:LIN:SIGNal:DEFinition" on page 741
- "TRIGger:LIN:SOURce" on page 565
**:TRIGger:LIN:PATTern:DATA**

(see page 788)

**Command Syntax**

```
:TRIGger:LIN:PATTern:DATA <string>
```

- `<string>` ::= "n" where n ::= 32-bit integer in signed 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 :TRIGger:LIN:PATTern:FORMat command. The default `<base>` is DECimal.

The :TRIGger:LIN: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 :TRIGger:LIN:PATTern:DATA command specifies the LIN trigger data pattern used by the DATA trigger condition. This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**NOTE**

- The length of the trigger data value is determined by the :TRIGger:LIN: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**

:TRIGger:LIN:PATTern:DATA?

The :TRIGger:LIN:PATTern:DATA? query returns the currently specified LIN trigger data pattern.

**Return Format**

<string><NL>

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:LIN:PATTern:FORMat" on page 562
- ":TRIGger:LIN:TRIGger" on page 568
- ":TRIGger:LIN:PATTern:DATA:LENGth" on page 561
Commands by Subsystem

:TRIGger:LIN:PATTern:DATA:LENGth

Command Syntax

:TRIGger:LIN:PATTern:DATA:LENGth <length>

(length) ::= integer from 1 to 8 in NR1 format

The :TRIGger:LIN: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 0 bytes to 8 bytes (64 bits). The value for these bytes is set by the :TRIGger:LIN:PATTern:DATA command.

NOTE

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

:TRIGger:LIN:PATTern:DATA:LENGth?

The :TRIGger:LIN: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 747

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:LIN:PATTern:DATA" on page 559
- ":TRIGger:LIN:SOURce" on page 565
:TRIGger:LIN:PA TTern:FORMat

Command Syntax

:TRIGger:LIN:PA TTern:FORMat <base>

<base> ::= {BINary | HEX | DECimal}

The :TRIGger:LIN:PA TTern:FORMat command sets the entry (and query) number base used by the :TRIGger:LIN:PA TTern:DATA command. The default <base> is DECimal.

Query Syntax

:TRIGger:LIN:PA TTern:FORMat?

The :TRIGger:LIN:PA TTern: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 468
- ".TRIGger:LIN:PA TTern:DATA" on page 559
- ".TRIGger:LIN:PA TTern:DATA:LENGth" on page 561

NOTE
This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.
**:TRIGger:LIN:SAMPlepoint**

(see page 788)

**Command Syntax**

```
:TRIGger:LIN:SAMPlepoint <value>
```

<value><NL>

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

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

**NOTE**

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

**Query Syntax**

```
:TRIGger:LIN:SAMPlepoint?
```

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

**Return Format**

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

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

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:LIN:TRIGger" on page 568
Commands by Subsystem

:TRIGger:LIN:SIGNal:BAUDrate

(see page 788)

Command Syntax

:TRIGger:LIN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments

The :TRIGger: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

:TRIGger:LIN:SIGNal:BAUDrate?

The :TRIGger: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 468
- "TRIGger:MODE" on page 475
- "TRIGger:LIN:TRIGger" on page 568
- "TRIGger:LIN:SIGNal:DEFinition" on page 741
- "TRIGger:LIN:SOURce" on page 565
:\texttt{TRIGger:LIN:SOURce}

\textbf{Command Syntax}
\begin{itemize}
\item \texttt{:TRIGger:LIN:SOURce <source>}
\end{itemize}

\begin{itemize}
\item \texttt{<source> ::= \{\text{CHANnel}\texttt{n} | EXTernal\} for the DSO models}
\item \texttt{<source> ::= \{\text{CHANnel}\texttt{n} | DIGital0,...,DIGital15\} for the MSO models}
\item \texttt{<n> ::= \{1 | 2 | 3 | 4\} for the four channel oscilloscope models}
\item \texttt{<n> ::= \{1 | 2\} for the two channel oscilloscope models}
\end{itemize}

The :TRIGger:LIN:SOURce command sets the source for the LIN signal.

\textbf{Query Syntax}
\begin{itemize}
\item \texttt{:TRIGger:LIN:SOURce?}
\end{itemize}

The :TRIGger:LIN:SOURce? query returns the current source for the LIN signal.

\textbf{Return Format}
\begin{itemize}
\item \texttt{\langle source\rangle<NL>}
\end{itemize}

\textbf{See Also}
\begin{itemize}
\item "Introduction to :TRIGger Commands" on page 468
\item ":TRIGger:MODE" on page 475
\item ":TRIGger:LIN:TRIGger" on page 568
\item ":TRIGger:LIN:SIGNal:DEFinition" on page 741
\end{itemize}
### \textbf{:TRIGger:LIN:STANdard}

(see page 788)

#### Command Syntax

\texttt{:TRIGger:LIN:STANdard <std>}

\texttt{<std> ::= \{LIN13 | LIN20\}}

The \texttt{:TRIGger:LIN:STANdard} command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

#### Query Syntax

\texttt{:TRIGger:LIN:STANdard?}

The \texttt{:TRIGger:LIN:STANdard?} query returns the current LIN standard setting.

#### Return Format

\texttt{<std><NL>}

\texttt{<std> ::= \{LIN13 | LIN20\}}

#### See Also

- "Introduction to :TRIGger Commands" on page 468
- ":\texttt{TRIGger:MODE}" on page 475
- ":\texttt{TRIGger:LIN:SIGNal:DEFinition}" on page 741
- ":\texttt{TRIGger:LIN:SOURce}" on page 565
:TRIGger:LIN:SYNCbreak

N (see page 788)

**Command Syntax**

:TRIGger:LIN:SYNCbreak <value>

<value> ::= integer = {11 | 12 | 13}

The :TRIGger:LIN:SYNCbreak command sets the length of the LIN sync break to be greater than or equal to 11, 12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.

**Query Syntax**

:TRIGger:LIN:SYNCbreak?

The :TRIGger:LIN:STANdard? query returns the current LIN sync break setting.

**Return Format**

<value><NL>

<value> ::= {11 | 12 | 13}

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:LIN:SIGNal:DEFinition" on page 741
- ":TRIGger:LIN:SOURce" on page 565
**:TRIGger:LIN:TRIGger**

(see page 788)

**Command Syntax**

`:TRIGger:LIN:TRIGger <condition>`

`<condition> ::= {SYNCbreak | ID | DATA}`

The :TRIGger:LIN:TRIGger command sets the LIN trigger condition to be:

- SYNCbreak — Sync Break.
- ID — Frame ID.
- DATA — Frame ID and Data.

Use the :TRIGger:LIN:ID command to specify the frame ID.

Use the :TRIGger:LIN:PATTern:DATA:LENGth and :TRIGger:LIN:PATTern:DATA commands to specify the data string length and value.

**NOTE**

The ID and DATA options are available when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax**

`:TRIGger:LIN:TRIGger?`

The :TRIGger:LIN:TRIGger? query returns the current LIN trigger value.

**Return Format**

`<condition><NL>`

`<condition> ::= {SYNC | ID | DATA}`

**Errors**

"-241, Hardware missing" on page 747

**See Also**

"Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:LIN:ID" on page 558
- ":TRIGger:LIN:PATTern:DATA:LENGth" on page 561
- ":TRIGger:LIN:PATTern:DATA" on page 559
- ":TRIGger:LIN:SIGNal:DEFinition" on page 741
- ":TRIGger:LIN:SOURce" on page 565
## :TRIGger:M1553 Commands

### Table 87  :TRIGger:M1553 Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:M1553:AUTosetup (see page 570)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>:TRIGger:M1553:PATTern &lt;string&gt; (see page 571)</td>
<td>:TRIGger:M1553:PATTern:DATA? (see page 571)</td>
<td>&lt;string&gt; ::= &quot;nn...n&quot; where n ::= (0</td>
</tr>
<tr>
<td>:TRIGger:M1553:RTA &lt;value&gt; (see page 572)</td>
<td>:TRIGger:M1553:RTA? (see page 572)</td>
<td>&lt;value&gt; ::= 5-bit integer in decimal, &lt;nondecimal&gt;, or &lt;string&gt; from 0-31 &lt;nondecimal&gt; ::= #Hnn where n ::= (0,...9</td>
</tr>
<tr>
<td>:TRIGger:M1553:SOURce:LOWer &lt;source&gt; (see page 573)</td>
<td>:TRIGger:M1553:SOURce:LOWer? (see page 573)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= (2</td>
</tr>
<tr>
<td>:TRIGger:M1553:SOURce:UPPer &lt;source&gt; (see page 574)</td>
<td>:TRIGger:M1553:SOURce:UPPer? (see page 574)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= (1</td>
</tr>
<tr>
<td>:TRIGger:M1553:TYPE &lt;type&gt; (see page 575)</td>
<td>:TRIGger:M1553:TYPE? (see page 575)</td>
<td>&lt;type&gt; ::= {DSTart</td>
</tr>
</tbody>
</table>
:**TRIGger:M1553:AUTosetup**

(see page 788)

**Command Syntax**

```
:TRIGger:M1553:AUTosetup
```

The :TRIGger:M1553:AUTosetup command copies the position, volts/div, and probe attenuation from the upper threshold channel to the lower threshold channel, sets the upper/lower trigger levels to +/-500 mV, turns on serial decode, and sets the trigger mode to M1553.

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:M1553:SOURce:UPPer" on page 574
Commands by Subsystem

:TRIGger:M1553:PATTern:DATA

(see page 788)

Command Syntax

:TRIGger:M1553:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X}

The :TRIGger:M1553:PATTern:DATA command sets the 11 bits to trigger on if the trigger type has been set to RTA11 (RTA + 11 Bits) using the :TRIG:M1553:TYPE command.

Query Syntax

:TRIGger:M1553:PATTern:DATA?

The :TRIGger:M1553:PATTern:DATA? query returns the current 11-bit setting.

Return Format

<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:M1553:TYPE" on page 575
**:TRIGger:M1553:RTA**

(see page 788)

**Command Syntax**

```
:TRIGger:M1553:RTA <value>
```

`<value>` ::= 5-bit integer in decimal, `<nondecimal>`, or `<string>` from 0-31

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

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

The :TRIGger:M1553:RTA command sets the Remote Terminal Address (RTA) to trigger on if the trigger type has been set to RTA using the :TRIG:M1553:TYPE command.

**Query Syntax**

```
:TRIGger:M1553:RTA?
```

The :TRIGger:M1553:RTA? query returns the current TV trigger line number setting.

**Return Format**

`<value><NL>` in nondecimal format

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:M1553:TYPE" on page 575
:TRIGger:M1553:SOURce:LOWer

(see page 788)

Command Syntax

:TRIGger:M1553:SOURce:LOWer <source>

<source> ::= {CHANnel<n>}

<n> ::= (2 | 4)

The :TRIGger:M1553:SOURce:LOWer command controls which signal is used as the Lower Threshold Channel source by the serial decoder and/or trigger when in MIL-1553 mode.

Query Syntax

:TRIGger:M1553:SOURce:LOWer?

The :TRIGger:M1553:SOURce:LOWer? query returns the currently set Lower Threshold Channel source.

Return Format

<source><NL>

<source> ::= {CHAN<n>}

<n> ::= (2 | 4)

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:M1553:SOURce:UPPer" on page 574
Commands by Subsystem

:TRIGger:M1553:SOURce:UPPer

Command Syntax
:TRIGger:M1553:SOURce:UPPer <source>

<source> ::= {CHANnel<n>}

<n> ::= {1 | 3}

The :TRIGger:M1553:SOURce:UPPer command controls which signal is used as the Upper Threshold Channel source by the serial decoder and/or trigger when in MIL-1553 mode.

Query Syntax
:TRIGger:M1553:SOURce:UPPer?

The :TRIGger:M1553:SOURce:UPPer? query returns the currently set Upper Threshold Channel source.

Return Format
<source><NL>

<source> ::= {CHAN<n>}

<n> ::= {1 | 3}

See Also
* "Introduction to :TRIGger Commands" on page 468
* ":TRIGger:MODE" on page 475
* ":TRIGger:M1553:SOURce:LOWer" on page 573
:TRIGger:M1553:TYPE

Command Syntax

:TRIGger:M1553:TYPE <type>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:M1553:RTA" on page 572
- ":TRIGger:M1553:PA" on page 571
- ":TRIGger:PA" on page 475
### :TRIGger:SEQuence Commands

#### Table 88 :TRIGger:SEQuence Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:SEQuence:COUNT &lt;count&gt; (see page 577)</td>
<td>:TRIGger:SEQuence:COUNT? (see page 577)</td>
<td>&lt;count&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:SEQuence:EDGE{1</td>
<td>2} &lt;source&gt;, &lt;slope&gt; (see page 578)</td>
<td>:TRIGger:SEQuence:EDGE{1</td>
</tr>
<tr>
<td>:TRIGger:SEQuence:FIND &lt;value&gt; (see page 579)</td>
<td>:TRIGger:SEQuence:FIND? (see page 579)</td>
<td>&lt;value&gt; ::= {PATTern1,ENTERed</td>
</tr>
<tr>
<td>:TRIGger:SEQuence:PATTERN{1</td>
<td>2} &lt;value&gt;, &lt;mask&gt; (see page 580)</td>
<td>:TRIGger:SEQuence:PATTERN{1</td>
</tr>
<tr>
<td>:TRIGger:SEQuence:RESET &lt;value&gt; (see page 581)</td>
<td>:TRIGger:SEQuence:RESET? (see page 581)</td>
<td>&lt;value&gt; ::= {NONE</td>
</tr>
<tr>
<td>:TRIGger:SEQuence:TIMER &lt;time_value&gt; (see page 582)</td>
<td>:TRIGger:SEQuence:TIMER? (see page 582)</td>
<td>&lt;time_value&gt; ::= time from 10 ns to 10 seconds in NR3 format</td>
</tr>
<tr>
<td>:TRIGger:SEQuence:TRIGGER &lt;value&gt; (see page 583)</td>
<td>:TRIGger:SEQuence:TRIGGER? (see page 583)</td>
<td>&lt;value&gt; ::= {PATTern2,ENTERed</td>
</tr>
</tbody>
</table>
**:TRIGger:SEQUence:COUNt**

(see page 788)

**Command Syntax**

`:TRIGger:SEQUence:COUNt <count>`

<count> ::= integer in NR1 format

The :TRIGger:SEQUence:COUNt command sets the sequencer edge counter resource. The edge counter is used in the trigger stage to determine the number of edges that must be found before the sequencer generates a trigger.

**Query Syntax**

`:TRIGger:SEQUence:COUNt?`

The :TRIGger:SEQUence:COUNt? query returns the current sequencer edge counter setting.

**Return Format**

<count><NL>

<count> ::= integer in NR1 format

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SEQUence:TRIGger" on page 583
- ":TRIGger:SEQUence:EDGE" on page 578
5 Commands by Subsystem

:TRIGGER:SEQUENCE:EDGE

(see page 788)

Command Syntax

:TRIGGER:SEQUENCE:EDGE{1 | 2} <source>, <slope>

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

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

<slope> ::= {POSitive | NEGative}

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

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

The :TRIGGER:SEQUENCE:EDGE<n> command defines the specified sequencer edge resource according to the specified <source> and <slope>. To disable an edge resource, set its <source> to NONE. In this case, <slope> has no meaning.

Query Syntax

:TRIGGER:SEQUENCE:EDGE{1 | 2}?

The :TRIGGER:SEQUENCE:EDGE<n>? query returns the specified sequencer edge resource setting. If the edge resource is disabled, the returned <source> value is NONE. In this case, the <slope> is undefined.

Return Format

<source>, <slope><NL>

See Also

- "Introduction to :TRIGGER Commands" on page 468
- ":TRIGGER:SEQUENCE:FIND" on page 579
- ":TRIGGER:SEQUENCE:TRIGGER" on page 583
- ":TRIGGER:SEQUENCE:RESET" on page 581
- ":TRIGGER:SEQUENCE:COUNTO" on page 577
:\TRIgger:\SEquence:FIND

(see page 788)

Command Syntax

:\TRIgger:\SEquence:FIND <value>

<value> ::= (PATTern1,ENTERed | PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1)

The \TRIgger:\SEquence:FIND command specifies the find stage of a sequence trigger. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example,"EDGE1,NONE,NONE").

PATTern1 is specified with the\TRIgger:\SEquence:PATTern command. EDGE1 is specified with the \TRIgger:\SEquence:EDGE command.

Query Syntax

:\TRIgger:\SEquence:FIND?

The \TRIgger:\SEquence:FIND? query returns the find stage specification for a sequence trigger. NONE is returned for unused parameters.

Return Format

<find_value><NL>

<find_value> ::= (PATT1,ENT,NONE | PATT1,EXIT,NONE | EDGE1,NONE,NONE | PATT1,AND,EDGE1)

See Also

- "Introduction to \TRIgger Commands" on page 468
- ":\TRIgger:\SEquence:PATTern" on page 580
- ":\TRIgger:\SEquence:EDGE" on page 578
- ":\TRIgger:\SEquence:TRIgger" on page 583
- ":\TRIgger:\SEquence:RESet" on page 581
:TRIGger:SEQuence:PATTern

(see page 788)

Command Syntax

:TRIGger:SEQuence:PATTern{1 | 2} <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

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

The :TRIGger:SEQuence:PATTern<n> command defines the specified sequence pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

<table>
<thead>
<tr>
<th>Oscilloscope Models</th>
<th>Value and Mask Bit Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 analog + 16 digital channels (mixed-signal)</td>
<td>Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.</td>
</tr>
<tr>
<td>2 analog + 16 digital channels (mixed-signal)</td>
<td>Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.</td>
</tr>
<tr>
<td>4 analog channels only</td>
<td>Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.</td>
</tr>
<tr>
<td>2 analog channels only</td>
<td>Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.</td>
</tr>
</tbody>
</table>

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

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

Query Syntax

:TRIGger:SEQuence:PATTern{1 | 2}?

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

Return Format

<value>, <mask><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SEQuence:RESet" on page 581
- "TRIGger:SEQuence:TRIGger" on page 583
- "TRIGger:SEQuence:FIND" on page 579
- ":TRIGger:SEQuence:TRIGger" on page 583
:TRIGger:SEQUence:RESet

(see page 788)

Command Syntax

:TRIGger:SEQUence:RESet <value>

<value> ::= (NONE | PATTERN1,ENTERed | PATTERN1,EXITed | EDGE1
    | PATTERN1,AND,EDGE1 | PATTERN2,ENTERed | PATTERN2,EXITed
    | EDGE2 | TIMer)

Values used in find and trigger stages are not available. EDGE2 is not available if EDGE2,COUNT is used in trigger stage.

The :TRIGger:SEQUence:RESet command specifies the reset stage of a sequence trigger. In multi-level trigger specifications, you may find a pattern, then search for another in sequence, but reset the entire search to the beginning if another condition occurs. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example, "EDGE1,NONE,NONE").

PATTERN1 and PATTERN2 are specified with the :TRIGger:SEQUence:PATTERN command. EDGE1 and EDGE2 are specified with the :TRIGger:SEQUence:EDGE command. TIMer is specified with the :TRIGger:SEQUence:TIMer command.

Query Syntax

:TRIGger:SEQUence:RESet?

The :TRIGger:SEQUence:RESet? query returns the reset stage specification for a sequence trigger. NONE is returned for unused parameters.

Return Format

<reset_value><NL>

<reset_value> ::= (NONE,NONE,NONE | PATT1,ENT,NONE | PATT1,EXIT,NONE
    | EDGE1,NONE,NONE | PATT1,AND,EDGE1 | PATT2,ENTERed | PATT2,EXITed
    | EDGE2,ENTERed | EDGE2,EXIT,NONE | TIM,NONE,NONE

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SEQUence:PATTERN" on page 580
- ":TRIGger:SEQUence:EDGE" on page 578
- ":TRIGger:SEQUence:TIMer" on page 582
- ":TRIGger:SEQUence:FIND" on page 579
- ":TRIGger:SEQUence:TRIGger" on page 583
582 Agilent InfiniiVision 7000A Series Oscilloscopes Programmer’s Guide

5 Commands by Subsystem

:TRIGger:SEQUence:TIMer

N (see page 788)

Command Syntax

:TRIGger:SEQUence:TIMer <time_value>

<time_value> ::= time in seconds in NR1 format

The :TRIGger:SEQUence:TIMer command sets the sequencer timer resource in seconds from 10 ns to 10 s. The timer is used in the reset stage to determine how long to wait for the trigger to occur before restarting.

Query Syntax

:TRIGger:SEQUence:TIMer?

The :TRIGger:SEQUence:TIMer? query returns current sequencer timer setting.

Return Format

<time_value><NL>

<time_value> ::= time in seconds in NR1 format

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SEQUence:RESet" on page 581
:TRIGger:SEQUence:TRIGger

(see page 788)

Command Syntax

:TRIGger:SEQUence:TRIGger <value>

<value> ::= (PATTern2,ENTERed | PATTern2,EXITed | EDGE2
| PATTern2,AND,EDGE2 | EDGE2,COUNt | EDGE2,COUNt,NREFind)

The :TRIGger:SEQUence:TRIGger command specifies the trigger stage of a sequence trigger. The sequence commands set various search terms. After all of these are found in sequence, the trigger condition itself is searched for. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example, "EDGE2,NONE,NONE").

PATTern2 is specified with the :TRIGger:SEQUence:PATTern command. EDGE2 is specified with the :TRIGger:SEQUence:EDGE command. COUNt is specified with the :TRIGger:SEQUence:COUNt command.

Query Syntax

:TRIGger:SEQUence:TRIGger?

The :TRIGger:SEQUence:TRIGger? query returns the trigger stage specification for a sequence trigger. NONE is returned for unused parameters.

Return Format

<trigger_value><NL>

<trigger_value> ::= {PATT2,ENT,NONE | PATT2,EXIT,NONE
| EDGE2,NONE,NONE | PATT2,AND,EDGE2
| EDGE2,COUN,NONE | EDGE2,COUN,NREF}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SEQUence:PATTern" on page 580
- ":TRIGger:SEQUence:EDGE" on page 578
- ":TRIGger:SEQUence:COUNt" on page 577
- ":TRIGger:SEQUence:FIND" on page 579
- ":TRIGger:SEQUence:RESet" on page 581
- ":TRIGger:SEQUence:RESet" on page 581
### :TRIGger:SPI Commands

#### Table 89 :TRIGger:SPI Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:SPI:CLOCK:SL</td>
<td>OPe &lt;slope&gt; (see page 585)</td>
<td>(see page 585)</td>
</tr>
<tr>
<td>:TRIGger:SPI:CLOCK:SL</td>
<td>OPe? (see page 585)</td>
<td>&lt;slope&gt; ::= {NEGative</td>
</tr>
<tr>
<td>:TRIGger:SPI:CLOCK:TI</td>
<td>Meout &lt;time_value&gt; (see page 586)</td>
<td>Meout? (see page 586)</td>
</tr>
<tr>
<td>:TRIGger:SPI:CLOCK:TI</td>
<td>&lt;time_value&gt; ::= time in seconds in NR1 format</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:SPI:FRAMing</td>
<td>&lt;value&gt; (see page 587)</td>
<td>(see page 587)</td>
</tr>
<tr>
<td>:TRIGger:SPI:FRAMing?</td>
<td>(see page 587)</td>
<td>&lt;value&gt; ::= {CHIPselect</td>
</tr>
<tr>
<td>:TRIGger:SPI:PATTern:DATA</td>
<td>&lt;value&gt;, &lt;mask&gt; (see page 588)</td>
<td>(see page 588)</td>
</tr>
<tr>
<td>:TRIGger:SPI:PATTern:DATA?</td>
<td>(see page 588)</td>
<td>&lt;value&gt; ::= integer or &lt;string&gt;</td>
</tr>
<tr>
<td>:TRIGger:SPI:PATTern:WIDTH</td>
<td>&lt;width&gt; (see page 589)</td>
<td>(see page 589)</td>
</tr>
<tr>
<td>:TRIGger:SPI:PATTern:WIDTH?</td>
<td>(see page 589)</td>
<td>&lt;width&gt; ::= integer from 4 to 32 in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:CLK</td>
<td>&lt;source&gt; (see page 590)</td>
<td>(see page 590)</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:CLK?</td>
<td>(see page 590)</td>
<td>&lt;value&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:DATA</td>
<td>&lt;source&gt; (see page 591)</td>
<td>(see page 591)</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:DATA?</td>
<td>(see page 591)</td>
<td>&lt;value&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:FRAME</td>
<td>&lt;source&gt; (see page 592)</td>
<td>(see page 592)</td>
</tr>
<tr>
<td>:TRIGger:SPI:SOURce:FRAME?</td>
<td>(see page 592)</td>
<td>&lt;value&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
</tbody>
</table>
:TRIGger:SPI:CLOSEk:SLOPe

(see page 788)

Command Syntax :TRIGger:SPI:CLOSEk:SLOPe <slope>

<slope> ::= (NEGative | POSitive)

The :TRIGger:SPI:CLOSEk:SLOPe command specifies the rising edge (POSitive) or falling edge (NEGative) of the SPI clock source that will clock in the data.

Query Syntax :TRIGger:SPI:CLOSEk:SLOPe?

The :TRIGger:SPI:CLOSEk:SLOPe? query returns the current SPI clock source slope.

Return Format <slope><NL>

<slope> ::= (NEG | POS)

See Also • "Introduction to :TRIGger Commands" on page 468
• ":TRIGger:SPI:CLOSEk:TIMEout" on page 586
• ":TRIGger:SPI:SOURce:CLOSEk" on page 590
**:TRIGger:SPI:CLOCk:TIMeout**

(see page 788)

**Command Syntax**

```
:TRIGger:SPI:CLOCk:TIMeout <time_value>
```

*time_value* ::= time in seconds in NR1 format

The :TRIGger:SPI:CLOCk:TIMeout command sets the SPI signal clock timeout resource in seconds from 500 ns to 10 s when the :TRIGger:SPI:FRAMing command is set to TIMeout. The timer is used to frame a signal by a clock timeout.

**Query Syntax**

```
:TRIGger:SPI:CLOCk:TIMeout?
```


**Return Format**

```
<time_value><NL>
```

*time_value* ::= time in seconds in NR1 format

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SPI:CLOCk:SLOPe" on page 585
- ":TRIGger:SPI:SOURce:CLOCk" on page 590
- ":TRIGger:SPI:FRAMing" on page 587
:TRIGger:SPI:FRAMing

(see page 788)

Command Syntax

:TRIGger:SPI:FRAMing <value>

<value> ::= {CHIPselect | NOTChipselect | TIMEOUT}

The :TRIGger:SPI:FRAMing command sets the SPI trigger framing value. If TIMeout is selected, the timeout value is set by the :TRIGger:SPI:CLOCk:TIMeout command.

Query Syntax

:TRIGger:SPI:FRAMing?

The :TRIGger:SPI:FRAMing? query returns the current SPI framing value.

Return Format

<value>

<value> ::= {CHIPselect | NOTChipselect | TIMeout}

See Also

• "Introduction to :TRIGger Commands" on page 468
• ":TRIGger:MODE" on page 475
• ":TRIGger:SPI:CLOCk:TIMeout" on page 586
• ":TRIGger:SPI:SOURce:FRAME" on page 592
:TRIGger:SPI:PATTern:DATA

(see page 788)

**Command Syntax**

`:TRIGger:SPI:PATTern:DATA <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

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

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

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

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

**Query Syntax**

`:TRIGger:SPI:PATTern:DATA?`


**Return Format**

<value>, <mask><NL>

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SPI:PATTern:WIDTh" on page 589
- ":TRIGger:SPI:SOURce:DATA" on page 591
Commands by Subsystem  5

:TRIGger:SPl:PAATtern:WIDTh

(see page 788)

Command Syntax  :TRIGger:SPl:PAATtern:WIDTh <width>

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

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

Query Syntax  :TRIGger:SPl:PAATtern:WIDTh?

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

Return Format  <width><NL>

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

See Also  
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SPl:PAATtern:DATA" on page 588
- ":TRIGger:SPl:SOURce:DATA" on page 591
**:TRIGger:SPI:SOURce:CLOCk**

(see page 788)

**Command Syntax**

[:TRIGger:SPI:SOURce:CLOCk] <source>

<source> ::= \( \text{CHANnel<n>} \mid \text{EXTernal} \) for the DSO models

<source> ::= \( \text{CHANnel<n>} \mid \text{DIGital0}, \ldots, \text{DIGital15} \) for the MSO models

\(<n> ::= \{1 \mid 2 \mid 3 \mid 4\} \) for the four channel oscilloscope models

\(<n> ::= \{1 \mid 2\} \) for the two channel oscilloscope models

The [:TRIGger:SPI:SOURce:CLOCk] command sets the source for the SPI serial clock.

**Query Syntax**

[:TRIGger:SPI:SOURce:CLOCk]?

The [:TRIGger:SPI:SOURce:CLOCk]? query returns the current source for the SPI serial clock.

**Return Format**

<source><NL>

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SPI:CLOCk:SLOPe" on page 585
- ":TRIGger:SPI:CLOCk:TIMEout" on page 586
- ":TRIGger:SPI:SOURce:FRAMe" on page 592
- ":TRIGger:SPI:SOURce:DATA" on page 591
:TRIGger:SPI:SOURce:DATA

Command Syntax
:TRIGger:SPI:SOURce:DATA <source>

<source> ::= (CHANnel<n> | EXTernal) for the DSO models
<source> ::= (CHANnel<n> | DIGital0,...,DIGital15) for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:SPI:SOURce:DATA command sets the source for the SPI serial data.

Query Syntax
:TRIGger:SPI:SOURce:DATA?

The :TRIGger:SPI:SOURce:DATA? query returns the current source for the SPI serial data.

Return Format
<source><NL>

See Also
- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SPI:SOURce:CLOCKS" on page 590
- ":TRIGger:SPI:SOURce:FRAMES" on page 592
- ":TRIGger:SPI:Pattern:DATA" on page 588
- ":TRIGger:SPI:Pattern:WIDTH" on page 589
592 Agilent InfiniiVision 7000A Series Oscilloscopes Programmer’s Guide

5 Commands by Subsystem

:TRIGger:SPI:SOURce:FRAMe

N (see page 788)

Command Syntax

:TRIGger:SPI:SOURce:FRAMe <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models
<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

The :TRIGger:SPI:SOURce:FRAMe command sets the frame source when :TRIGger:SPI:FRAMing is set to CHIPselect or NOTChipselect.

Query Syntax

:TRIGger:SPI:SOURce:FRAMe? 

The :TRIGger:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:SPI:SOURce:CLOCk" on page 590
- ":TRIGger:SPI:SOURce:DATA" on page 591
- ":TRIGger:SPI:FRAMing" on page 587
### :TRIGger:TV Commands

**Table 90  :TRIGger:TV Commands Summary**

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:TV:LINE &lt;line number&gt; (see page 594)</td>
<td>:TRIGger:TV:LINE? (see page 594)</td>
<td>&lt;line number&gt; ::= integer in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:TV:MODE &lt;tv mode&gt; (see page 595)</td>
<td>:TRIGger:TV:MODE? (see page 595)</td>
<td>&lt;tv mode&gt; ::= {FIEld1</td>
</tr>
<tr>
<td>:TRIGger:TV:POLarity &lt;polarity&gt; (see page 596)</td>
<td>:TRIGger:TV:POLarity? (see page 596)</td>
<td>&lt;polarity&gt; ::= {POSitive</td>
</tr>
<tr>
<td>:TRIGger:TV:SOURce &lt;source&gt; (see page 597)</td>
<td>:TRIGger:TV:SOURce? (see page 597)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;} &lt;n&gt; ::= 1-2 or 1-4 integer in NR1 format</td>
</tr>
<tr>
<td>:TRIGger:TV:STANdard &lt;standard&gt; (see page 598)</td>
<td>:TRIGger:TV:STANdard? (see page 598)</td>
<td>&lt;standard&gt; ::= {GENeric</td>
</tr>
</tbody>
</table>
Commands by Subsystem

:TRIGger:TV:LINE

(see page 788)

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 91  TV Trigger Line Number Limits

<table>
<thead>
<tr>
<th>TV Standard</th>
<th>Mode</th>
<th>LINE</th>
<th>LField1</th>
<th>LField2</th>
<th>LAlternate</th>
<th>VERTical</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSC</td>
<td></td>
<td>1 to 263</td>
<td>1 to 262</td>
<td>1 to 262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAL</td>
<td></td>
<td>1 to 313</td>
<td>314 to 625</td>
<td>1 to 312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAL-M</td>
<td></td>
<td>1 to 263</td>
<td>264 to 525</td>
<td>1 to 262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECAM</td>
<td></td>
<td>1 to 313</td>
<td>314 to 625</td>
<td>1 to 312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GENERIC</td>
<td></td>
<td>1 to 1024</td>
<td>1 to 1024</td>
<td>1 to 1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P480L60HZ</td>
<td></td>
<td>1 to 525</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P720L60HZ</td>
<td></td>
<td>1 to 750</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1080L24HZ</td>
<td></td>
<td>1 to 1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1080L25HZ</td>
<td></td>
<td>1 to 1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1080L50HZ</td>
<td></td>
<td>1 to 1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1080L60HZ</td>
<td></td>
<td>1 to 1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1080L50HZ</td>
<td></td>
<td>1 to 1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1080L60HZ</td>
<td></td>
<td>1 to 1125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 468
- ".:TRIGger:TV:STANdard" on page 598
- ".:TRIGger:TV:MODE" on page 595
:TRIGger:TV:MODE

(see page 788)

Command Syntax

:TRIGger:TV:MODE <mode>

<mode> ::= {FIELd1 | FIELd2 | AFIelds | ALINes | LINE | VERTical |
LFIeld1 | LFIeld2 | LALTernate | LVERTical}

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERTical parameter is only available when :TRIGger:TV:STANdard is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric.

Old forms for <mode> are accepted:

<table>
<thead>
<tr>
<th>&lt;mode&gt;</th>
<th>Old Forms Accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELd1</td>
<td>F1</td>
</tr>
<tr>
<td>FIELd2</td>
<td>F2</td>
</tr>
<tr>
<td>AFIelds</td>
<td>ALLFields, ALLFLDS</td>
</tr>
<tr>
<td>ALINes</td>
<td>ALLLines</td>
</tr>
<tr>
<td>LFIeld1</td>
<td>LINEF1, LINEFIELD1</td>
</tr>
<tr>
<td>LFIeld2</td>
<td>LINEF2, LINEFIELD2</td>
</tr>
<tr>
<td>LALTernate</td>
<td>LINEAlt</td>
</tr>
<tr>
<td>LVERTical</td>
<td>LINEVert</td>
</tr>
</tbody>
</table>

Query Syntax

:TRIGger:TV:MODE?

The :TRIGger:TV:MODE? query returns the TV trigger mode.

Return Format

/value/<NL>

/value> ::= (FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2 |
| LALT | LVER}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:TV:STANdard" on page 598
- ":TRIGger:MODE" on page 475
:TRIGger:TV:POLarity

(see page 788)

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 468
- ":TRIgger:MODE" on page 475
- ":TRIgger:TV:SOURce" on page 597
**:TRIGger:TV:SOURce**

(see page 788)

**Command Syntax**

```
:TRIGger:TV:SOURce <source>
```

```
<source> ::= (CHANnel<n>)
```

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

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

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

**Query Syntax**

```
:TRIGger:TV:SOURce?
```

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

**Return Format**

```
<source><NL>
```

```
<source> ::= (CHAN<n>)
```

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:TV:POLarity" on page 596

**Example Code**

- "Example Code" on page 507
Commands by Subsystem

:TRIGger:TV:STANdard

(see page 788)

Command Syntax

:TRIGger:TV:STANdard <standard>

<standard> ::= {GENeric | NTSC | PALM | PAL | SECam
 | (P480L60HZ | P480) | (P720L60HZ | P720)
 | (P1080L24HZ | P1080) | P1080L25HZ
 | P1080L50HZ | P1080L60HZ
 | (I1080L50HZ | I1080) | I1080L60HZ}

The :TRIGger:TV:STANdard command selects the video standard. GENeric mode is non-interlaced.

Query Syntax

:TRIGger:TV:STANdard?

The :TRIGger:TV:STANdard? query returns the current TV trigger standard setting.

Return Format

<standard><NL>

<standard> ::= {GEN | NTSC | PALM | PAL | SEC | P480L60HZ | P720L60HZ
 | P1080L24HZ | P1080L25HZ | P1080L50HZ | P1080L60HZ
 | I1080L50HZ | I1080L60HZ}
### :TRIGger:UART Commands

#### Table 92 :TRIGger:UART Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:UART:BASE &lt;base&gt; (see page 601)</td>
<td>:TRIGger:UART:BASE? (see page 601)</td>
<td>&lt;base&gt; ::= {ASCii</td>
</tr>
<tr>
<td>:TRIGger:UART:BAUDrate &lt;baudrate&gt; (see page 602)</td>
<td>:TRIGger:UART:BAUDrate? (see page 602)</td>
<td>&lt;baudrate&gt; ::= integer from 1200 to 3000000 in 100 b/s increments</td>
</tr>
<tr>
<td>:TRIGger:UART:BITorde &lt;bitorder&gt; (see page 603)</td>
<td>:TRIGger:UART:BITorde? (see page 603)</td>
<td>&lt;bitorder&gt; ::= {LSBFirst</td>
</tr>
<tr>
<td>:TRIGger:UART:BURSt &lt;value&gt; (see page 604)</td>
<td>:TRIGger:UART:BURSt? (see page 604)</td>
<td>&lt;value&gt; ::= {OFF</td>
</tr>
<tr>
<td>:TRIGger:UART:DATA &lt;value&gt; (see page 605)</td>
<td>:TRIGger:UART:DATA? (see page 605)</td>
<td>&lt;value&gt; ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, &lt;hexadecimal&gt;, &lt;binary&gt;, or &lt;quoted_string&gt; format &lt;hexadecimal&gt; ::= #Hnn where n ::= {0,..,9</td>
</tr>
<tr>
<td>:TRIGger:UART:IDLE &lt;time_value&gt; (see page 606)</td>
<td>:TRIGger:UART:IDLE? (see page 606)</td>
<td>&lt;time_value&gt; ::= time from 1 us to 10 s in NR3 format</td>
</tr>
<tr>
<td>:TRIGger:UART:PARity &lt;parity&gt; (see page 607)</td>
<td>:TRIGger:UART:PARity? (see page 607)</td>
<td>&lt;parity&gt; ::= {EVEN</td>
</tr>
<tr>
<td>:TRIGger:UART:POLarity &lt;polarity&gt; (see page 608)</td>
<td>:TRIGger:UART:POLarity? (see page 608)</td>
<td>&lt;polarity&gt; ::= {HIGH</td>
</tr>
<tr>
<td>:TRIGger:UART:QUALifier &lt;value&gt; (see page 609)</td>
<td>:TRIGger:UART:QUALifier? (see page 609)</td>
<td>&lt;value&gt; ::= {EQUal</td>
</tr>
</tbody>
</table>
### Table 92: \texttt{:TRIGger:UART} Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
</table>
| \texttt{:TRIGger:UART:SOURce:RX} <source> (see page 610) | \texttt{:TRIGger:UART:SOURce:RX?} (see page 610) | <source> ::= \{CHANnel<n> | EXTernal\} for DSO models  
<source> ::= \{CHANnel<n> | DIGital0,..,DIGital15\} for MSO models  
<n> ::= 1-2 or 1-4 in NR1 format |
| \texttt{:TRIGger:UART:SOURce:TX} <source> (see page 611) | \texttt{:TRIGger:UART:SOURce:TX?} (see page 611) | <source> ::= \{CHANnel<n> | EXTernal\} for DSO models  
<source> ::= \{CHANnel<n> | DIGital0,..,DIGital15\} for MSO models  
<n> ::= 1-2 or 1-4 in NR1 format |
| \texttt{:TRIGger:UART:TYPE} <value> (see page 612) | \texttt{:TRIGger:UART:TYPE?} (see page 612) | <value> ::= \{RSTArt | RSTOp | RDAta | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATA | TD1 | TD0 | TDX\} |
| \texttt{:TRIGger:UART:WIDTH} <width> (see page 613) | \texttt{:TRIGger:UART:WIDTH?} (see page 613) | <width> ::= \{5 | 6 | 7 | 8 | 9\} |
**:TRIGger:UART:BASE**

(see page 788)

**Command Syntax**

:TRIGger:UART:BASE <base>

<base> ::= {ASCii | HEX}

The :TRIGger:UART: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 :TRIGger:UART:BASE setting does not affect the :TRIGger:UART:DATA command which can always set data values using ASCII or hexadecimal values.

**NOTE**

The :TRIGger:UART:BASE command is independent of the :SBUS:UART:BASE command which affects decode only.

**Query Syntax**

:TRIGger:UART:BASE?

The :TRIGger:UART:BASE? query returns the current UART base setting.

**Return Format**

<base><NL>

<base> ::= {ASC | HEX}

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:DATA" on page 605
Command Syntax

:TRIGger:UART:BAUDrate <baudrate>

<baudrate> ::= integer from 1200 to 3000000 in 100 b/s increments

The :TRIGger: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 1200 b/s to 3 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.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

:TRIGger:UART:BAUDrate?

The :TRIGger:UART:BAUDrate? query returns the current UART baud rate setting.

Return Format

<baudrate><NL>

<baudrate> ::= integer from 1200 to 3000000 in 100 b/s increments

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:TYPE" on page 612
**:TRIGger:UART:BITorder**

(see page 788)

**Command Syntax**

:TRIGger:UART:BITorder <bitorder>

<bitorder> ::= {LSBFirst | MSBFirst}

The :TRIGger:UART:BITorder command specifies the order of transmission used by the physical Tx and Rx input signals for the serial decoder and/or trigger when in UART mode. LSBFirst sets the least significant bit of each message "byte" as transmitted first. MSBFirst sets the most significant bit as transmitted first.

**Query Syntax**

:TRIGger:UART:BITorder?

The :TRIGger:UART:BITorder? query returns the current UART bit order setting.

**Return Format**

<bitorder><NL>

<bitorder> ::= {LSBF | MSBF}

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:TYPE" on page 612
- ":TRIGger:UART:SOURce:RX" on page 610
- ":TRIGger:UART:SOURce:TX" on page 611
Command Syntax
:TRIGGER:UART:BURSt <value>

<value> ::= (OFF | 1 to 4096 in NR1 format)

The :TRIGGER:UART:BURSt command selects the burst value (Nth frame after idle period) in the range 1 to 4096 or OFF, for the trigger when in UART mode.

Query Syntax
:TRIGGER:UART:BURSt?

The :TRIGGER:UART:BURSt? query returns the current UART trigger burst value.

Return Format
<value><NL>

<value> ::= (OFF | 1 to 4096 in NR1 format)

See Also
- "Introduction to :TRIGGER Commands" on page 468
- ":TRIGGER:MODE" on page 475
- ":TRIGGER:UART:IDLE" on page 606
- ":TRIGGER:UART:TYPE" on page 612
5

Commands by Subsystem

:TRIGger:UART:DATA
(see page 788)
Command Syntax

:TRIGger: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 :TRIGger:UART:DATA command selects the data byte value (0x00 to
0xFF) for the trigger QUALifier when in UART mode. The data value is
used when one of the RD or TD trigger types is selected.
When entering an ASCII character via the quoted string, it must be one of
the 128 valid characters (case- sensitive): "NUL", "SOH", "STX", "ETX",
"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",":", ";", "<", "=", ">", "?",
"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

:TRIGger:UART:DATA?

The :TRIGger:UART:DATA? query returns the current UART trigger data
value.
Return Format

<value><NL>
<value> ::= 8-bit integer in decimal from 0-255

See Also

• "Introduction to :TRIGger Commands" on page 468
• ":TRIGger:MODE" on page 475
• ":TRIGger:UART:BASE" on page 601
• ":TRIGger:UART:TYPE" on page 612

Agilent InfiniiVision 7000A Series Oscilloscopes Programmer's Guide

605


5 Commands by Subsystem

:TRIGger:UART:IDLE

(see page 788)

Command Syntax

:TRIGger:UART:IDLE <time_value>

<time_value> ::= time from 1 us to 10 s in NR3 format

The :TRIGger:UART:IDLE command selects the value of the idle period for burst trigger in the range from 1 us to 10 s when in UART mode.

Query Syntax

:TRIGger:UART:IDLE?

The :TRIGger:UART:IDLE? query returns the current UART trigger idle period time.

Return Format

<time_value><NL>

<time_value> ::= time from 1 us to 10 s in NR3 format

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:BURSt" on page 604
- ":TRIGger:UART:TYPE" on page 612
**:TRIGger:UART:PARity**

(see page 788)

**Command Syntax**

**:TRIGger:UART:PARity <parity>**

<parity> ::= {EVEN | ODD | NONE}

The :TRIGger:UART:PARity command selects the parity to be used with each message "byte" for the serial decoder and/or trigger when in UART mode.

**Query Syntax**

**:TRIGger:UART:PARity?**

The :TRIGger:UART:PARity? query returns the current UART parity setting.

**Return Format**

<parity><NL>

<parity> ::= {EVEN | ODD | NONE}

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:TYPE" on page 612
:TRIGger:UART:POLarity

(see page 788)

Command Syntax

:TRIGger:UART:POLarity <polarity>

<polarity> ::= {HIGH | LOW}

The :TRIGger:UART:POLarity command selects the polarity as idle low or idle high for the serial decoder and/or trigger when in UART mode.

Query Syntax

:TRIGger:UART:POLarity?

The :TRIGger:UART:POLarity? query returns the current UART polarity setting.

Return Format

<polarity><NL>

<polarity> ::= {HIGH | LOW}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:TYPE" on page 612
:TRIGGER:UART:QUALifier

Command Syntax

:TRIGGER:UART:QUALifier <value>

<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

The :TRIGGER:UART:QUALifier command selects the data qualifier when :TYPE is set to RDATa, RD1, RD0, RDX, TDATa, TD1, TD0, or TDX for the trigger when in UART mode.

Query Syntax

:TRIGGER:UART:QUALifier?

The :TRIGGER:UART:QUALifier? query returns the current UART trigger qualifier.

Return Format

<value><NL>

<value> ::= {EQU | NOT | GRE | LESS}

See Also

- "Introduction to :TRIGGER Commands" on page 468
- "TRIGGER:MODE" on page 475
- "TRIGGER:UART:TYPE" on page 612
5 Commands by Subsystem

:TRIGger:UART:SOURce:RX

(see page 788)

Command Syntax

:TRIGger:UART:SOURce:RX <source>

<source> ::= (CHANnel<n> | EXTernal) for the DSO models
<source> ::= (CHANnel<n> | DIGital0,...,DIGital15) for the MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

:TRIGger:UART:SOURce:RX?

The :TRIGger:UART:SOURce:RX? query returns the current source for the UART Rx signal.

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:TYPE" on page 612
- ":TRIGger:UART:BITorder" on page 603
 Commands by Subsystem  

### :TRIGger:UART:SOURce:TX

(see page 788)

**Command Syntax**

:TRIGger:UART:SOURce:TX <source>

*<source>* ::= (CHANnel<n> | EXTernal) for the DSO models
*<source>* ::= (CHANnel<n> | DIGital0,...,DIGital15) for the MSO models

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

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

The :TRIGger:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

**Query Syntax**

:TRIGger:UART:SOURce:TX?

The :TRIGger:UART:SOURce:TX? query returns the current source for the UART Tx signal.

**Return Format**

*<source>*

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":.TRIGger:MODE" on page 475
- ":.TRIGger:UART:TYPE" on page 612
- ":.TRIGger:UART:BITorder" on page 603
5 Commands by Subsystem

:TRIGger:UART:TYPE

(see page 788)

Command Syntax

:TRIGger:UART:TYPE <value>

/value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror
           | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}

The :TRIGger:UART:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :TRIGger:UART:DATA and
:TRIGger:UART:QUALifier commands are used to specify the data value
and comparison operator.

The RD1, RD0, RDX, TD1, TD0, and TDX types (for triggering on data and
alert bit values) are only valid when a 9-bit width has been selected.

Query Syntax

:TRIGger:UART:TYPE?

The :TRIGger:UART:TYPE? query returns the current UART trigger data
value.

Return Format

<value><NL>

/value> ::= {RSTA | RSTO | RDAT | RD1 | RD0 | RDX | PAR | TSTA |
           | TSTO | TDAT | TD1 | TD0 | TDX}

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:UART:DATA" on page 605
- ":TRIGger:UART:QUALifier" on page 609
- ":TRIGger:UART:WIDTH" on page 613
:TRIGGER:UART:WIDTH

Command Syntax

:TRIGGER:UART:WIDTH <width>

<width> ::= (5 | 6 | 7 | 8 | 9)

The :TRIGGER:UART:WIDTH command determines the number of bits (5-9) for each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

:TRIGGER:UART:WIDTH?

The :TRIGGER:UART:WIDTH? query returns the current UART width setting.

Return Format

<width><NL>

<width> ::= (5 | 6 | 7 | 8 | 9)

See Also

- "Introduction to :TRIGGER Commands" on page 468
- ":TRIGGER:MODE" on page 475
- ":TRIGGER:UART:TYPE" on page 612
### Commands by Subsystem

#### :TRIGger:USB Commands

**Table 93** :TRIGger:USB Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:USB:SOURce:D MINus &lt;source&gt; (see page 615)</td>
<td>:TRIGger:USB:SOURce:D MINus? (see page 615)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:USB:SOURce:D PLus &lt;source&gt; (see page 616)</td>
<td>:TRIGger:USB:SOURce:D PLus? (see page 616)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:TRIGger:USB:SPEed &lt;value&gt; (see page 617)</td>
<td>:TRIGger:USB:SPEed? (see page 617)</td>
<td>&lt;value&gt; ::= {LOW</td>
</tr>
<tr>
<td>:TRIGger:USB:TRIGger &lt;value&gt; (see page 618)</td>
<td>:TRIGger:USB:TRIGger? (see page 618)</td>
<td>&lt;value&gt; ::= {SOP</td>
</tr>
</tbody>
</table>
:TRIGger:USB:SOURce:DMINus

(see page 788)

Command Syntax

:TRIGger:USB:SOURce:DMINus <source>

<source> ::= (CHANnel<n> | EXTernal) for the DSO models
<source> ::= (CHANnel<n> | DIGital0,...,DIGital15) for the MSO models
<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models

The :TRIGger:USB:SOURce:DMINus command sets the source for the USB D- signal.

Query Syntax

:TRIGger:USB:SOURce:DMINus?

The :TRIGger:USB:SOURce:DMINus? query returns the current source for the USB D- signal.

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:USB:SOURce:DPLus" on page 616
- ":TRIGger:USB:TRIGger" on page 618
Commands by Subsystem

:TRIGger:USB:SOURce:DPLus

(see page 788)

Command Syntax

:TRIGger:USB:SOURce:DPLus <source>

<source> ::= {CHANnel<n> | EXternal} for the DSO models
<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

The :TRIGger:USB:SOURce:DPLus command sets the source for the USB D+ signal.

Query Syntax

:TRIGger:USB:SOURce:DPLus?

The :TRIGger:USB:SOURce:DPLus? query returns the current source for the USB D+ signal.

Return Format

<source><NL>

See Also

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:USB:SOURce:DMINus" on page 615
- ":TRIGger:USB:TRIGger" on page 618
:TRIGGER:USB:SPEED

(see page 788)

Command Syntax

:TRIGGER:USB:SPEED <value>

<value> ::= {LOW | FULL}

The :TRIGGER:USB:SPEED command sets the expected USB signal speed to be Low Speed (1.5 Mb/s) or Full Speed (12 Mb/s).

Query Syntax

:TRIGGER:USB:SPEED?

The :TRIGGER:USB:SPEED? query returns the current speed value for the USB signal.

Return Format

<value><NL>

See Also

- "Introduction to :TRIGGER Commands" on page 468
- ":TRIGGER:MODE" on page 475
- ":TRIGGER:USB:SOURce:DMINus" on page 615
- ":TRIGGER:USB:SOURce:DPLus" on page 616
- ":TRIGGER:USB:TRIGGER" on page 618
:TRIGger:USB:TRIGger

(see page 788)

Command Syntax

:TRIGger:USB:TRIGger <value>

<value> ::= {SOP | EOP | ENTERsusp | EXITsusp | RESET}

The :TRIGger:USB:TRIGger command sets where the USB trigger will occur:
- SOP — Start of packet.
- EOP — End of packet.
- ENTERsusp — Enter suspend state.
- EXITsusp — Exit suspend state.
- RESET — Reset complete.

Query Syntax

:TRIGger:USB:TRIGger?

The :TRIGger:USB:TRIGger? query returns the current USB trigger value.

Return Format

<value><NL>

<value> ::= {SOP | EOP | ENTERsusp | EXITsusp | RESET}

See Also
- "Introduction to :TRIGger Commands" on page 468
- ".TRIGger:MODE" on page 475
- ".TRIGger:USB:SPEed" on page 617
**:WAVeform Commands**

Provide access to waveform data. See "Introduction to :WAVeform Commands" on page 621.

### Table 94 :WAVeform Commands Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>:WAVeform:BYTeorder &lt;value&gt; (see page 627)</td>
<td>:WAVeform:BYTeorder? (see page 627)</td>
<td>&lt;value&gt; ::= {LSBFirst</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:COUNt? (see page 628)</td>
<td>&lt;count&gt; ::= an integer from 1 to 65536 in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:DATA? (see page 629)</td>
<td>&lt;binary block length bytes&gt;, &lt;binary data&gt; For example, to transmit 1000 bytes of data, the syntax would be: #80001000&lt;1000 bytes of data&gt;&lt;NL&gt; 8 is the number of digits that follow 0001000 is the number of bytes to be transmitted &lt;1000 bytes of data&gt; is the actual data</td>
</tr>
<tr>
<td>:WAVeform:FORMat &lt;value&gt; (see page 631)</td>
<td>:WAVeform:FORMat? (see page 631)</td>
<td>&lt;value&gt; ::= {WORD</td>
</tr>
<tr>
<td>:WAVeform:POINts &lt;# points&gt; (see page 632)</td>
<td>:WAVeform:POINts? (see page 632)</td>
<td>&lt;# points&gt; ::= {100</td>
</tr>
<tr>
<td>:WAVeform:POINts:MODE &lt;points_mode&gt; (see page 634)</td>
<td>:WAVeform:POINts:MODE? (see page 635)</td>
<td>&lt;points_mode&gt; ::= {NORMal</td>
</tr>
</tbody>
</table>
### Table 94: :WAVeform Commands Summary (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
<th>Options and Query Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>:WAVeform:PREamble? (see page 636)</td>
<td>&lt;preamble_block&gt; ::= &lt;format NR1&gt;, &lt;type NR1&gt;, &lt;points NR1&gt;, &lt;count NR1&gt;, &lt;xincrement NR3&gt;, &lt;xorigin NR3&gt;, &lt;xreference NR1&gt;, &lt;yincrement NR3&gt;, &lt;yorigin NR3&gt;, &lt;yreference NR1&gt; &lt;format&gt; ::= an integer in NR1 format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 for BYTE format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 for WORD format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 for ASCII format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;type&gt; ::= an integer in NR1 format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 for NORMAL type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 for PEAK detect type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 for AVERAGE type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 for HRESolution type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;count&gt; ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:SEGmented:COUNT? (see page 639)</td>
<td>&lt;count&gt; ::= an integer from 2 to 2000 in NR1 format (with Option SGM)</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:SEGmented:TAG? (see page 640)</td>
<td>&lt;time_tag&gt; ::= in NR3 format (with Option SGM)</td>
</tr>
<tr>
<td>:WAVeform:SOURce &lt;source&gt; (see page 641)</td>
<td>:WAVeform:SOURce? (see page 641)</td>
<td>&lt;source&gt; ::= {CHANnel&lt;n&gt;</td>
</tr>
<tr>
<td>:WAVeform:SOURce:SUBSource &lt;subsource&gt; (see page 645)</td>
<td>:WAVeform:SOURce:SUBSource? (see page 645)</td>
<td>&lt;subsource&gt; ::= {{NONE</td>
</tr>
<tr>
<td>n/a</td>
<td>:WAVeform:TYPE? (see page 646)</td>
<td>&lt;return_mode&gt; ::= {NORMAL</td>
</tr>
<tr>
<td>:WAVeform:UNSIGNED {{0</td>
<td>OFF}</td>
<td>{1</td>
</tr>
<tr>
<td>:WAVeform:VIEW &lt;view&gt; (see page 648)</td>
<td>:WAVeform:VIEW? (see page 648)</td>
<td>&lt;view&gt; ::= {MAIN}</td>
</tr>
</tbody>
</table>
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 629) and :WAVeform:PREamble (see page 636). 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 196): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always
acquired using NORMal. When the data is acquired using the :DIGitize command (see page 150) 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 629) 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 188).

**Helpful Hints:**

The number of points transferred to the computer is controlled using the :WAVEform:POINts command (see page 632). 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 632). 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 185). 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 632). 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 632). 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 196), the value returned by the :WAVEform:XINCrement query (see page 649) 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:

\[
\text{voltage} = [(\text{data value} - \text{yreference}) \times \text{yincrement}] + \text{yorigin}
\]

If the :WAVEform:FORMat data format is ASCII (see page 631), 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:

\[
\text{time} = [(\text{data point number} - \text{xreference}) \times \text{xincrement}] + \text{xorigin}
\]

This would result in the following calculation for time bucket 3:

\[
\text{time} = [(3 - 0) \times 2 \text{ ns}] + 16 \text{ ns} = 22 \text{ ns}
\]

In :ACQuire:TYPE PEAK mode (see page 196), because data is acquired in max-min pairs, modify the previous time formula to the following:

\[
\text{time}=[(\text{data pair number} - \text{xreference}) \times \text{xincrement} \times 2] + \text{xorigin}
\]

**Data Format for Transfer**

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCII (see "WAVEform:FORMat" on page 631). 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 647) 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 631) 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 627) and :WAVeform:UNSigned (see page 647) have no effect when the format is ASCII.

**Data Format for Transfer - WORD format**

WORD format (see ":WAVeform:FORMat" on page 631) 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 632). 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 627) 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 631) 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 627) 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 647) 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:

<table>
<thead>
<tr>
<th>:WAVeform:SOURce</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>POD1</td>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>POD2</td>
<td>D15</td>
<td>D14</td>
<td>D13</td>
<td>D12</td>
<td>D11</td>
<td>D10</td>
<td>D9</td>
<td>D8</td>
</tr>
</tbody>
</table>

If the :WAVeform:FORMat is WORD (see page 631) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see page 627) 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 198) 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 788)

**Command Syntax**

:WAVeform:BYTeorder <value>

<value> ::= {LSBFirst | MSBFirst}

The :WAVeform:BYTeorder command sets the output sequence of the WORD data. The parameter MSBFirst sets the most significant byte to be transmitted first. The parameter LSBFirst sets the least significant byte to be transmitted first. This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected. The default setting is LSBFirst.

**Query Syntax**

:WAVeform:BYTeorder?

The :WAVeform:BYTeorder query returns the current output sequence.

**Return Format**

<value><NL>

<value> ::= {LSBF | MSBF}

**See Also**

- "Introduction to :WAVeform Commands" on page 621
- ":WAVeform:DATA" on page 629
- ":WAVeform:FORMat" on page 631
- ":WAVeform:PREAMble" on page 636

**Example Code**

- "Example Code" on page 642
- "Example Code" on page 637
5 Commands by Subsystem

**:WAVeform:COUNt**

(see page 788)

**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 621
- ":ACQuire:COUNt" on page 185
- ":ACQuire:TYPE" on page 196
:WAVEform:DATA

(see page 788)

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:BYTorder, :WAVEform:FORMat, and :WAVEform:SOURce commands. The number of points returned is controlled by the :WAVEform:POINts command.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 — Hole. Holes are locations where data has not yet been acquired. Holes can be reasonably expected in the equivalent time acquisition mode (especially at slower horizontal sweep speeds when measuring low frequency signals).
- 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 621
- ":WAVEform:UNSigned" on page 647
- ":WAVEform:BYTorder" on page 627
- ":WAVEform:FORMat" on page 631
- ":WAVEform:POINts" on page 632
- ":WAVEform:PREamble" on page 636
- ":WAVEform:SOURce" on page 641
- ":WAVEform:TYPE" on page 646

Example Code

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString "*:WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
and the actual waveform data followed by a new line (NL) character.
The query data has the following format:

<header><waveform_data><NL>

Where:
<header> = #800001000 (This is an example header)
The "#8" may be stripped off of the header and the remaining numbers are the size, in bytes, of the waveform data block. The size can vary depending on the number of points acquired for the waveform. You can then read that number of bytes from the oscilloscope and the terminating NL character.

Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 _
        + varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) _
            * sngYIncrement + sngYOrigin) + " V," + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) _
            * sngXIncrement +dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:WAVeform:FORMat**

(see page 788)

**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.
  
  WORD formatted data transfers 16-bit data as two bytes.

When the :WAVeform:SOURce is the serial decode bus (SBUS), ASCii is the only waveform format allowed.

When the :WAVeform:SOURce is one of the digital channel buses (BUS1 or BUS2), ASCii and WORD are the only waveform formats allowed.

**Query Syntax**

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

**Return Format**

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

<value> ::= {WORD | BYTE | ASC}

**See Also**

- "Introduction to :WAVeform Commands" on page 621
- ":WAVeform:BYTeorder" on page 627
- ":WAVeform:SOURce" on page 641
- ":WAVeform:DATA" on page 629
- ":WAVeform:PREamble" on page 636

**Example Code**

- "Example Code" on page 642
Commands by Subsystem

:WAVEform:POINts

(see page 788)

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 634) for more information.

Only data visible on the display will be returned.

When the :WAVEform:SOURce is the serial decode bus (SBUS), this command is ignored, and all available serial decode bus data is returned.

Query Syntax

:WAVEform:POINts?

The :WAVEform:POINts query returns the number of waveform points to be transferred when using the :WAVEform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVEform:POINts:MODE command (see page 634) for more information).

When the :WAVEform:SOURce is the serial decode bus (SBUS), this query returns the number of messages that were decoded.

Return Format

<# points><NL>

<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>}
if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 | 5000 | 10000 | 20000
| 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
| 4000000 | 8000000 | <maximum # points>}
if waveform points mode is MAXimum or RAW
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 621
- ":ACquire:POINt" on page 188
- ":WAVEform:DATA" on page 629
- ":WAVEform:SOURce" on page 641
- ":WAVEform:VIEW" on page 648
- ":WAVEform:PREamble" on page 636
- ":WAVEform:POINts:MODE" on page 634

Example Code

```vbscript
' WAVE_POINTS - Specifies the number of points to be transferred
 ' using the ":WAVEFORM:DATA?" query.
myScope.WriteString " :WAVEFORM:POINTS 1000"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 814
:WAVeform:POINts:MODE

(see page 788)

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 three different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.

- The second is referred to as the measurement record and is a 1000-point (maximum) representation of the raw acquisition record. The measurement record can be retrieved when :SYSTem:PRECision is OFF, from any source.

- The third is referred to as the precision analysis record and is a 128K-point (maximum) representation of the raw acquisition record. The precision analysis record can be retrieved when :SYSTem:PRECision is ON, from analog sources.

If the <points_mode> is NORMal and :SYSTem:PRECision is OFF, the measurement record is retrieved.

If the <points_mode> is NORMal and :SYSTem:PRECision is ON, the precision analysis record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), or if the reconstruction filter (Sin(x)/x interpolation) is in use, the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command (see page 178) or the :DIGitize command (see page 150) in the root subsystem) in order to return more than the measurement record or precision analysis record.
• :TIMebase:MODE must be set to MAIN.
• :ACQuire:TYPE must be set to NORMal, AVERage, or HRESolution. If AVERage, :ACQuire:COUNt must be set to 1 in order to return more than the measurement record or precision analysis record.
• MAXimum or RAW will allow up to 8,000,000 points to be returned. The number of points returned will vary as the instrument's configuration is changed. Use the :WAVEform:POINts? MAXimum query to determine the maximum number of points that can be retrieved at the current settings.

Query Syntax

:WAVEform:POINts:MODE?

The :WAVEform:POINts:MODE? query returns the current points mode. Setting the points mode will affect what data is transferred. See the discussion above.

Return Format

<points_mode><NL>

(points_mode) ::= {NORMal | MAXimum | RAW)

See Also

• "Introduction to :WAVEform Commands" on page 621
• ":WAVEform:DATA" on page 629
• ":ACQuire:POINts" on page 188
• ":SYSTem:PRECision" on page 451
• ":WAVEform:VIEW" on page 648
• ":WAVEform:PREamble" on page 636
• ":WAVEform:POINts" on page 632
• ":TIMebase:MODE" on page 458
• ":ACQuire:TYPE" on page 196
• ":ACQuire:COUNt" on page 185
**:WAVeform:PREamble**

(see page 788)

**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 621
- ":ACQuire:COUNt" on page 185
- ":ACQuire:POINts" on page 188
- ":ACQuire:TYPE" on page 196
- ":DIGitize" on page 150
- ":WAVeform:COUNt" on page 628
- ":WAVeform:DATA" on page 629
- ":WAVeform:FORMat" on page 631
- ":WAVeform:POINts" on page 632
- ":WAVeform:TYPE" on page 646
- ":WAVeform:XINCrement" on page 649
- ":WAVeform:XORigin" on page 650
- ":WAVeform:XREFerence" on page 651
- ":WAVeform:YINCrement" on page 652
- ":WAVeform:YORigin" on page 653
- ":WAVeform:YREFerence" on page 654

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.
5 Commands by Subsystem

\[ TYPE \quad : \quad \text{int16} - 0 = \text{NORMAL}, \; 1 = \text{PEAK DETECT}, \; 2 = \text{AVERAGE} \]
\[ POINTS \quad : \quad \text{int32} - \text{number of data points transferred.} \]
\[ COUNT \quad : \quad \text{int32} - 1 \text{ and is always 1.} \]
\[ XINCREMENT \quad : \quad \text{float64} - \text{time difference between data points.} \]
\[ XORIGIN \quad : \quad \text{float64} - \text{always the first data point in memory.} \]
\[ XREFERENCE \quad : \quad \text{int32} - \text{specifies the data point associated with x-origin.} \]
\[ YINCREMENT \quad : \quad \text{float32} - \text{voltage diff between data points.} \]
\[ YORIGIN \quad : \quad \text{float32} - \text{value is the voltage at center screen.} \]
\[ YREFERENCE \quad : \quad \text{int32} - \text{specifies the data point where y-origin occurs.} \]

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":\WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

Example program from the start: "VISA COM Example in Visual Basic" on page 814
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 2000 in NR1 format (count set by :ACQuire:SEGMenTed:COUNt).

**See Also**

- ":ACQuire:MODE" on page 187
- ":ACQuire:SEGMenTed:COUNt" on page 191
- ":DIGitize" on page 150
- ":SINGle" on page 176
- ":RUN" on page 174
- "Introduction to :WAVeform Commands" on page 621

**Example Code**

- "Example Code" on page 192
**:WAVeform:SEGMented:TTAG**

(see page 788)

**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 192
- "Introduction to :WAVeform Commands" on page 621

**Example Code**

- "Example Code" on page 192
**:WAVeform:SOURce**

(see page 788)

**Command Syntax**

:WAVeform:SOURce <source>

```
<source> ::= {CHANnel<n> | FUNCTION | MATH | SBUS} for DSO models
<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCTION | MATH | SBUS} for MSO models
<n> ::= (1 | 2 | 3 | 4) for the four channel oscilloscope models
<n> ::= (1 | 2) for the two channel oscilloscope models
```

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply; integrate, differentiate, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS), ASCii is the only waveform format allowed.

With MSO oscilloscope models, you can choose a POD or BUS as the waveform source. There are some differences between POD and BUS when formatting and getting data from the oscilloscope:

- When POD1 or POD2 is selected as the waveform source, you can choose the BYTE, WORD, or ASCII formats (see "WAVeform:FORMAT" on page 631).
  
  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

<source><NL>
<source> ::= {CHAN<n> | FUNC | SBUS} for DSO models
<source> ::= {CHAN<n> | POD(1 | 2) | BUS(1 | 2) | FUNC | SBUS}
   for MSO models
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models

See Also

- "Introduction to :WAVEform Commands" on page 621
- ":DIGitize" on page 150
- ":WAVEform:FORMat" on page 631
- ":WAVEform:BYTeorder" on page 627
- ":WAVEform:DATA" on page 629
- ":WAVEform:PREamble" on page 636

Example Code

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' "WAVEFORM:DATA?" query. Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString "::WAVEFORM:SOURCE CHAN1"
'
' WAVE_POINTS - Specifies the number of points to be transferred
' using the "WAVEFORM:DATA?" query.
myScope.WriteString "::WAVEFORM:POINTS 1000"
'
' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output. This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString "::WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
myScope.WriteString "::WAVEFORM:FORMAT BYTE"
lngVSteps = 256
intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'  FORMAT    : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'  TYPE      : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'  POINTS    : int32 - number of data points transferred.
'  COUNT     : int32 - 1 and is always 1.
'  XINCREMENT: float64 - time difference between data points.
XORIGIN : float64 - always the first data point in memory.
XREFERENCE : int32 - specifies the data point associated with x-origin.
YINCREMENT : float32 - voltage diff between data points.
YORIGIN : float32 - value is the voltage at center screen.
YREFERENCE : int32 - specifies the data point where y-origin occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""

' strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
' strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
' strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
' strOutput = strOutput + "X increment = " + FormatNumber(dblXIncrement * 1000000) + " us" + vbCrLf
' strOutput = strOutput + "X origin = " + FormatNumber(dblXOrigin * 1000000) + " us" + vbCrLf
' strOutput = strOutput + "X reference = " + FormatNumber(dblXReference) + vbCrLf
' strOutput = strOutput + "Y increment = " + FormatNumber(sngYIncrement * 1000) + " mV" + vbCrLf
' strOutput = strOutput + "Y origin = " + FormatNumber(sngYOrigin) + " V" + vbCrLf
' strOutput = strOutput + "Y reference = " + FormatNumber(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) + " s" + vbCrLf

Commands by Subsystem

1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
FormatNumber(((lngPoints / 2 - lngXReference) * _
dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString "::WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
' <header><waveform_data><NL>
' Where:
' <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
')
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 _
        + varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + " , " + _
        FormatNumber((lngDataValue - lngYReference) _
        * sngYIncrement + sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) _
        * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

Example program from the start: "VISA COM Example in Visual Basic" on
page 814
:**WAVeform:SOURce:SUBSource**

(see page 788)

**Command Syntax**

`:WAVeform:SOURce:SUBSource <subsource>`

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

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

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

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

**Query Syntax**

`:WAVeform:SOURce:SUBSource?`

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

**Return Format**

<subsource><NL>

<subsource> ::= (NONE | TX)

**See Also**

- "Introduction to :WAVeform Commands" on page 621
- ":.WAVeform:SOURce" on page 641
:WAVeform:TYPE

(see page 788)

**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>
```

<br>

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

**NOTE**

If the :WAVeform:SOURce is POD1, POD2, or SBUS, the type is always NORM.

**See Also**

- "Introduction to :WAVeform Commands" on page 621
- ":ACQuire:TYPE" on page 196
- ":WAVeform:DATA" on page 629
- ":WAVeform:PREamble" on page 636
- ":WAVeform:SOURce" on page 641
**:WAVEform:UNSIGNED**

(see page 788)

**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 621
- ":WAVEform:SOURce" on page 641
5  Commands by Subsystem

:WAVeform:VIEW

(see page 788)

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 621
  • ":WAVeform:POINts" on page 632
**:WAVEform:XINCrement**

(see page 788)

**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 621
- ":WAVEform:PREamble" on page 636

**Example Code**

- "Example Code" on page 637
**:WAVeform:XORigin**

(see page 788)

**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 621
- ":WAVeform:PREamble" on page 636
- ":WAVeform:XREFerence" on page 651

**Example Code**

- "Example Code" on page 637
:WAVEform:XREference

(see page 788)

**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 621
- ":WAVEform:PREamble" on page 636
- ":WAVEform:XRigin" on page 650

**Example Code**

- "Example Code" on page 637
:WAVEform:YINCrement

(see page 788)

**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 621
- "WAVEform:PREamble" on page 636

**Example Code**

- "Example Code" on page 637
**:WAVeform:YORigin**

(see page 788)

**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 621
- ":WAVeform:PREamble" on page 636
- ":WAVeform:YREFerence" on page 654

**Example Code**

- "Example Code" on page 637
**:WAVEform:YREFerence**

(see page 788)

**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 621
- ":WAVEform:PREamble" on page 636
- ":WAVEform:YORigin" on page 653

**Example Code**

- "Example Code" on page 637
6
Commands A-Z

A 655
B 657
C 657
D 660
E 662
F 662
G 664
H 664
I 665
L 666
M 667
N 670
O 671
P 671
Q 673
R 673
S 675
T 679
U 685
V 686
W 686
X 687
Y 688

A
• AALias, ":ACQuire:AALias" on page 183
• ACKNowledge, ":TRIGger:CAN:ACKNowledge" on page 740
• ":ACQuire:AALias" on page 183
• ":ACQuire:COMComplete" on page 184
• ":ACQuire:COUNt" on page 185
• ":ACQuire:DAALias" on page 186
- ":ACQuire:MODE" on page 187
- ":ACQuire:POINts" on page 188
- ":ACQuire:RSIGnal" on page 189
- ":ACQuire:SEGMeented:ANALyze" on page 190
- ":ACQuire:SEGMeented:COUNt" on page 191
- ":ACQuire:SEGMeented:INDex" on page 192
- ":ACQuire:SRATe" on page 195
- ":ACQuire:TYPE" on page 196
- ":ACTivity" on page 142
- ADDRess, ":TRIGger:IIC:PATTern:ADDResS" on page 548
- ":AER (Arm Event Register)" on page 143
- ALIGnment, ":TRIGger:I2S:ALIGnment" on page 531

AMASk Commands:
  - ":MTESt:AMASk:CREate" on page 364
  - ":MTESt:AMASk:{SAVE | STORe}" on page 729
  - ":MTESt:AMASk:SOURce" on page 365
  - ":MTESt:AMASk:UNITs" on page 366
  - ":MTESt:AMASk:XDELta" on page 367
  - ":MTESt:AMASk:YDELta" on page 368
- ANALyze, ":ACQuire:SEGMeented:ANALyze" on page 190
- APRinter, ":HARDcopy:APRinter" on page 284

AREA Commands:
  - ":HARDcopy:AREA" on page 283
  - ":SAVE:IMAGe:AREA" on page 408
  - ASIZe, ":SBUS:IIC:ASIZe" on page 434
  - AUDio, ":TRIGger:I2S:AUDio" on page 532
  - ":AUToscale" on page 144
    - ":AUToscale:AMODE" on page 146
    - ":AUToscale:CHANnels" on page 147

AUTosetup Commands:
  - ":TRIGger:FLEXray:AUTosetup" on page 509
  - ":TRIGger:M1553:AUTosetup" on page 570

AVERage Commands:
  - ":MTESt:AVERage" on page 730
  - ":MTESt:AVERage:COUNt" on page 731
B

- BASE Commands:
  - ".SBUS:M1553:BASE" on page 436
  - ".SBUS:UART:BASE" on page 440
  - ".TRIGGER:UART:BASE" on page 601

- BAUDrate Commands:
  - ".TRIGGER:CAN:SIGNal:BAUDrate" on page 487
  - ".TRIGGER:FLEXray:BAUDrate" on page 510
  - ".TRIGGER:LIN:SIGNal:BAUDrate" on page 564
  - ".TRIGGER:UART:BAUDrate" on page 602

- BIND, ".MTEST:SCALE:BIND" on page 386

- BIT<m>, ".BUS<n>:BIT<m>" on page 200

- BITorder Commands:
  - ".SBUS:SPI:BITorder" on page 438
  - ".TRIGGER:UART:BITorder" on page 603

- BITS, ".BUS<n>:BITS" on page 201

- ".BLANK" on page 148

- BURSt, ".TRIGGER:UART:BURSt" on page 604
  - ".BUS<n>:BIT<m>" on page 200
  - ".BUS<n>:BITS" on page 201
  - ".BUS<n>:CLEar" on page 203
  - ".BUS<n>:DISPLAY" on page 204
  - ".BUS<n>:LABEL" on page 205
  - ".BUS<n>:MASK" on page 206

- BWLimit Commands:
  - ".CHANnel<n>:BWLimit" on page 220
  - ".EXTERNAL:BWLimit" on page 256

- BYTEorder, ".WAVEform:BYTEorder" on page 627

C

- ".CALibrate:DATE" on page 209
  - ".CALibrate:LABEL" on page 210
  - ".CALibrate:OUTPUT" on page 211
  - ".CALibrate:START" on page 212
  - ".CALibrate:STATUS" on page 213
  - ".CALibrate:SWITCH" on page 214
  - ".CALibrate:TEMPerature" on page 215
  - ".CALibrate:TIME" on page 216
CAN Commands:
- ":SBUS:CAN:COUNt:ERRor" on page 423
- ":SBUS:CAN:COUNt:OVERload" on page 424
- ":SBUS:CAN:COUNt:RESet" on page 425
- ":SBUS:CAN:COUNt:TOTal" on page 426
- ":SBUS:CAN:COUNt:UTILization" on page 427
- ":TRIGger:CAN Commands" on page 480

CCBASE, ":TRIGger:FLEXray:FRAME:CCBase" on page 514
CCRepetition, ":TRIGger:FLEXray:FRAME:CCRepetition" on page 515
":CDISplay" on page 149
CENTer, ":FUNCtion:CENTer" on page 267
CHANnel, ":TRIGger:FLEXray:CHANnel" on page 511
":CHANnel:ACTivity" on page 695
":CHANnel:LABel" on page 696
":CHANnel:THReshold" on page 697
":CHANnel2:SKEW" on page 698
":CHANnel<n>:BWLimit" on page 220
":CHANnel<n>:COUpling" on page 221
":CHANnel<n>:DISPlay" on page 222
":CHANnel<n>:IMPedance" on page 223
":CHANnel<n>:INPut" on page 699
":CHANnel<n>:INVert" on page 224
":CHANnel<n>:LABel" on page 225
":CHANnel<n>:OFFSet" on page 226
":CHANnel<n>:PMODE" on page 700
":CHANnel<n>:PROBe" on page 227
":CHANnel<n>:PROBe:HEAD[:TYPE]" on page 228
":CHANnel<n>:PROBe:ID" on page 229
":CHANnel<n>:PROBe:SKEW" on page 230
":CHANnel<n>:PROBe:STYPe" on page 231
":CHANnel<n>:PROTection" on page 232
":CHANnel<n>:RANGE" on page 233
":CHANnel<n>:SCALe" on page 234
":CHANnel<n>:UNITs" on page 235
":CHANnel<n>:VERNier" on page 236
• CLEar Commands:
  - ":BUS<n>:CLEar" on page 203
  - ":DISPlay:CLEar" on page 246
  - ":MEASure:CLEar" on page 314

• CLOCk Commands:
  - ":TRIGger:IIC[:SOURce]:CLOCk" on page 551
  - ":TRIGger:I2S:CLOCk:SLOPe" on page 533
  - ":TRIGger:I2S:SOURce:CLOCk" on page 540
  - ":TRIGger:SPI:CLOCk:SLOPe" on page 585
  - ":TRIGger:SPI:CLOCk:TImeout" on page 586
  - ":TRIGger:SPI:SOURce:CLOCk" on page 590

• "*CLS (Clear Status)" on page 117

• COMComplete, ":ACQuire:COMComplete" on page 184

• CONDition, ":HWERegister:CONDition (Hardware Event Condition Register)" on page 154

• CONNection, ":DISPlay:CONNection" on page 701

• COUNt Commands:
  - ":ACQuire:COUNt" on page 185
  - ":ACQuire:SEGmented:COUNt" on page 191
  - ":MTESt:AVERage:COUNt" on page 731
  - ":MTESt:COUNt:FWAVeforms" on page 369
  - ":MTESt:COUNt:RESet" on page 370
  - ":MTESt:COUNt:TIme" on page 371
  - ":MTESt:COUNt:WAVeforms" on page 372
  - ":SBUS:CAN:COUNt:ERRor" on page 423
  - ":SBUS:CAN:COUNt:OVERload" on page 424
  - ":SBUS:CAN:COUNt:RESet" on page 425
  - ":SBUS:CAN:COUNt:TOTal" on page 426
  - ":SBUS:CAN:COUNt:UTILization" on page 427
  - ":SBUS:FLEXray:COUNt:NULL" on page 429
  - ":SBUS:FLEXray:COUNt:RESet" on page 430
  - ":SBUS:FLEXray:COUNt:SYNC" on page 431
  - ":SBUS:FLEXray:COUNt:TOTal" on page 432
  - ":SBUS:UART:COUNt:ERRor" on page 441
  - ":SBUS:UART:COUNt:RESet" on page 442
 Commands A-Z

- ".SBUS:UART:COUNt:RXFRames" on page 443
- ".SBUS:UART:COUNt:TXFRames" on page 444
- ".TRIGger:EBURst:COUNt" on page 499
- ".TRIGger:SEQuence:COUNt" on page 577
- ".WAVeform:COUNt" on page 628
- ".WAVeform:SEGMeneted:COUNt" on page 639
- COUNter, ":MEASure:COUNt" on page 315

COUPling Commands:
- ":CHANnel<n>:COUPling" on page 221
- ".TRIGger[:EDGE]:COUPling" on page 503
- CREate, ":MTESt:AMASk:CREate" on page 364

D
- DAALias, ":ACQuire:DAALias" on page 186

DATA Commands:
- ":DISPlay:DATA" on page 247
- ":LISTer:DATA" on page 293
- ":MTESt:DATA" on page 373
- ":TRIGger:CAN:PATTern:DATA" on page 482
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 483
- ":TRIGger:I2S:PATTern:DATA" on page 534
- ":TRIGger:I2S:SOURce:DATA" on page 541
- ":TRIGger:IIC:PATTern:DATA" on page 549
- ":TRIGger:IIC[:SOURce]:DATA2" on page 550
- ":TRIGger:IIC[:SOURce]:DATA" on page 552
- ":TRIGger:LIN:PATTern:DATA" on page 559
- ":TRIGger:LIN:PATTern:DATA:LENGth" on page 561
- ":TRIGger:M1553:PATTern:DATA" on page 571
- ":TRIGger:SPI:PATTern:DATA" on page 588
- ":TRIGger:SPI:SOURce:DATA" on page 591
- ":TRIGger:UART:DATA" on page 605
- ":WAVeform:DATA" on page 629

DATE Commands:
- ":CALibrate:DATE" on page 209
- ":SYSTem:DATE" on page 447

DEFine, ":MEASure:DEFine" on page 316

DEFinition Commands:
• ":TRIGger:CAN:SIGNal:DEFinition" on page 488
• ":TRIGger:LIN:SIGNal:DEFinition" on page 741
• DELay Commands:
  • ":MEASure:DELay" on page 319
  • ":TIMEbase:DELay" on page 739
• DELete, ":MTESt:DELeTe" on page 374
• DESTination, ":HARDcopy:DESTination" on page 708
• DEVice, ":HARDcopy:DEVice" on page 709
• ":DIGital<n>:DISPlay" on page 239
• ":DIGital<n>:LABel" on page 240
• ":DIGital<n>:POSition" on page 241
• ":DIGital<n>:SIZE" on page 242
• ":DIGital<n>:THReshold" on page 243
• ":DIGitize" on page 150
• DISPlay Commands:
  • ":BUS<n>:DISPlay" on page 204
  • ":CHANnel<n>:DISPlay" on page 222
  • ":DIGital<n>:DISPlay" on page 239
  • ":FUNCtion:DISPlay" on page 268
  • ":LISTer:DISPlay" on page 294
  • ":POD<n>:DISPlay" on page 394
  • ":SBUS:DISPlay" on page 428
• ":DISPlay:CLEar" on page 246
• ":DISPlay:CONNect" on page 701
• ":DISPlay:DATA" on page 247
• ":DISPlay:LABel" on page 249
• ":DISPlay:LABList" on page 250
• ":DISPlay:ORDer" on page 702
• ":DISPlay:PERSistence" on page 251
• ":DISPlay:SOURce" on page 252
• ":DISPlay:VECTors" on page 253
• DMINus, ":TRIGger:USB:SOURce:DMINus" on page 615
• DPLus, ":TRIGger:USB:SOURce:DPLus" on page 616
• DSP, ":SYSTem:DSPl" on page 448
• DURation, ":TRIGger:DURation Commands" on page 492
• DUTYcycle, "MEASure:DUTYcycle" on page 321

• EBURst, "TRIGger:EBURst Commands" on page 498

• EDGE Commands:
  • "TRIGger[:EDGE] Commands" on page 502
  • "TRIGger:SEQUence:EDGE" on page 578

• ENABle"MTESt:ENABLE" on page 375

• "ERASe" on page 703

• ERRor Commands:
  • "SBUS:CAN:COUNT:ERRor" on page 423
  • "SBUS:UART:COUNT:ERRor" on page 441
  • "SYSTem:ERRor" on page 449
  • "TRIGger:FLEXray:ERRor:TYPE" on page 512

• "ESE (Standard Event Status Enable)" on page 118

• "ESR (Standard Event Status Register)" on page 120

• EVENt Commands:
  • "HWERegister[:EVENt] (Hardware Event Event Register)" on page 156
  • "MTERegister[:EVENt] (Mask Test Event Event Register)" on page 161
  • "TRIGger:FLEXray:EVENt:TYPE" on page 513

• "EXternal:BWLimit" on page 256

• "EXternal:IMPedance" on page 257

• "EXternal:INPut" on page 704

• "EXternal:PMODE" on page 705

• "EXternal:PROBe" on page 258

• "EXternal:PROBe:ID" on page 259

• "EXternal:PROBe:STYPE" on page 260

• "EXternal:PROTection" on page 261

• "EXternal:RANGe" on page 262

• "EXternal:UNITs" on page 263

• FACTion Commands:
  • "MTESt:RMODe:FACTion:MEASure" on page 379
  • "MTESt:RMODe:FACTion:PRINt" on page 380
  • "MTESt:RMODe:FACTion:SAVE" on page 381
  • "MTESt:RMODe:FACTion:STOP" on page 382

• FACTors Commands:
• ":HARDcopy:FACTors" on page 285
• ":SAVE:IMAGe:FACTors" on page 409
• FALLtime, ":MEASure:FALLtime" on page 322
• FFEed, ":HARDcopy:FFEed" on page 286
• FILEname Commands:
  • ":HARDcopy:FILEneme" on page 710
  • ":RECall:FILEneme" on page 399
  • ":SAVE:FILEneme" on page 406
• FIND, ":TRIGger:SEQUence:FIND" on page 579
• FLEXray Commands:
  • ":SBUS:FLEXray:COUNt:NULl" on page 429
  • ":SBUS:FLEXray:COUNt:RESet" on page 430
  • ":SBUS:FLEXray:COUNt:SYNC" on page 431
  • ":SBUS:FLEXray:COUNt:TOTal" on page 432
  • ":TRIGger:FLEXray:AUTosetup" on page 509
  • ":TRIGger:FLEXray:BAUDrate" on page 510
  • ":TRIGger:FLEXray:CHANnel" on page 511
  • ":TRIGger:FLEXray:ERRor:TYPE" on page 512
  • ":TRIGger:FLEXray:EVENt:TYPE" on page 513
  • ":TRIGger:FLEXray:FRAMe:CCBase" on page 514
  • ":TRIGger:FLEXray:FRAMe:CCRepetition" on page 515
  • ":TRIGger:FLEXray:FRAMe:ID" on page 516
  • ":TRIGger:FLEXray:FRAMe:TYPE" on page 517
  • ":TRIGger:FLEXray:SOURce" on page 518
  • ":TRIGger:FLEXray:TRIGger" on page 519
• FORMat Commands:
  • ":HARDcopy:FORMat" on page 711
  • ":SAVE:IMAGe:FORMat" on page 410
  • ":SAVE:WAVEform:FORMat" on page 418
  • ":TRIGger:I2S:PATTern:FORMat" on page 536
  • ":TRIGger:LIN:PATTern:FORMat" on page 562
  • ":WAVEform:FORMat" on page 631
• FRAMe Commands:
  • ":TRIGger:FLEXray:FRAMe:CCBase" on page 514
  • ":TRIGger:FLEXray:FRAMe:CCRepetition" on page 515
FRAMing Commands:
- ".SBUS:UART:FRAMing" on page 445
- ".TRIGGER:SPI:FRAMing" on page 587

FREQuency, ":MEASure:FREQuency" on page 323

".FUNCTION:CENTer" on page 267
".FUNCTION:DISPLAY" on page 268
".FUNCTION:GOFT:OPERation" on page 269
".FUNCTION:GOFT:SOURce1" on page 270
".FUNCTION:GOFT:SOURce2" on page 271
".FUNCTION:OFFSET" on page 272
".FUNCTION:OPERation" on page 273
".FUNCTION:RANGE" on page 274
".FUNCTION:REFERENCE" on page 275
".FUNCTION:SCALE" on page 276
".FUNCTION:SOURce" on page 706
".FUNCTION:SOURce1" on page 277
".FUNCTION:SOURce2" on page 278
".FUNCTION:SPAN" on page 279
".FUNCTION:VIEW" on page 707
".FUNCTION:WINDOW" on page 280

FWAVEforms, ":MTEST:COUNt:FWAVEforms" on page 369

GLITCh (Pulse Width), ":TRIGGER:GLITCh Commands" on page 520

GOFT Commands:
- ".FUNCTION:GOFT:OPERation" on page 269
- ".FUNCTION:GOFT:SOURce1" on page 270
- ".FUNCTION:GOFT:SOURce2" on page 271

GRAYscale, ":HARDcopy:GRAYscale" on page 712

GREaterthan Commands:
- ".TRIGGER:DURATION:GREaterthan" on page 493
- ".TRIGGER:GLITCh:GREaterthan" on page 522

HARDcopy:AREA" on page 283
• ":HARDcopy:APRinter" on page 284
• ":HARDcopy:DESTination" on page 708
• ":HARDcopy:DEVice" on page 709
• ":HARDcopy:FACTors" on page 285
• ":HARDcopy:FEed" on page 286
• ":HARDcopy:FILename" on page 710
• ":HARDcopy:FORMat" on page 711
• ":HARDcopy:GRAYscale" on page 712
• ":HARDcopy:IGColors" on page 713
• ":HARDcopy:INKSaver" on page 287
• ":HARDcopy:LAYout" on page 288
• ":HARDcopy:PALette" on page 289
• ":HARDcopy:PDRiver" on page 714
• ":HARDcopy:PRINter:LIST" on page 290
• ":HARDcopy:STARt" on page 291
• HEAD, ":CHANnel<n>:PROBe:HEAD[:TYPE]" on page 228
• HFReject, ":TRIGger:HFReject" on page 472
• HOLDoff, ":TRIGger:HOLDoff" on page 473
• ":HWEnable (Hardware Event Enable Register)" on page 152
• ":HWERegister:CONDition (Hardware Event Condition Register)" on page 154
• ":HWERegister[:EVENt] (Hardware Event Event Register)" on page 156

I

• ID Commands:
  • ":TRIGger:CAN:PATTern:ID" on page 484
  • ":TRIGger:CAN:PATTern:ID:MODE" on page 485
  • ":TRIGger:FLEXray:FRAMe:ID" on page 516
• IDLE Commands:
  • ":TRIGger:EBURst:IDLE" on page 500
  • ":TRIGger:UART:IDLE" on page 606
• ":IDN (Identification Number)" on page 122
• I2S Commands:
  • ":SBUS:I2S:BASE" on page 433
  • ":TRIGger:I2S Commands" on page 529
• IIC Commands:
  • ":SBUS:IIC:ASIZe" on page 434
• ":TRIGger:IIC Commands" on page 547

• IGColors Commands:
  • ":HARDcopy:IGColors" on page 713
  • ":SAVE:IMAGE:INKSaver" on page 411

• IMAGE Commands:
  • ":RECall:IMAGE[:STArt]" on page 400
  • ":SAVE:IMAGE:AREA" on page 408
  • ":SAVE:IMAGE:FACTors" on page 409
  • ":SAVE:IMAGE:FORMat" on page 410
  • ":SAVE:IMAGE:INKSaver" on page 411
  • ":SAVE:IMAGE:PALETTE" on page 412
  • ":SAVE:IMAGE[:STArt]" on page 407

• IMPedance Commands:
  • ":CHANnel<n>:IMPedance" on page 223
  • ":EXTERN:IMPedance" on page 257

• INCRement, ":MEASure:STATistics:INCRement" on page 340

• INDEX, ":ACQuire:SEGmented:INDEX" on page 192

• INKSaver, ":HARDcopy:INKSaver" on page 287

• INVert, ":CHANnel<n>:INVert" on page 224

• LABel Commands:
  • ":BUS<n>:LABel" on page 205
  • ":CALibrate:LABel" on page 210
  • ":CHANnel:LABel" on page 696
  • ":CHANnel<n>:LABel" on page 225
  • ":DIGital<n>:LABel" on page 240
  • ":DISPlay:LABel" on page 249

• LABList, ":DISPlay:LABList" on page 250

• LAYOUT, ":HARDcopy:LAYOUT" on page 288

• LENGth Commands:
  • ":SAVE:WAVEform:LENGth" on page 419
  • ":TRIGger:CAN:PARTtern:DATA:LENGth" on page 483
  • ":TRIGger:LIN:PARTtern:DATA:LENGth" on page 561

• LESSthan Commands:
  • ":TRIGger:DURATION:LESSthan" on page 494
  • ":TRIGger:GLITch:LESSthan" on page 523
• LEVel Commands:
  • ":TRIGger[:EDGE]:LEVel" on page 504
  • ":TRIGger:GLITch:LEVel" on page 524
• LFIFty, ":TRIGger:LFIFty" on page 474
• LIN Commands:
  • ":SBUS:LIN:PARity" on page 435
  • ":TRIGger:LIN Commands" on page 556
• LINE, ":TRIGger:TV:LINE" on page 594
• LIST, ":HARDcopy:PRINter:LIST" on page 290
• LISTER Commands:
  • ":LISTER:DATA" on page 293
  • ":LISTER:DISPLAY" on page 294
  • ":SAVE:LISTER[:START]" on page 413
• LOAD, ":MTEST:LOAD" on page 732
• LOCK Commands:
  • ":MTEST:LOCK" on page 376
  • ":SYSTem:LOCK" on page 450
  • ":SYSTem:PROTection:LOCK" on page 452
• LOWer Commands:
  • ":MEASURE:LOWer" on page 715
  • ":TRIGger:M1553:SOURce:LOWer" on page 573
• "*LRN (Learn Device Setup)" on page 123

M
• M1553 Commands:
  • ":SBUS:M1553:BASE" on page 436
  • ":TRIGger:M1553 Commands" on page 569
  • ":MARKer:MODE" on page 297
  • ":MARKer:X1Position" on page 298
  • ":MARKer:X1Y1source" on page 299
  • ":MARKer:X2Position" on page 300
  • ":MARKer:X2Y2source" on page 301
  • ":MARKer:XDELta" on page 302
  • ":MARKer:Y1Position" on page 303
  • ":MARKer:Y2Position" on page 304
  • ":MARKer:YDELta" on page 305
• MASK Commands:
- ":BUS<n>:MASK" on page 206
- ":RECall:MASK[:START]" on page 401
- ":SAVE:MASK[:START]" on page 414
- ":MEASure:CLEAR" on page 314
- ":MEASure:COUNTer" on page 315
- ":MEASure:DEFine" on page 316
- ":MEASure:DELAY" on page 319
- ":MEASure:DUTYcycle" on page 321
- ":MEASure:FA LLtime" on page 322
- ":MEASure:FR EQuency" on page 323
- ":MEASure:LO Wer" on page 715
- ":MEASure:NWIDth" on page 324
- ":MEASure:OVERshoot" on page 325
- ":MEASure:PERiod" on page 327
- ":MEASure:PHASe" on page 328
- ":MEASure:PREShoot" on page 329
- ":MEASure:PWIDth" on page 330
- ":MEASure:RESULTS" on page 331
- ":MEASure:RISetime" on page 334
- ":MEASure:SCRatch" on page 716
- ":MEASure:SDEViation" on page 335
- ":MEASure:SHOW" on page 336
- ":MEASure:SOURce" on page 337
- ":MEASure:STATistics" on page 339
- ":MEASure:STATistics:INCRement" on page 340
- ":MEASure:STATistics:RESET" on page 341
- ":MEASure:TDELta" on page 717
- ":MEASure:TEDGE" on page 342
- ":MEASure:THResholds" on page 718
- ":MEASure:TMAX" on page 719
- ":MEASure:TMIN" on page 720
- ":MEASure:TSTArt" on page 721
- ":MEASure:TSTOP" on page 722
- ":MEASure:TVALue" on page 344
- ":MEASure:TVOLT" on page 723
• ":MEASure:UPPer" on page 725
• ":MEASure:VAMPlitude" on page 346
• ":MEASure:VAVerage" on page 347
• ":MEASure:VBASe" on page 348
• ":MEASure:VDELta" on page 726
• ":MEASure:VMAX" on page 349
• ":MEASure:VMIN" on page 350
• ":MEASure:VPP" on page 351
• ":MEASure:VRATio" on page 352
• ":MEASure:VRMS" on page 353
• ":MEASure:VSTArt" on page 727
• ":MEASure:VSTOp" on page 728
• ":MEASure:VTIMe" on page 354
• ":MEASure:VTOP" on page 355
• ":MEASure:WINDow" on page 356
• ":MEASure:XMAX" on page 357
• ":MEASure:XMIN" on page 358

MEASure, ":MTESt:RMODE:FACTion:MEASure" on page 379

• ":MERGe" on page 158

• MODE Commands:
  • ":ACQuire:MODE" on page 187
  • ":MARKer:MODE" on page 297
  • ":SBUS:MODE" on page 437
  • ":TIMebase:MODE" on page 458
  • ":TRIGger:CAN:PA TTern:ID:MODE" on page 485
  • ":TRIGger:MODE" on page 475
  • ":TRIGger:TV:MODE" on page 595
  • ":WA Veform:POINts:MODE" on page 634
  • ":MTEenable (Mask Test Event Enable Register)" on page 159
  • ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 161
  • ":MTESt:AMASk:CREate" on page 364
  • ":MTESt:AMASk:{SAVE | STORe}" on page 729
  • ":MTESt:AMASk:SOURce" on page 365
  • ":MTESt:AMASk:UNITs" on page 366
  • ":MTESt:AMASk:XDELta" on page 367
- ":MTEST:AMASk:YDELta" on page 368
- ":MTEST:AVERage" on page 730
- ":MTEST:AVERage:COUNt" on page 731
- ":MTEST:COUNt:FWAveforms" on page 369
- ":MTEST:COUNt:RESet" on page 370
- ":MTEST:COUNt:TIME" on page 371
- ":MTEST:COUNt:WAveforms" on page 372
- ":MTEST:DATA" on page 373
- ":MTEST:DELete" on page 374
- ":MTEST:ENABle" on page 375
- ":MTEST:LOAD" on page 732
- ":MTEST:LOCK" on page 376
- ":MTEST:OUTPut" on page 377
- ":MTEST:RMODe" on page 378
- ":MTEST:RMODe:FACTion:MEASure" on page 379
- ":MTEST:RMODe:FACTion:PRINT" on page 380
- ":MTEST:RMODe:FACTion:SAVE" on page 381
- ":MTEST:RMODe:FACTion:STOP" on page 382
- ":MTEST:RMODe:SIGMa" on page 383
- ":MTEST:RMODe:TIME" on page 384
- ":MTEST:RMODe:WAveforms" on page 385
- ":MTEST:RUMode" on page 733
- ":MTEST:RUMode:SOFailure" on page 734
- ":MTEST:SCALe:BIND" on page 386
- ":MTEST:SCALe:X1" on page 387
- ":MTEST:SCALe:XDELta" on page 388
- ":MTEST:SCALe:Y1" on page 389
- ":MTEST:SCALe:Y2" on page 390
- ":MTEST:SOURce" on page 391
- ":MTEST:{STARt | STOP}" on page 735
- ":MTEST:TITLe" on page 392
- ":MTEST:TRIGger:SOURce" on page 736
- NREJect, ":TRIGger:NREJect" on page 476
- NULL, ":SBUS:FLEXray:COUNt:NULl" on page 429
- NWIDth, ":MEASure:NWIDth" on page 324
OFFSet Commands:
- "::CHANnel<n>::OFFSet" on page 226
- "::FUNCtion::OFFSet" on page 272
- "::OPC (Operation Complete)" on page 124
- "::OPEE (Operation Status Enable Register)" on page 163

OPERation Commands:
- "::FUNCtion::GOFT::OPERation" on page 269
- "::FUNCtion::OPERation" on page 273
- "::OPERegister::CONDition (Operation Status Condition Register)" on page 165
- "::OPERegister::<EVENt> (Operation Status Event Register)" on page 167
- "::OPT (Option Identification)" on page 125

ORDER, "::DISPlay::ORDER" on page 702

OUTPut Commands:
- "::CALibrate::OUTPut" on page 211
- "::MTESt::OUTPut" on page 377
- OVERload, "::SBUS:CAN:COUNt::OVERload" on page 424
- OVERshoot, "::MEASure::OVERshoot" on page 325
- "::OVLenable (Overload Event Enable Register)" on page 169
- "::OVLRegister (Overload Event Register)" on page 171

PALette Commands:
- "::HARDcopy::PALette" on page 289
- "::SAVE::IMAGe::PALette" on page 412

PARity Commands:
- "::SBUS:LIN:PARity" on page 435
- "::TRIGger:UART:PARity" on page 607

PATTern Commands:
- "::TRIGger:CAN:PATTern:DATA" on page 482
- "::TRIGger:CAN:PATTern:DATA::LENGTH" on page 483
- "::TRIGger:CAN:PATTern::ID" on page 484
- "::TRIGger:CAN:PATTern::ID::MODE" on page 485
- "::TRIGger:DURation::PATTern" on page 495
- "::TRIGger:I2S:PATTern::DATA" on page 534
- "::TRIGger:I2S:PATTern::FORMat" on page 536
- "::TRIGger:IIC:PATTern::ADDResS" on page 548
• ":TRIGger:IIC:PATTern:DATA" on page 549
• ":TRIGger:IIC:PATTern:DATA2" on page 550
• ":TRIGger:LIN:PATTern:DATA" on page 559
• ":TRIGger:LIN:PATTern:DATA:LENGth" on page 561
• ":TRIGger:LIN:PATTern:FORMAT" on page 562
• ":TRIGger:M1553:PATTern:DATA" on page 571
• ":TRIGger:PATTern" on page 477
• ":TRIGger:SEQUence:PATTern" on page 580
• ":TRIGger:SPI:PATTern:DATA" on page 588
• ":TRIGger:SPI:PATTern:WIDTH" on page 589
• PDRiver, ":HARDcopy:PDRiver" on page 714
• PERiod, ":MEASure:PERiod" on page 327
• PERSistence, ":DISPlay:PERSistence" on page 251
• PHASe, ":MEASure:PHASe" on page 328
• PMODe, ":CHANnel<n>:PMODe" on page 700
• ":POD<n>:DISPlay" on page 394
• ":POD<n>:SIZE" on page 395
• ":POD<n>:THReshold" on page 396
• POINts Commands:
  • ":ACQuire:POINts" on page 188
  • ":WAVeform:POINts" on page 632
  • ":WAVeform:POINts:MODE" on page 634
• POLarity Commands:
  • ":TRIGger:GLITch:POLarity" on page 525
  • ":TRIGger:TV:POLarity" on page 596
  • ":TRIGger:UART:POLarity" on page 608
• POSition Commands:
  • ":DIGital<n>:POSition" on page 241
  • ":TIMebase:POSition" on page 459
  • ":TIMebase:WINDOW:POSition" on page 465
• PREamble, ":WAVeform:PREamble" on page 636
• PRECision, ":SYSTem:PRECision" on page 451
• PREShoot, ":MEASure:PREShoot" on page 329
• PRINt, ":MTESt:RMODe:FACTion:PRINt" on page 380
• ":PRINt" on page 173
• ":PRINt?" on page 737
• PRINter, ":HARDcopy:PRINter:LIST" on page 290
• PROBe Commands:
  • ":CHANnel<n>:PROBe" on page 227
  • ":CHANnel<n>:PROBe:HEAD[:TYPE]" on page 228
  • ":CHANnel<n>:PROBe:ID" on page 229
  • ":CHANnel<n>:PROBe:SKEW" on page 230
  • ":CHANnel<n>:PROBe:STYPE" on page 231
  • ":EXTernal:PROBe" on page 258
• PROTection Commands:
  • ":CHANnel<n>:PROTection" on page 232
  • ":EXTernal:PROTection" on page 261
  • ":SYSTem:PROTection:LOCK" on page 452
• Pulse Width (GLITch), ":TRIGger:GLITch Commands" on page 520
• PWD Commands:
  • ":RECall:PWD" on page 402
  • ":SAVE:PWD" on page 415
  • PWIDth, ":MEASure:PWIDth" on page 330
• QUALifier Commands:
  • ":TRIGger:DURation:QUALifier" on page 496
  • ":TRIGger:GLITch:QUALifier" on page 526
  • ":TRIGger:IIC:TRIGger:QUALifier" on page 553
  • ":TRIGger:UART:QUALifier" on page 609
• RANGe Commands:
  • ":CHANnel<n>:RANGe" on page 233
  • ":EXTernal:RANGe" on page 262
  • ":FUNCTION:RANGe" on page 274
  • ":TIMebase:RANGe" on page 460
  • ":TIMebase:WINDow:RANGe" on page 466
  • ":TRIGger:DURation:RANGe" on page 497
  • ":TRIGger:GLITch:RANGe" on page 527
  • ":TRIGger:I2S:RANGe" on page 537
• "*RCL (Recall)" on page 127
• ":RECall:FIlename" on page 399
- ":RECall:IMAGe[:STARt]" on page 400
- "RECall:MASK[:STARt]" on page 401
- "RECall:PWD" on page 402
- "RECall:SETup[:STARt]" on page 403
- REFClock, "TIMebase:REFClock" on page 461
- REFerence Commands:
  - ":FUNCtion:REference" on page 275
  - ":TIMebase:REference" on page 462
- REJect, "TRIGger[:EDGE]:REJect" on page 505
- RESet Commands:
  - ":MEASure:STATistics:RESet" on page 341
  - ":MTESt:COUNt:RESet" on page 370
  - ":SBUS:CAN:COUNt:RESet" on page 425
  - ":SBUS:FLEXray:COUNt:RESet" on page 430
  - ":SBUS:UART:COUNt:RESet" on page 442
  - "TRIGger:SEQUence:RESet" on page 581
- RESults, ":MEASure:RESults" on page 331
- RISetime, ":MEASure:RISetime" on page 334
- RMODe Commands:
  - ":MTESt:RMODe" on page 378
  - ":MTESt:RMODe:FACTion:MEASure" on page 379
  - ":MTESt:RMODe:FACTion:PRINt" on page 380
  - ":MTESt:RMODe:FACTion:SAVE" on page 381
  - ":MTESt:RMODe:FACTion:STOP" on page 382
  - ":MTESt:RMODe:SIGMa" on page 383
  - ":MTESt:RMODe:TIME" on page 384
  - ":MTESt:RMODe:WAveforms" on page 385
- "Root (:) Commands" on page 139
- RSIGnal, "ACQuire:RSIGnal" on page 189
- "RST (Reset)" on page 128
- RTA, "TRIGger:M1553:RTA" on page 572
- RUMode Commands:
  - ":MTESt:RUMode" on page 733
  - ":MTESt:RUMode:SOFailure" on page 734
- ":RUN" on page 174
- **R**WIDth, "."!TRIGger:I2S:R!WIDth" on page 539
- **R**X, "."!TRIGger:UART:SOURce:RX" on page 610
- RXFRames, "."SBUS:UART:COUNt:RXFRames" on page 443

**S**AMPlépoint Commands:
- "."!TRIGger:CAN:SAMPlépoint" on page 486
- "."!TRIGger:LIN:SAMPlépoint" on page 563
- "*SAV (Save)" on page 131

**S**AVE Commands:
- "."MTEST:AMASK:{SAVE | STOR!e}" on page 729
- "."MTEST:RMODE:FACTion:SAV!e" on page 381
- "."SAVE:FİLeName" on page 406
- "."SAVE:IMAGe:AREA" on page 408
- "."SAVE:IMAGe:FACTors" on page 409
- "."SAVE:IMAGe:FORMat" on page 410
- "."SAVE:IMAGe:INKSaver" on page 411
- "."SAVE:IMAGe:PA!Lette" on page 412
- "."SAVE:IMAGe[:S!ARt]" on page 407
- "."SAVE:LISTer[:S!ARt]" on page 413
- "."SAVE:MASK[:S!ARt]" on page 414
- "."SAVE:PWD" on page 415
- "."SAVE:SETup[:S!ARt]" on page 416
- "."SAVE:WAveform:FORMat" on page 418
- "."SAVE:WAveform:LEN!Gth" on page 419
- "."SAVE:WAveform:SEG!Mented" on page 420
- "."SAVE:WAveform[:S!ARt]" on page 417
- "."SBUS:CAN:COUNt:ERRor" on page 423
- "."SBUS:CAN:COUNt:OVERload" on page 424
- "."SBUS:CAN:COUNt:RES!et" on page 425
- "."SBUS:CAN:COUNt:TOTal" on page 426
- "."SBUS:CAN:COUNt:UTILITYization" on page 427
- "."SBUS:DISPlay" on page 428
- "."SBUS:FLExray:COUNt:NULL" on page 429
- "."SBUS:FLExray:COUNt:RES!et" on page 430
- "."SBUS:FLExray:COUNt:SYNC" on page 431
- "."SBUS:FLExray:COUNt:TOTal" on page 432
• ".SBUS:I2S:BASE" on page 433
• ".SBUS:IIC:ASIZE" on page 434
• ".SBUS:LIN:PARity" on page 435
• ".SBUS:M1553:BASE" on page 436
• ".SBUS:MODE" on page 437
• ".SBUS:SPI:BITorder" on page 438
• ".SBUS:SPI:WIDTH" on page 439
• ".SBUS:UART:BASE" on page 440
• ".SBUS:UART:COUNt:ERRor" on page 441
• ".SBUS:UART:COUNt:RESSet" on page 442
• ".SBUS:UART:COUNt:RXFRames" on page 443
• ".SBUS:UART:COUNt:TXFRames" on page 444
• ".SBUS:UART:FRAMing" on page 445
• SCALE Commands:
  • ":CHANnel<n>:SCALE" on page 234
  • ":FUNCTION:SCALE" on page 276
  • ":MTEST:SCALE:BIND" on page 386
  • ":MTEST:SCALE:X1" on page 387
  • ":MTEST:SCALE:XDELta" on page 388
  • ":MTEST:SCALE:Y1" on page 389
  • ":MTEST:SCALE:Y2" on page 390
  • ":TIMebase:SCALE" on page 463
  • ":TIMebase:WINDOW:SCALE" on page 467
• SCRatch, ":MEASure:SCRatch" on page 716
• SDEViation, ":MEASure:SDEViation" on page 335
• ":SERial" on page 175
• SEGmented Commands:
  • ":ACQuire:SEGmented:ANALyze" on page 190
  • ":ACQuire:SEGmented:COUNt" on page 191
  • ":ACQuire:SEGmented:INDex" on page 192
  • ":SAVE:WAVEform:SEGmented" on page 420
  • ":WAVEform:SEGmented:COUNt" on page 639
  • ":WAVEform:SEGmented:TTAG" on page 640
• SETup Commands:
  • ":RECall:SETup[:STARt]" on page 403
• ":SAVE:SETup[:START]" on page 416
• ":SYSTem:SETup" on page 453
• SEQuence, ":TRIGger:SEQuence Commands" on page 576
• SHOW, ":MEASure:SHOW" on page 336
• SIGMa, ":MTESt:RMODe:SIGMa" on page 383
• SIGNal Commands:
  • ":TRIGger:CAN:SIGNal:BAUDrate" on page 487
  • ":TRIGger:CAN:SIGNal:DEFinition" on page 488
  • ":TRIGger:LIN:SIGNal:BAUDrate" on page 564
  • ":TRIGger:LIN:SIGNal:DEFinition" on page 741
• ":SINGle" on page 176
• SIZE Commands:
  • ":DIGital<n>:SIZE" on page 242
  • ":POD<n>:SIZE" on page 395
• SKEW, ":CHANnel<n>:PROBe:SKEW" on page 230
• SLOPe Commands:
  • ":TRIGger:EBURst:SLOPe" on page 501
  • ":TRIGger[:EDGE]:SLOPe" on page 506
  • ":TRIGger:I2S:CLOCk:SLOPe" on page 533
  • ":TRIGger:SPI:CLOCk:SLOPe" on page 585
• SOFailure, ":MTESt:RUMode:SOFailure" on page 734
• SOURce Commands:
  • ":DISPlay:SOURce" on page 252
  • ":FUNCTION:SOURce" on page 706
  • ":MEASure:SOURce" on page 337
  • ":MTESt:AMASk:SOURce" on page 365
  • ":MTESt:SOURce" on page 391
  • ":MTESt:TRIGger:SOURce" on page 736
  • ":TRIGger:CAN:SOURce" on page 489
  • ":TRIGger:FLExray:SOURce" on page 518
  • ":TRIGger:GLITch:SOURce" on page 528
  • ":TRIGger:I2S:SOURce:CLOCk" on page 540
  • ":TRIGger:I2S:SOURce:DATA" on page 541
  • ":TRIGger:I2S:SOURce:WSELect" on page 542
  • ":TRIGger:IIC[:SOURce]:CLOCk" on page 551
• ".:TRIGger:IIC:SOURce:DATA" on page 552
• ":TRIGger:LIN:SOURce" on page 565
• ":TRIGger:M1553:SOURce:LOWer" on page 573
• ":TRIGger:M1553:SOURce:UPPer" on page 574
• ":TRIGger:SPI:SOURce:CLOCK" on page 590
• ":TRIGger:SPI:SOURce:DATA" on page 591
• ":TRIGger:SPI:SOURce:FRAME" on page 592
• ":TRIGger:TV:SOURce" on page 597
• ":TRIGger:UART:SOURce:RX" on page 610
• ":TRIGger:UART:SOURce:TX" on page 611
• ":TRIGger:USB:SOURce:DMINus" on page 615
• ":TRIGger:USB:SOURce:DPLUS" on page 616
• ":WAVEform:SOURce" on page 641
• ":WAVEform:SOURce:SUBSource" on page 645
• SOURce1 Commands:
  • ":FUNCTION:GOFT:SOURce1" on page 270
  • ":FUNCTION:SOURce1" on page 277
• SOURce2 Commands:
  • ":FUNCTION:GOFT:SOURce2" on page 271
  • ":FUNCTION:SOURce2" on page 278
• SPAN, ":FUNCTION:SPAN" on page 279
• SPEed, ":TRIGger:USB:SPEed" on page 617
• SPI Commands:
  • ":SBUS:SPI:BITorder" on page 438
  • ":SBUS:SPI:WIDTH" on page 439
  • ":TRIGger:SPI Commands" on page 584
• SRATe, ":ACQuire:SRATe" on page 195
• "*SRE (Service Request Enable)" on page 132
• STANdard Commands:
  • ":TRIGger:LIN:STANdard" on page 566
  • ":TRIGger:TV:STANdard" on page 598
• STARt Commands:
  • ":CALibrate:START" on page 212
  • ":HARDcopy:START" on page 291
  • ":MTEST:{START | STOP}" on page 735
- ".:RECall:IMAGe[:STARt]" on page 400
- ".:RECall:MASK[:STARt]" on page 401
- ".:RECall:SETup[:STARt]" on page 403
- ".:SAVE:IMAGe[:STARt]" on page 407
- ".:SAVE:LISTer[:STARt]" on page 413
- ".:SAVE:MASK[:STARt]" on page 414
- ".:SAVE:SETup[:STARt]" on page 416
- ".:SAVE:WAVeform[:STARt]" on page 417
- STATistics Commands:
  - ".:MEASure:STATistics" on page 339
  - ".:MEASure:STATistics:INCRement" on page 340
  - ".:MEASure:STATistics:RESet" on page 341
- STATus Commands:
  - ":CALibrate:STATus" on page 213
  - ":STATus" on page 177
- "*STB (Read Status Byte)" on page 134
- STOP Commands:
  - ":MTESt:RMODe:FACTion:STOP" on page 382
  - ":MTESt:{STARt | STOP}" on page 735
  - ":STOP" on page 178
  - STORE, ":MTESt:AMASk:{SAVE | STORe}" on page 729
  - SUBSource, ":WAVeform:SOURce:SUBSource" on page 645
  - SWEep, ":TRIGger:SWEep" on page 479
  - SWITch, ":CALibrate:SWITch" on page 214
  - SYNC, ":SBUS:FLEXray:COUNt:SYNC" on page 431
  - SYNCbreak, ":TRIGger:LiN:SYNCbreak" on page 567
- ":SYSTem:DATE" on page 447
- ":SYSTem:DSP" on page 448
- ":SYSTem:ERRot" on page 449
- ":SYSTem:LOCK" on page 450
- ":SYSTem:PRECision" on page 451
- ":SYSTem:SETup" on page 453
- ":SYSTem:TIME" on page 455
- T
  - TDELta, ":MEASure:TDELta" on page 717
  - TEDGe, ":MEASure:TEDGe" on page 342
• TEMPerature, ":CALibrate:TEMPerature" on page 215
• ":TER (Trigger Event Register)" on page 179
• THReshold Commands:
  • ":CHANnel:THReshold" on page 697
  • ":DIGital<n>:THReshold" on page 243
  • ":MEASure:THResholds" on page 718
  • ":POD<n>:THReshold" on page 396
  • ":TRIGger:THReshold" on page 742
• THResholds, ":MEASure:THResholds" on page 718
• TIME Commands:
  • ":CALibrate:TIME" on page 216
  • ":MTESt:COUNT:TIME" on page 371
  • ":MTESt:RMODe:TIME" on page 384
  • ":SYSTem:TIME" on page 455
  • ":TIMebase:DELay" on page 739
  • ":TIMebase:MODE" on page 458
  • ":TIMebase:POSition" on page 459
  • ":TIMebase:RANGe" on page 460
  • ":TIMebase:REFClock" on page 461
  • ":TIMebase:REFERence" on page 462
  • ":TIMebase:SCALe" on page 463
  • ":TIMebase:VERNier" on page 464
  • ":TIMebase:WINDow:POSition" on page 465
  • ":TIMebase:WINDow:RANGe" on page 466
  • ":TIMebase:WINDow:SCALe" on page 467
• TIMEout, ":TRIGger:SPI:CLOCk:TIMEout" on page 586
• TIMER, ":TRIGger:SEQUence:TIMER" on page 582
• TITLE, ":MTESt:TITLe" on page 392
• TMAX, ":MEASure:TMAX" on page 719
• TMIN, ":MEASure:TMIN" on page 720
• TOTal Commands:
  • ":SBUS:CAN:COUNT:TOTal" on page 426
  • ":SBUS:FLEXray:COUNT:TOTal" on page 432
  • ":TRG (Trigger)" on page 136
• TRIGger Commands:
- ".MTEST:TRIGger:SOURce" on page 736
- ".TRIGger:CAN:TRIGger" on page 490
- ".TRIGger:FLEXray:TRIGger" on page 519
- ".TRIGger:I2S:TRIGger" on page 543
- ".TRIGger:IIC:TRIGger:QUALifier" on page 553
- ".TRIGger:IIC:TRIGger[:TYPE]" on page 554
- ".TRIGger:LIN:TRIGger" on page 568
- ".TRIGger:SEQUence:TRIGger" on page 583
- ".TRIGger:USB:TRIGger" on page 618
- ".TRIGger:HFReject" on page 472
- ".TRIGger:HOLDoff" on page 473
- ".TRIGger:LFIFty" on page 474
- ".TRIGger:MODE" on page 475
- ".TRIGger:NREJect" on page 476
- ".TRIGger:PATTernt" on page 477
- ".TRIGger:SWEeep" on page 479
- ".TRIGger:THReshold" on page 742
- ".TRIGger:CAN:ACKNowledge" on page 740
- ".TRIGger:CAN:PATTernt:DATA" on page 482
- ".TRIGger:CAN:PATTernt:DATA:LENGth" on page 483
- ".TRIGger:CAN:PATTernt:ID" on page 484
- ".TRIGger:CAN:PATTernt:ID:MODE" on page 485
- ".TRIGger:CAN:SAMPiepoint" on page 486
- ".TRIGger:CAN:SIGNal:BAUDrate" on page 487
- ".TRIGger:CAN:SIGNal:DEFinition" on page 488
- ".TRIGger:CAN:SOURce" on page 489
- ".TRIGger:CAN:TRIGger" on page 490
- ".TRIGger:DURation:GREaterthan" on page 493
- ".TRIGger:DURation:LESSthan" on page 494
- ".TRIGger:DURation:PATTernt" on page 495
- ".TRIGger:DURation:QUALifier" on page 496
- ".TRIGger:DURation:RANGe" on page 497
- ".TRIGger[:EDGE]:COUPling" on page 503
- ".TRIGger[:EDGE]:LEVel" on page 504
- ".TRIGger[:EDGE]:REJect" on page 505
6 Commands A-Z

- ".:TRIGger[:EDGE]:SLOPe" on page 506
- ".:TRIGger[:EDGE]:SOURce" on page 507
- ".:TRIGger:FLEXray:AUTosetup" on page 509
- ".:TRIGger:FLEXray:BAUDrate" on page 510
- ".:TRIGger:FLEXray:CHANnel" on page 511
- ".:TRIGger:FLEXray:ERRor:TYPE" on page 512
- ".:TRIGger:FLEXray:EVENt:TYPE" on page 513
- ".:TRIGger:FLEXray:FRAMe:CCBase" on page 514
- ".:TRIGger:FLEXray:FRAMe:CCRepetition" on page 515
- ".:TRIGger:FLEXray:FRAMe:ID" on page 516
- ".:TRIGger:FLEXray:FRAMe:TYPE" on page 517
- ".:TRIGger:FLEXray:SOURce" on page 518
- ".:TRIGger:FLEXray:TRIGger" on page 519
- ".:TRIGger:GLITch:GREaterthan" on page 522
- ".:TRIGger:GLITch:LESSthan" on page 523
- ".:TRIGger:GLITch:LEVel" on page 524
- ".:TRIGger:GLITch:POLarity" on page 525
- ".:TRIGger:GLITch:QUALifier" on page 526
- ".:TRIGger:GLITch:RANGe" on page 527
- ".:TRIGger:GLITch:SOURce" on page 528
- ".:TRIGger:HFReject" on page 472
- ".:TRIGger:HOLDoff" on page 473
- ".:TRIGger:I2S:ALIGnment" on page 531
- ".:TRIGger:I2S:AUDeo" on page 532
- ".:TRIGger:I2S:CLOCk:SLOPe" on page 533
- ".:TRIGger:I2S:PATTern:DATA" on page 534
- ".:TRIGger:I2S:PATTern:FORMAT" on page 536
- ".:TRIGger:I2S:RANGe" on page 537
- ".:TRIGger:I2S:RWIDTH" on page 539
- ".:TRIGger:I2S:SOURce:CLOCK" on page 540
- ".:TRIGger:I2S:SOURce:DATA" on page 541
- ".:TRIGger:I2S:SOURce:WSELect" on page 542
- ".:TRIGger:I2S:TRIGger" on page 543
- ".:TRIGger:I2S:TWIDTH" on page 545
- ".:TRIGger:I2S:WSLOW" on page 546
- "[:TRIGger:IIC:PA]TTern:ADDRes" on page 548
- "[:TRIGger:IIC:PA]TTern:DATA" on page 549
- "[:TRIGger:IIC:PA]TTern:DATa2" on page 550
- "[:TRIGger:IIC[:SOURce]:CLOCk" on page 551
- "[:TRIGger:IIC[:SOURce]:DATA" on page 552
- "[:TRIGger:IIC:TRIGger:QUALifier" on page 553
- "[:TRIGger:IIC:TRIGger[:TYPE]]" on page 554
- "[:TRIGger:LIN:ID" on page 558
- "[:TRIGger:LIN:PA]TTern:DATA" on page 559
- "[:TRIGger:LIN:PA]TTern:DATA:LENGth" on page 561
- "[:TRIGger:LIN:PA]TTern:FORMat" on page 562
- "[:TRIGger:LIN:SAMPLEpoint" on page 563
- "[:TRIGger:LIN:SIGNal:BAUDrate" on page 564
- "[:TRIGger:LIN:SIGNal:DEFinition" on page 741
- "[:TRIGger:LIN:SIGNal:SOURce" on page 565
- "[:TRIGger:LIN:STANDard" on page 566
- "[:TRIGger:LIN:SYNCbreak" on page 567
- "[:TRIGger:LIN:TRIGger" on page 568
- "[:TRIGger:M1553:AUTosetup" on page 570
- "[:TRIGger:M1553:PA]TTern:DATA" on page 571
- "[:TRIGger:M1553:RTA" on page 572
- "[:TRIGger:M1553:SOURce:LOWer" on page 573
- "[:TRIGger:M1553:SOURce:UPPer" on page 574
- "[:TRIGger:M1553:TYPE" on page 575
- "[:TRIGger:MODE" on page 475
- "[:TRIGger:NREJect" on page 476
- "[:TRIGger:PA]TTern" on page 477
- "[:TRIGger:SEQuence:COU]NT" on page 577
- "[:TRIGger:SEQuence:EDGE" on page 578
- "[:TRIGger:SEQuence:FIND" on page 579
- "[:TRIGger:SEQuence:PA]TTern" on page 580
- "[:TRIGger:SEQuence:RESet" on page 581
- "[:TRIGger:SEQuence:TIMer" on page 582
- "[:TRIGger:SEQuence:TRIGger" on page 583
- "[:TRIGger:SPI:CLOCk:SLOPe" on page 585
- :TRIGger:SPI:CLOCk:TIMEout" on page 586
- :TRIGger:SPI:FRAMing" on page 587
- :TRIGger:SPI:PATTern:DATA" on page 588
- :TRIGger:SPI:PATTern:WIDTH" on page 589
- :TRIGger:SPI:SOURce:CLOCk" on page 590
- :TRIGger:SPI:SOURce:DATA" on page 591
- :TRIGger:SPI:SOURce:FRAME" on page 592
- :TRIGger:SWEep" on page 479
- :TRIGger:THReshold" on page 742
- :TRIGger:TV:LINE" on page 594
- :TRIGger:TV:MODE" on page 595
- :TRIGger:TV:POLarity" on page 596
- :TRIGger:TV:SOURce" on page 597
- :TRIGger:TV:STANdard" on page 598
- :TRIGger:TV:TVMode" on page 743
- :TRIGger:UART:BASE" on page 601
- :TRIGger:UART:BAUDrate" on page 602
- :TRIGger:UART:BITorder" on page 603
- :TRIGger:UART:BURSt" on page 604
- :TRIGger:UART:DATA" on page 605
- :TRIGger:UART:IDLE" on page 606
- :TRIGger:UART:PARity" on page 607
- :TRIGger:UART:POLarity" on page 608
- :TRIGger:UART:QUALifier" on page 609
- :TRIGger:UART:SOURce:RX" on page 610
- :TRIGger:UART:SOURce:TX" on page 611
- :TRIGger:UART:TYPE" on page 612
- :TRIGger:UART:WIDTH" on page 613
- :TRIGger:USB:SOURce:DMINus" on page 615
- :TRIGger:USB:SOURce:DPLus" on page 616
- :TRIGger:USB:SPEed" on page 617
- :TRIGger:USB:TRIGger" on page 618
- "*TST (Self Test)*" on page 137
- TSTArt, "MEASure:TSTArt" on page 721
- TSTOP, "MEASure:TSTOP" on page 722
- TTAG, ":WAVEform:SEGmented:TTAG" on page 640
- TV, ":TRIGGER:TV Commands" on page 593
- TVALue, ":MEASURE:TVALue" on page 344
- TVOLT, ":MEASURE:TVOLT" on page 723
- TWIDTH, ":TRIGGER:I2S:TWIDTH" on page 545
- TX, ":TRIGGER:UART:SOURce:TX" on page 611
- TXFRAMES, ":SBUS:UART:COUNt:TXFRAMES" on page 444
- TYPE Commands:
  - ":ACQUIRE:TYPE" on page 196
  - ":CHANnel<n>:PROBE:HEAD[:TYPE]" on page 228
  - ":WAVEform:TYPE" on page 646
  - ":TRIGGER:FLEXRay:ERROR:TYPE" on page 512
  - ":TRIGGER:FLEXRay:Event:TYPE" on page 513
  - ":TRIGGER:FLEXRay:FRAME:TYPE" on page 517
  - ":TRIGGER:IIC:TRIGGER[:TYPE]" on page 554
  - ":TRIGGER:M1553:TYPE" on page 575
  - ":TRIGGER:UART:TYPE" on page 612
- UART Commands:
  - ":SBUS:UART:BASE" on page 440
  - ":SBUS:UART:COUNt:ERROR" on page 441
  - ":SBUS:UART:COUNt:RESET" on page 442
  - ":SBUS:UART:COUNt:RXFRAMES" on page 443
  - ":SBUS:UART:COUNt:TXFRAMES" on page 444
  - ":SBUS:UART:FRAMing" on page 445
  - ":TRIGGER:UART:BASE" on page 601
  - ":TRIGGER:UART:BAUDrate" on page 602
  - ":TRIGGER:UART:BITorder" on page 603
  - ":TRIGGER:UART:BURST" on page 604
  - ":TRIGGER:UART:DATA" on page 605
  - ":TRIGGER:UART:IDLE" on page 606
  - ":TRIGGER:UART:PARity" on page 607
  - ":TRIGGER:UART:POLarity" on page 608
  - ":TRIGGER:UART:QUALifier" on page 609
  - ":TRIGGER:UART:SOURce:RX" on page 610
  - ":TRIGGER:UART:SOURce:TX" on page 611
• "$\text{TRIGger:UART:TYPE}$" on page 612
• "$\text{TRIGger:UART:WIDTH}$" on page 613

• UNITs Commands:
  • "$\text{CHANnel<n>:UNITs}$" on page 235
  • "$\text{EXTernal:UNITs}$" on page 263
  • "$\text{MTESt:AMASk:UNITs}$" on page 366

• UNSigned, "$\text{WAVeform:UNS}igned$" on page 647

• UPPer Commands:
  • "$\text{MEASure:UPPer}$" on page 725
  • "$\text{TRIGger:M1553:SOURce:UPPer}$" on page 574

• USB, "$\text{TRIGger:USB \text{Commands}}$" on page 614

• UTILization, "$\text{SBUS:CAN:COUNT:UTILization}$" on page 427

• VAMPlitude, "$\text{MEASure:VAMPlitude}$" on page 346

• VAverage, "$\text{MEASure:VAverage}$" on page 347

• VBASe, "$\text{MEASure:VBASe}$" on page 348

• VDELta, "$\text{MEASure:VDELta}$" on page 726

• VECTors, "$\text{DISPlay:VECTors}$" on page 253

• VERNier, "$\text{CHANnel<n>:VERNier}$" on page 236

• "$\text{VIEW}$" on page 180

• VMAX, "$\text{MEASure:VMAX}$" on page 349

• VMIN, "$\text{MEASure:VMIN}$" on page 350

• VPP, "$\text{MEASure:VPP}$" on page 351

• VRATio, "$\text{MEASure:VRATio}$" on page 352

• VRMS, "$\text{MEASure:VRMS}$" on page 353

• VSTArt, "$\text{MEASure:VSTArt}$" on page 727

• VSTOP, "$\text{MEASure:VSTOP}$" on page 728

• VTIME, "$\text{MEASure:VTIME}$" on page 354

• VTOP, "$\text{MEASure:VTOP}$" on page 355

• "$\text{WAI (Wait To Continue)}$" on page 138

• WAVeform Commands:
  • "$\text{SAVE:WAVeform:FORMat}$" on page 418
  • "$\text{SAVE:WAVeform:LENGTH}$" on page 419
  • "$\text{SAVE:WAVeform[:STARt]}$" on page 417
  • "$\text{WAVeform:BYTEorder}$" on page 627
- ":WAVeform:COUNT" on page 628
- ":WAVeform:DATA" on page 629
- ":WAVeform:FORMat" on page 631
- ":WAVeform:POINts" on page 632
- ":WAVeform:POINts:MODE" on page 634
- ":WAVeform:PREAmble" on page 636
- ":WAVeform:SEGmented:COUNT" on page 639
- ":WAVeform:SEGmented:TTag" on page 640
- ":WAVeform:SOURce" on page 641
- ":WAVeform:SOURce:SUBSource" on page 645
- ":WAVeform:TYPE" on page 646
- ":WAVeform:UNSIGNED" on page 647
- ":WAVeform:VIEW" on page 648
- ":WAVeform:XINCReement" on page 649
- ":WAVeform:XORigin" on page 650
- ":WAVeform:XREFerence" on page 651
- ":WAVeform:YINCReement" on page 652
- ":WAVeform:YORigin" on page 653
- ":WAVeform:YREFerence" on page 654

WAVEforms Commands:
- ":MTESt:COUNt:WAVEforms" on page 372
- ":MTESt:RMODE:WAVEforms" on page 385

WIDTH Commands:
- ":SBUS:SPI:WIDTH" on page 439
- ":TRIGger:SPI:PATTERN:WIDTH" on page 589
- ":TRIGger:UART:WIDTH" on page 613

WINDow Commands:
- ":FUNCTION:WINDow" on page 280
- ":TIMebase:WINDow:POSIton" on page 465
- ":TIMebase:WINDow:RANGE" on page 466
- ":TIMebase:WINDow:SCALE" on page 467
- ":MEASure:WINDow" on page 356

WSELect, ":TRIGger:I2S:SOURce:WSELect" on page 542
WSLow, ":TRIGger:I2S:WSLow" on page 546

X1, ":MTESt:SCALE:X1" on page 387
• X1Position, ":MARKer:X1Position" on page 298
• X1Y1source, ":MARKer:X1Y1source" on page 299
• X2Position, ":MARKer:X2Position" on page 300
• X2Y2source, "MARKer:X2Y2source" on page 301
• XDELta Commands:
  • ":MARKer:XDELta" on page 302
  • ":MTESt:AMASk:XDELta" on page 367
  • ":MTESt:SCALe:XDELta" on page 388
• XINCrement, ":WAVEform:XINCrement" on page 649
• XMAX, ":MEASure:XMAX" on page 357
• XMIN, ":MEASure:XMIN" on page 358
• XORigin, ":WAVEform:XORigin" on page 650
• XREFerence, ":WAVEform:XREFerence" on page 651
Y
• Y1, ":MTESt:SCALe:Y1" on page 389
• Y1Position, ":MARKer:Y1Position" on page 303
• Y2, ":MTESt:SCALe:Y2" on page 390
• Y2Position, ":MARKer:Y2Position" on page 304
• YDELta Commands:
  • ":MARKer:YDELta" on page 305
  • ":MTESt:AMASk:YDELta" on page 368
• YINCrement, ":WAVEform:YINCrement" on page 652
• YORigin, ":WAVEform:YORigin" on page 653
• YREFerence, ":WAVEform:YREFerence" on page 654
# 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 788).

<table>
<thead>
<tr>
<th>Obsolete Command</th>
<th>Current Command Equivalent</th>
<th>Behavior Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALog&lt;n&gt;:BWLimit</td>
<td>:CHANnel&lt;n&gt;:BWLimit (see page 220)</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:COUpling</td>
<td>:CHANnel&lt;n&gt;:COUpling (see page 221)</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:INVert</td>
<td>:CHANnel&lt;n&gt;:INVert (see page 224)</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:LABel</td>
<td>:CHANnel&lt;n&gt;:LABel (see page 225)</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:OFFSet</td>
<td>:CHANnel&lt;n&gt;:OFFSet (see page 226)</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:PROBe</td>
<td>:CHANnel&lt;n&gt;:PROBe (see page 227)</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:PMODE</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>ANALog&lt;n&gt;:RANGE</td>
<td>:CHANnel&lt;n&gt;:RANGE (see page 233)</td>
<td></td>
</tr>
<tr>
<td>:CHANnel:ACTivity (see page 695)</td>
<td>:ACTivity (see page 142)</td>
<td></td>
</tr>
<tr>
<td>:CHANnel:LABel (see page 696)</td>
<td>:CHANnel&lt;n&gt;:LABel (see page 225) or :DIGital&lt;n&gt;:LABel (see page 240)</td>
<td>use CHANnel&lt;n&gt;:LABel for analog channels and use DIGital&lt;n&gt;:LABel for digital channels</td>
</tr>
<tr>
<td>:CHANnel:THReshold (see page 697)</td>
<td>:POD&lt;n&gt;:THReshold (see page 396) or :DIGital&lt;n&gt;:THReshold (see page 243)</td>
<td></td>
</tr>
<tr>
<td>:CHANnel2:SKEW (see page 698)</td>
<td>:CHANnel&lt;n&gt;:PROBe:SKEW (see page 230)</td>
<td></td>
</tr>
</tbody>
</table>
## Obsolete and Discontinued Commands

<table>
<thead>
<tr>
<th>Obsolete Command</th>
<th>Current Command Equivalent</th>
<th>Behavior Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;n&gt;:INPut (see page 699)</td>
<td>:CHANnel&lt;n&gt;:IMPedance (see page 223)</td>
<td></td>
</tr>
<tr>
<td>:CHANnel&lt;n&gt;:PMOdE (see page 700)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>:DISPlay:CONNect (see page 701)</td>
<td>:DISPlay:VECTors (see page 253)</td>
<td></td>
</tr>
<tr>
<td>:DISPlay:ORDer (see page 702)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>:ERASe (see page 703)</td>
<td>:CDISplay (see page 149)</td>
<td></td>
</tr>
<tr>
<td>:EXTernal:INPut (see page 704)</td>
<td>:EXTernal:IMPedance (see page 257)</td>
<td></td>
</tr>
<tr>
<td>:EXTernal:PMOdE (see page 705)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>FUNCTION1, FUNCTION2</td>
<td>FUNCTION Commands (see page 264)</td>
<td>ADD not included</td>
</tr>
<tr>
<td>:FUNCTION:SOURce (see page 706)</td>
<td>:FUNCTION:SOURce1 (see page 277)</td>
<td>Obsolete command has ADD, SUBtract, and MULTiply parameters; current command has GOFT parameter</td>
</tr>
<tr>
<td>:FUNCTION:VIEW (see page 707)</td>
<td>:FUNCTION:DISPlay (see page 268)</td>
<td></td>
</tr>
<tr>
<td>:HARDcopy:DESTination (see page 708)</td>
<td>:HARDcopy:FILename (see page 710)</td>
<td></td>
</tr>
<tr>
<td>:HARDcopy:DEVice (see page 709)</td>
<td>:HARDcopy:FORMat (see page 711)</td>
<td>PLOTter, THINkjet not supported; TIF, BMP, CSV, SElko added</td>
</tr>
<tr>
<td>:HARDcopy:FILename (see page 710)</td>
<td>:RECAll:FILename (see page 399)</td>
<td></td>
</tr>
<tr>
<td>:HARDcopy:FORMat (see page 711)</td>
<td>:HARDcopy:APRinter (see page 284)</td>
<td></td>
</tr>
<tr>
<td>:HARDcopy:GRAYscale (see page 712)</td>
<td>:HARDcopy:PALette (see page 289)</td>
<td></td>
</tr>
<tr>
<td>:HARDcopy:IGColors (see page 713)</td>
<td>:HARDcopy:INKSaver (see page 287)</td>
<td></td>
</tr>
<tr>
<td>Obsolete Command</td>
<td>Current Command Equivalent</td>
<td>Behavior Differences</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>:HARDcopy:PDRiver (see page 714)</td>
<td>:HARDcopy:APRinter (see page 284)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:LOWer (see page 715)</td>
<td>:MEASure:DEFine:THResholds (see page 316)</td>
<td>MEASure:DEFine:THResholds can define absolute values or percentage</td>
</tr>
<tr>
<td>:MEASure:SCRatch (see page 716)</td>
<td>:MEASure:CLEar (see page 314)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:TDELta (see page 717)</td>
<td>:MARKer:XDELta (see page 302)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:THResholds (see page 718)</td>
<td>:MEASure:DEFine:THResholds (see page 316)</td>
<td>MEASure:DEFine:THResholds can define absolute values or percentage</td>
</tr>
<tr>
<td>:MEASure:TMAX (see page 719)</td>
<td>:MEASure:XMAX (see page 357)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:TMIN (see page 720)</td>
<td>:MEASure:XMIN (see page 358)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:TSTArt (see page 721)</td>
<td>:MARKer:X1Position (see page 298)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:TSTOp (see page 722)</td>
<td>:MARKer:X2Position (see page 300)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:TVOLt (see page 723)</td>
<td>:MEASure:TVALue (see page 344)</td>
<td>TVALue measures additional values such as db, Vs, etc.</td>
</tr>
<tr>
<td>:MEASure:UPPer (see page 725)</td>
<td>:MEASure:DEFine:THResholds (see page 316)</td>
<td>MEASure:DEFine:THResholds can define absolute values or percentage</td>
</tr>
<tr>
<td>:MEASure:VDELta (see page 726)</td>
<td>:MARKer:YDELta (see page 305)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:VSTArt (see page 727)</td>
<td>:MARKer:Y1Position (see page 303)</td>
<td></td>
</tr>
<tr>
<td>:MEASure:VSTOp (see page 728)</td>
<td>:MARKer:Y2Position (see page 304)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:AMASK:{SAVE</td>
<td>STORe} (see page 729)</td>
<td>:SAVE:MASK[:STARt] (see page 414)</td>
</tr>
<tr>
<td>:MTESt:AVERage (see page 730)</td>
<td>:ACQuire:TYPE AVERage (see page 196)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:AVERage:COUNt (see page 731)</td>
<td>:ACQuire:COUNt (see page 185)</td>
<td></td>
</tr>
<tr>
<td>:MTESt:LOAD (see page 732)</td>
<td>:RECall:MASK[:STARt] (see page 401)</td>
<td></td>
</tr>
</tbody>
</table>
### Obsolete and Discontinued Commands

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

<table>
<thead>
<tr>
<th>Discontinued Command</th>
<th>Current Command Equivalent</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MTEnST:RUMode (see page 733)</td>
<td>:MTEnST:RMODe (see page 378)</td>
<td></td>
</tr>
<tr>
<td>:MTEnST:RUMode:SOfailure (see page 734)</td>
<td>:MTEnST:RMODe:FACTion:STOP (see page 382)</td>
<td></td>
</tr>
<tr>
<td>:MTEnST:{STRAT</td>
<td>STOP} (see page 735)</td>
<td>:RUN (see page 174) or :STOP (see page 178)</td>
</tr>
<tr>
<td>:MTEnST:TRIGger:SOURce (see page 736)</td>
<td>:TRIGger Commands (see page 468)</td>
<td>There are various commands for setting the source with different types of triggers.</td>
</tr>
<tr>
<td>:PRINt? (see page 737)</td>
<td>:DISPlay:DATA? (see page 247)</td>
<td></td>
</tr>
<tr>
<td>:TIMebase:DELay (see page 739)</td>
<td>:TIMebase:POSition (see page 459) or :TIMebase:WINDow:POSition (see page 465)</td>
<td>TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of zoomed (delayed) time base window.</td>
</tr>
<tr>
<td>:TRIGger:CAN:ACKNowledge (see page 740)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:LIN:SIGNal:DEFinition (see page 741)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:THReshold (see page 742)</td>
<td>:POD&lt;n&gt;:THReshold (see page 396) or :DIGital&lt;n&gt;:THReshold (see page 243)</td>
<td></td>
</tr>
<tr>
<td>:TRIGger:TV:TVMode (see page 743)</td>
<td>:TRIGger:TV:MODE (see page 595)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Discontinued Command</th>
<th>Current Command Equivalent</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTore</td>
<td>:DISPlay:PERSistence INfinite (see page 251)</td>
<td></td>
</tr>
<tr>
<td>CHANnel:MATH</td>
<td>:FUNCtion:OPERation (see page 273)</td>
<td>ADD not included</td>
</tr>
<tr>
<td>Discontinued Command</td>
<td>Current Command Equivalent</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHANnel&lt;n&gt;:PROTect</td>
<td>:CHANnel&lt;n&gt;:PROTection</td>
<td>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.</td>
</tr>
<tr>
<td>DISPlay:INVerse</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:COLumn</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:FREeze</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:GRID</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:LINE</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:PIXel</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:POSition</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:ROW</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DISPlay:TEXT</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>FUNCTION:MOVE</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>FUNCTION:PEAKs</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>HARDcopy:ADDRESS</td>
<td>none</td>
<td>Only parallel printer port is supported. GPIB printing not supported.</td>
</tr>
<tr>
<td>MASK</td>
<td>none</td>
<td>All commands discontinued, feature not available.</td>
</tr>
<tr>
<td>SYSTem:KEY</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>TEST:ALL</td>
<td>&quot;TST (Self Test)&quot; (see page 137)</td>
<td></td>
</tr>
<tr>
<td>TRACE subsystem</td>
<td>none</td>
<td>All commands discontinued, feature not available.</td>
</tr>
<tr>
<td>TRIGger:ADVanced subsystem</td>
<td></td>
<td>Use new GLITch, PATtern, or TV trigger modes.</td>
</tr>
<tr>
<td>TRIGger:TV:FIELd</td>
<td>:TRIGger:TV:MODE (see page 595)</td>
<td></td>
</tr>
<tr>
<td>TRIGger:TV:TVHFrej</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIGger:TV:VIR</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>VAUToscale</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
Obsolete and Discontinued Commands

**Discontinued Parameters**

Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision 7000 Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).
**:CHANnel:ACTivity**

(see page 788)

**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 142) 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.
Obsolete and Discontinued Commands

:CHANnel:LABEL

(see page 788)

Command Syntax

:CHANnel:LABEL <source_text><string>

<source_text> ::= {CHANnel1 | CHANnel2 | DIGital0,..,DIGital15}
<string> ::= quoted ASCII string

The :CHANnel:LABEL command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

NOTE

The :CHANnel:LABEL command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABEL command (see page 225) or :DIGital<n>:LABEL command (see page 240) for the InfiniiVision 7000 Series oscilloscopes.

Query Syntax

:CHANnel:LABEL?

The :CHANnel:LABEL? query returns the label associated with a particular analog channel.

Return Format

<string><NL>
<string> ::= quoted ASCII string
**:CHANnel:THReshold**

(see page 788)

**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 396) or :DIGital<n>:THReshold command (see page 243) for the InfiniiVision 7000 Series oscilloscopes.

**Query Syntax**

:\CHANnel:THReshold? <channel group>

The :CHANnel:THReshold? query returns the voltage and threshold text for a specific group of channels.

**Return Format**

<threshold type> [, <value>]<NL>

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value> ::= voltage for USERdef (float 32 NR3)

**NOTE**

- CMOS = 2.5V
- TTL = 1.5V
- ECL = -1.3V
- USERdef ::= -6.0V to 6.0V
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 230) 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 218
:CHANnel<n>:INPut

Command Syntax

:CHANnel<n>:INPut <impedance>

<impedance> ::= (ONEMeg | FIFTy)

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

<n> ::= (1 | 2) for the two channel oscilloscope models

The :CHANnel<n>:INPut command selects the input impedance setting for the specified channel. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE

The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see page 223) 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)
Obsolete and Discontinued Commands

:CHANnel<n>:PMODe

Command Syntax

:CHANnel<n>:PMODe <pmode value>

<pmode value> ::= (AUTo | MANual)

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

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODe sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE

The :CHANnel<n>:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax

:CHANnel<n>:PMODe?

The :CHANnel<n>:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

<pmode value><NL>

<pmode value> ::= (AUT | MAN)
**:DISPlay:CONNect**

0 (see page 788)

**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 253) 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 253
### :DISPlay:ORDer

(see page 788)

#### 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.

#### Return Format

<order><NL>

<order> ::= Unquoted ASCII string

#### See Also

- "DIGital<n>:POSition" on page 241

#### Example Code

' DISP_ORDER - Set the order the channels are displayed on the analyzer. You can enter between 1 and 32 channels at one time. If you leave out channels, they will not be displayed.

' Display ONLY channel 0 and channel 10 in that order. myScope.WriteString "::DISPLAY:ORDER 0,10"

Example program from the start: "VISA COM Example in Visual Basic" on page 814

#### NOTE

The :DISPlay:ORDer command is an obsolete command provided for compatibility to previous oscilloscopes. This command is only available on the MSO models.

A return value is included for each digital channel. A return value of NONE indicates that a channel is turned off.
:ERASe

0 (see page 788)

Command Syntax :ERASe

The :ERASe command erases the screen.

NOTE The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CDISplay command (see page 149) instead.
:EXTernal:INPut

(see page 788)

Command Syntax
:EXTernal:INPut <impedance>

<impedance> ::= {ONEMeg | FIFTy}

The :EXTernal:INPut command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE
The :EXTernal:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :EXTernal:IMPedance command (see page 257) instead.

Query Syntax
:EXTernal:INPut?

The :EXTernal:INPut? query returns the current input impedance setting for the external trigger.

Return Format
<impedance value><NL>

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

See Also
- "Introduction to :EXTernal Trigger Commands" on page 254
- "Introduction to :TRIGger Commands" on page 468
- ":CHANnel<n>:IMPedance" on page 223
:EXTernal:PMODe

(see page 788)

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)
7 Obsolete and Discontinued Commands

:FUNCtion:SOURce

(see page 788)

Command Syntax

:FUNCtion:SOURce <value>

<value> ::= (CHANnel<n> | ADD | SUBTract | MULTiply)

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

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

The :FUNCtion:SOURce command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the :FUNCtion:OPERation command for more information about selecting an operation). The :FUNCtion:SOURce command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCtion:OPERation command.

NOTE

The :FUNCtion:SOURce command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :FUNCtion:SOURce1 command (see page 277) instead.

Query Syntax

:FUNCtion:SOURce?

The :FUNCtion:SOURce? query returns the current source for function operations.

Return Format

<value><NL>

<value> ::= (CHAN<n> | ADD | SUBT | MULT)

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

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

See Also

- "Introduction to :FUNCtion Commands" on page 266
- ":FUNCtion:OPERation" on page 273
**:FUNCtion:VIEW**

0  (see page 788)

**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 FUNCtions 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 268) 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 788)

**Command Syntax**

: H A R D c o p y : D E S T i n a t i o n  <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 710) instead.

**Query Syntax**

: H A R D c o p y : D E S T i n a t i o n ?

The :HARDcopy:DESTination? query returns the selected hardcopy destination.

**Return Format**

<destination><NL>

<destination> ::= {CENT | FLOP}

**See Also**

- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:FORMat" on page 711
**:HARDcopy:DEVice**

(see page 788)

**Command Syntax**

>:HARDcopy:DEVice <device>

<device> ::= {TIFF | GIF | BMP | LAserjet | EPSon | DESKjet | BWDeskjet | SEIKo}

The :HARDcopy:DEVice command sets the hardcopy device type.

**NOTE**

BWDeskjet option refers to the monochrome Deskjet printer.

---

**NOTE**

The :HARDcopy:DEVice command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FORMat command (see page 711) instead.

**Query Syntax**

>:HARDcopy:DEVice?

The :HARDcopy:DEVice? query returns the selected hardcopy device type.

**Return Format**

<device><NL>

<device> ::= {TIFF | GIF | BMP | LAS | EPS | DESK | BWD | SEIK}
7 Obsolete and Discontinued Commands

:HARDcopy:FILename

(see page 788)

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 406) and RECall:FILename command (see page 399) 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 282
- ":HARDcopy:FORMat" on page 711
**:HARDcopy:FORMat**

(see page 788)

**Command Syntax**

`:HARDcopy:FORMat <format>`

<format> ::= {BMP[24bit] | BMP8bit | PNG | CSV | ASCiixy | BINary | PRINter0 | PRINter1}

The **HARDcopy:FORMat** command sets the hardcopy format type.

PRINter0 and PRINter1 are only valid when printers are connected to the oscilloscope's USB ports. (The first printer connected/identified is PRINter0 and the second is PRINter1.)

**NOTE**

The **:HARDcopy:FORMat** command is an obsolete command provided for compatibility to previous oscilloscopes. Use the **:SAVE:IMAGe:FORMat** (see page 410), **:SAVE:WAVeform:FORMat** (see page 418), and **:HARDcopy:APRinter** (see page 284) commands instead.

**Query Syntax**

`:HARDcopy:FORMat?`

The **:HARDcopy:FORMat?** query returns the selected hardcopy format type.

**Return Format**

<format><NL>

<format> ::= {BMP | BMP8 | PNG | CSV | ASC | BIN | PRIN0 | PRIN1}

**See Also**

- "Introduction to **HARDcopy Commands**" on page 282
7 Obsolete and Discontinued Commands

**:HARDcopy:GRAYscale**

0 (see page 788)

**Command Syntax**

```plaintext
: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 289) instead. ("."HARDcopy:GRAYscale ON" is the same as ":HARDcopy:PALette GRAYscale" and ".HARDcopy:GRAYscale OFF" is the same as ":HARDcopy:PALette COlor").

**Query Syntax**

```plaintext
: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 282
**HARDcopy:IGColors**

0 (see page 788)

**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 287) 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 282
**:HARDcopy:PDRiver**

(see page 788)

**Command Syntax**

`:HARDcopy:PDRiver <driver>`

```
<driver> ::= {AP2Xxx | AP21xx | {AP2560 | AP25} | {Dj350 | DJ35} |
           DJ6xx | {DJ630 | DJ63} | DJ6Special | DJ6Photo |
           DJ8Special | DJ8xx | DJ9Vip | OJPBarxx50 | 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 284) 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 | DJ6S | DJ6P |
           DJ8S | DJ8 | DJ9V | OJPBar | DJ9 | GVIP | DJ55 | PS10 |
           PS47 | CLAS | MLAS | LJF | POST}
```

**See Also**

- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:FORMat" on page 711
:MEASURE:LOWER

(see page 788)

**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 316) 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 312
- ":MEASURE:THRESHolds" on page 718
- ":MEASURE:UPPER" on page 725
### :MEASure:SCRatch

(see page 788)

**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 314) instead.
:MEASure:TDELta

(see page 788)

Query Syntax

:MEASure:TDELta?

The :MEASure:TDELta? query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

\[ T_{\text{delta}} = T_{\text{stop}} - T_{\text{start}} \]

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 302) instead.

Return Format

<value><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 296
- "Introduction to :MEASure Commands" on page 312
- ":MARKer:X1Position" on page 298
- ":MARKer:X2Position" on page 300
- ":MARKer:XDELta" on page 302
- ":MEASure:TSTArt" on page 721
- ":MEASure:TSTOp" on page 722
**MEASURE:**THResholds

(see page 788)

**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 316) 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 312
- ":MEASURE:LOWer" on page 715
- ":MEASURE:UPPer" on page 725
:MEASure:TMAX

(see page 788)

Command Syntax
:MEASure:TMAX [<source>]

<sourced> ::= (CHANnel<n> | FUNCTION | MATH)

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

The :MEASure:TMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

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

The :MEASure:TMAX? query returns the horizontal axis value at which the maximum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format
<value><NL>

<value> ::= time at maximum in NR3 format

See Also
- "Introduction to :MEASure Commands" on page 312
- ":MEASure:TMIN" on page 720
- ":MEASure:XMAX" on page 357
- ":MEASure:XMIN" on page 358

NOTE
The :MEASure:TMAX command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMAX command (see page 357) instead.
7 Obsolete and Discontinued Commands

:MEASure:TMIN

(see page 788)

Command Syntax

:MEASure:TMIN [<source>]

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

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

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

The :MEASure:TMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

Query Syntax

:MEASure:TMIN? [<source>]

The :MEASure:TMIN? query returns the horizontal axis value at which the minimum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

<value><NL>

<value> ::= time at minimum in NR3 format

NOTE

The :MEASure:TMIN command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMIN command (see page 358) instead.

See Also

- "Introduction to :MEASure Commands" on page 312
- "MEASure:TMAX" on page 719
- "MEASure:XMAX" on page 357
- "MEASure:XMIN" on page 358
### :MEASURE:TSTArt

(see page 788)

**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 790). 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 298) 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 296
- "Introduction to :MEASURE Commands" on page 312
- ":MARKer:X1Position" on page 298
- ":MARKer:X2Position" on page 300
- ":MARKer:XDELta" on page 302
- ":MEASURE:TDELta" on page 717
- ":MEASURE:TSTOp" on page 722
**:MEASure:TSTOp**

(see page 788)

**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 790). 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 300) 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 296
- "Introduction to :MEASure Commands" on page 312
- ":MARKer:X1Position" on page 298
- ":MARKer:X2Position" on page 300
- ":MARKer:XDELta" on page 302
- ":MEASure:TDELta" on page 717
- ":MEASure:TSTArt" on page 721
:MEASure:TVOLt

(see page 788)

**Query Syntax**

:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

*<value>* ::= the voltage level that the waveform must cross.

*<slope>* ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

*<occurrence>* ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

*<source>* ::= ({<digital channels> | CHANnel<n> | FUNCtion | MATH})

*<digital channels>* ::= {DIGital0,...,DIGital15} for the MSO models

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

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

When the :MEASure:TVOLt? query is sent, the displayed signal is searched for the specified voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified voltage can be negative or positive. To specify a negative voltage, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

**NOTE**

The :MEASure:TVOLt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:TVALue command (see page 344) for the InfiniiVision 7000 Series oscilloscopes.

**Return Format**

<value><NL>
<value> ::= time in seconds of the specified voltage crossing
in NR3 format
**:MEASure:UPPer**

(see page 788)

**Command Syntax**

:MEASure:UPPer <value>

The :MEASure:UPPer command sets the upper measurement threshold value. This value and the LOWer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

**NOTE**

The :MEASure:UPPer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 316) 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 312
- ":MEASure:LOWer" on page 715
- ":MEASure:THResholds" on page 718
:MEASure:VDELta

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 305) instead.

Return Format

<value><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 296
- "Introduction to :MEASure Commands" on page 312
- ":MARKer:Y1Position" on page 303
- ":MARKer:Y2Position" on page 304
- ":MARKer:YDELta" on page 305
- ":MEASure:TDELta" on page 717
- ":MEASure:TSTArt" on page 721
:MEASure:VSTArt

(see page 788)

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 790). 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 303) 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 296
- "Introduction to :MEASure Commands" on page 312
- "MARKer:Y1Position" on page 303
- "MARKer:Y2Position" on page 304
- "MARKer:YDELta" on page 305
- "MARKer:X1Y1source" on page 299
- "MEASure:SOURce" on page 337
- "MEASure:TDELta" on page 717
- "MEASure:TSTArt" on page 721
**:MEASure:VSTOp**

(see page 788)

**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 790). 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 304) 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 296
- "Introduction to :MEASure Commands" on page 312
- ":MARKer:Y1Position" on page 303
- ":MARKer:Y2Position" on page 304
- ":MARKer:YDELta" on page 305
- ":MARKer:X2Y2source" on page 301
- ":MEASure:SOURce" on page 337
- ":MEASure:TDELta" on page 717
- ":MEASure:TSTArt" on page 721
:MTESt:AMASk:{SAVE | STOrE}

(see page 788)

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 414) instead.

See Also

• "Introduction to :MTESt Commands" on page 361
7 Obsolete and Discontinued Commands

:MTESt:AVERAge

0 (see page 788)

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 196) 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 361
• ":MTESt:AVERAge:COUNt" on page 731
:MTEST:AVERage:COUNt

(see page 788)

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 185) 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 361
- ":MTEST:AVERage" on page 730
:MTEST:LOAD

(see page 788)

**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[:STAr] command (see page 401) instead.

**See Also**

- "Introduction to :MTEST Commands" on page 361
- ":MTEST:AMASK:{SAVE | STORe}" on page 729
:MTESt:RUMode

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.

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 361
- ":MTESt:RUMode:SOFailure" on page 734

NOTE

The :MTESt:RUMode command is obsolete and is provided for backward compatibility to
previous oscilloscopes. Use the :MTESt:RMODe command (see page 378) instead.
:MTESt:RUMode:SOFailure

0 (see page 788)

Command Syntax

:MWTESt: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 382) 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 361
- "::MTESt::RUMode" on page 733
:MTESSt:{STAr | STOP}  
(see page 788)

Command Syntax  
:MTESSt:{STAr | STOP}  
The :MTESSt:{STAr | STOP} command starts or stops the acquisition system.

NOTE  
The :MTESSt:STAr and :MTESSt: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 :MTESSt Commands" on page 361
**:MTESt:TRIGger:SOURce**

(see page 788)

**Command Syntax**

`:MTESt:TRIGger:SOURce <source>`

<source> ::= CHANnel<n>

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

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

The :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 468) instead.

**Query Syntax**

`:MTESt:TRIGger:SOURce?`

The :MTESt:TRIGger:SOURce? query returns the currently selected trigger source.

**Return Format**

<source> ::= CHAN<n>

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

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

**See Also**

- "Introduction to :MTESt Commands" on page 361
:PRINt?

(see page 788)

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 247) instead.

<table>
<thead>
<tr>
<th>Print Option</th>
<th>:PRINt command</th>
<th>:PRINt? query</th>
<th>Query Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLor</td>
<td>Sets palette=COLor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRAYscale</td>
<td>Sets palette=GRAYscale</td>
<td>palette=COLor</td>
<td></td>
</tr>
<tr>
<td>PRINter0,1</td>
<td>Causes the USB printer #0,1 to be selected as destination (if connected)</td>
<td>Not used</td>
<td>N/A</td>
</tr>
<tr>
<td>BMP8bit</td>
<td>Sets print format to 8-bit BMP</td>
<td>Selects 8-bit BMP formatting for query</td>
<td>N/A</td>
</tr>
<tr>
<td>BMP</td>
<td>Sets print format to BMP</td>
<td>Selects BMP formatting for query</td>
<td>N/A</td>
</tr>
<tr>
<td>FACTors</td>
<td>Selects outputting of additional settings information for :PRINT</td>
<td>Not used</td>
<td>N/A</td>
</tr>
<tr>
<td>NOFactors</td>
<td>Deselects outputting of additional settings information for :PRINT</td>
<td>Not used</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Old Print Option: Is Now:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HIRes</td>
<td>COLor</td>
</tr>
<tr>
<td>LORes</td>
<td>GRAYscale</td>
</tr>
<tr>
<td>PARallel</td>
<td>PRINter0</td>
</tr>
</tbody>
</table>
7 Obsolete and Discontinued Commands

<table>
<thead>
<tr>
<th>Old Print Option</th>
<th>Is Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>invalid</td>
</tr>
<tr>
<td>PCL</td>
<td>invalid</td>
</tr>
</tbody>
</table>

**NOTE**

The PRINt? query is not a core command.

**See Also**

- "Introduction to Root (:) Commands" on page 141
- "Introduction to :HARDcopy Commands" on page 282
- ":HARDcopy:FORMat" on page 711
- ":HARDcopy:FACTors" on page 285
- ":HARDcopy:GRAYscale" on page 712
- ":DISPlay:DATA" on page 247
:TIMebase:DELay

(see page 788)

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 462).

NOTE

The :TIMebase:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMebase:POSition command (see page 459) instead.

Query Syntax

:TIMebase:DELay?

The :TIMebase:DELay query returns the current delay value.

Return Format

<delay_value><NL>

<delay_value> ::= time from trigger to display reference in seconds in NR3 format.

Example Code

' TIMEBASE_DELAY - Sets the time base delay. This delay is the internal time between the trigger event and the onscreen delay reference point.

' Set time base delay to 0.0.
myScope.WriteString ':TIMEBASE:DELAY 0.0'

Example program from the start: "VISA COM Example in Visual Basic" on page 814
**:TRIGger:CAN:ACKNowledge**

0 (see page 788)

**Command Syntax**

:TRIGger:CAN:ACKNowledge <value>

<value> ::= (0 | OFF)

This command was used with the N2758A CAN trigger module for 54620/54640 Series mixed-signal oscilloscopes. The InfiniiVision 7000 Series oscilloscopes do not support the N2758A CAN trigger module.

**Query Syntax**

:TRIGger:CAN:ACKNowledge?

The :TRIGger:CAN:ACKNowledge? query returns the current CAN acknowledge setting.

**Return Format**

<value><NL>

<value> ::= 0

**See Also**

- "Introduction to :TRIGger Commands" on page 468
- ":TRIGger:MODE" on page 475
- ":TRIGger:CAN:TRIGger" on page 490
:TRIGger:LIN:SIGNal:DEFinition

(see page 788)

Command Syntax

:TRIGger:LIN:SIGNal:DEFinition <value>

<value> ::= {LIN | RX | TX}

The :TRIGger:LIN:SIGNal:DEFinition command sets the LIN signal type. These signals can be set to:

Dominant low signals:
- LIN – the actual LIN single-end bus signal line.
- RX – the Receive signal from the LIN bus transceiver.
- TX – the Transmit signal to the LIN bus transceiver.

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

Query Syntax

:TRIGger:LIN:SIGNal:DEFinition?


Return Format

<Value><NL>
<Value> ::= LIN

See Also
- "Introduction to :TRIGger Commands" on page 468
- "TRIGger:MODE" on page 475
- "TRIGger:LIN:SIGNal:BAUDrate" on page 564
- "TRIGger:LIN:SOURce" on page 565
7 Obsolete and Discontinued Commands

:TRIGger:THReshold

(see page 788)

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 396), :DIGital<n>:THReshold command (see page 243), or :TRIGger[:EDGE]:LEVel command (see page 504) for the InfiniiVision 7000 Series oscilloscopes.

Query Syntax

:TRIGger:THReshold? <channel group>

The :TRIGger:THReshold? query returns the voltage and threshold text for analog channel 1 or 2, or POD1 or POD2.

Return Format

<threshold type> [, <value>] <NL>

<threshold type> ::= {CMOS | ECL | TTL | USER}

CMOS ::= 2.5V

TTL ::= 1.5V

ECL ::= -1.3V

USERdef ::= range from -8.0V to +8.0V.

<value> ::= voltage for USERdef (a floating-point number in NR1).
**:TRIGger:TV:TVMode**

(see page 788)

**Command Syntax**

`:TRIGger:TV:TVMode <mode>`

`<mode>` ::= \{FIELd1 | FIELd2 | AFIelds | ALINes | LINE | VERTical | LFIeld1 | LFIeld2 | LALTernate | LVERtical\}

The `:TRIGger:TV:MODE` command selects the TV trigger mode and field. The `LVERtical` parameter is only available when `:TRIGger:TV:STANdard` is `GENeric`. The `LALTernate` parameter is not available when `:TRIGger:TV:STANdard` is `GENeric` (see page 598).

Old forms for `<mode>` are accepted:

<table>
<thead>
<tr>
<th><code>&lt;mode&gt;</code></th>
<th>Old Forms Accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIEl1</td>
<td>F1</td>
</tr>
<tr>
<td>FIEl2</td>
<td>F2</td>
</tr>
<tr>
<td>AFIeld</td>
<td>ALLFields, ALLFLDS</td>
</tr>
<tr>
<td>ALINes</td>
<td>ALLLines</td>
</tr>
<tr>
<td>LFIeld1</td>
<td>LINEF1, LINEFIELD1</td>
</tr>
<tr>
<td>LFIeld2</td>
<td>LINEF2, LINEFIELD2</td>
</tr>
<tr>
<td>LALTernate</td>
<td>LINEAlt</td>
</tr>
<tr>
<td>LVERtical</td>
<td>LINEVert</td>
</tr>
</tbody>
</table>

**NOTE**

The `:TRIGger:TV:TVMode` command is an obsolete command provided for compatibility to previous oscilloscopes. Use the `:TRIGger:TV:MODE` command (see page 595) instead.

**Query Syntax**

`:TRIGger:TV:TVMode?`

The `:TRIGger:TV:TVMode?` query returns the TV trigger mode.

**Return Format**

`<value><NL>`

`<value>` ::= \{FIEL1 | FIEL2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2 | LALT | LVER\}
Obsolete and Discontinued Commands
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
Error Messages

-314, Save/recall memory lost

-313, Calibration memory lost

-311, Memory error

-310, System error

-300, Device specific error

-278, Macro header not found

-277, Macro redefinition not allowed

-276, Macro recursion error

-273, Illegal macro label

-272, Macro execution error

-258, Media protected

-257, File name error

-256, File name not found
-255, Directory full

-254, Media full

-253, Corrupt media

-252, Missing media

-251, Missing mass storage

-250, Mass storage error

-241, Hardware missing

This message can occur when a feature is unavailable or unlicensed.

For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.

-240, Hardware error

-231, Data questionable

-230, Data corrupt or stale

-224, Illegal parameter value
-223, Too much data

-222, Data out of range

-221, Settings conflict

-220, Parameter error

-200, Execution error

-183, Invalid inside macro definition

-181, Invalid outside macro definition

-178, Expression data not allowed

-171, Invalid expression

-170, Expression error

-168, Block data not allowed

-161, Invalid block data

-158, String data not allowed
-151, Invalid string data

-150, String data error

-148, Character data not allowed

-138, Suffix not allowed

-134, Suffix too long

-131, Invalid suffix

-128, Numeric data not allowed

-124, Too many digits

-123, Exponent too large

-121, Invalid character in number

-120, Numeric data error

-114, Header suffix out of range

-113, Undefined header
-112, Program mnemonic too long

-109, Missing parameter

-108, Parameter not allowed

-105, GET not allowed

-104, Data type error

-103, Invalid separator

-102, Syntax error

-101, Invalid character

-100, Command error

+10, Software Fault Occurred

+100, File Exists

+101, End-Of-File Found

+102, Read Error
+103, Write Error

+104, Illegal Operation

+105, Print Canceled

+106, Print Initialization Failed

+107, Invalid Trace File

+108, Compression Error

+109, No Data For Operation

A remote operation wants some information, but there is no information available. For example, you may request a stored TIFF image using the :DISPlay:DATA? query, but there may be no image stored.

+112, Unknown File Type

+113, Directory Not Supported
8  Error Messages
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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>14</td>
<td>Bat ON</td>
</tr>
<tr>
<td>13</td>
<td>Bat ON</td>
</tr>
<tr>
<td>12</td>
<td>Bat ON</td>
</tr>
<tr>
<td>11</td>
<td>Bat ON</td>
</tr>
<tr>
<td>10</td>
<td>Bat ON</td>
</tr>
<tr>
<td>9</td>
<td>Bat ON</td>
</tr>
<tr>
<td>8</td>
<td>Bat ON</td>
</tr>
<tr>
<td>7</td>
<td>Bat ON</td>
</tr>
<tr>
<td>6</td>
<td>Bat ON</td>
</tr>
<tr>
<td>5</td>
<td>Bat ON</td>
</tr>
<tr>
<td>4</td>
<td>Bat ON</td>
</tr>
<tr>
<td>3</td>
<td>Bat ON</td>
</tr>
<tr>
<td>2</td>
<td>Bat ON</td>
</tr>
<tr>
<td>1</td>
<td>Bat ON</td>
</tr>
<tr>
<td>0</td>
<td>Bat ON</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Ext Trig Fault</td>
</tr>
<tr>
<td>14</td>
<td>Ext Trig Fault</td>
</tr>
<tr>
<td>13</td>
<td>Ext Trig Fault</td>
</tr>
<tr>
<td>12</td>
<td>Ext Trig Fault</td>
</tr>
<tr>
<td>11</td>
<td>Chan1 Fault</td>
</tr>
<tr>
<td>10</td>
<td>Chan1 Fault</td>
</tr>
<tr>
<td>9</td>
<td>Chan1 Fault</td>
</tr>
<tr>
<td>8</td>
<td>Chan1 Fault</td>
</tr>
<tr>
<td>7</td>
<td>Chan2 Fault</td>
</tr>
<tr>
<td>6</td>
<td>Chan2 Fault</td>
</tr>
<tr>
<td>5</td>
<td>Chan2 Fault</td>
</tr>
<tr>
<td>4</td>
<td>Chan2 Fault</td>
</tr>
<tr>
<td>3</td>
<td>Chan3 Fault</td>
</tr>
<tr>
<td>2</td>
<td>Chan3 Fault</td>
</tr>
<tr>
<td>1</td>
<td>Chan3 Fault</td>
</tr>
<tr>
<td>0</td>
<td>Chan3 Fault</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Ext Trig OVL</td>
</tr>
<tr>
<td>14</td>
<td>Ext Trig OVL</td>
</tr>
<tr>
<td>13</td>
<td>Ext Trig OVL</td>
</tr>
<tr>
<td>12</td>
<td>Ext Trig OVL</td>
</tr>
<tr>
<td>11</td>
<td>Chan1 OVL</td>
</tr>
<tr>
<td>10</td>
<td>Chan1 OVL</td>
</tr>
<tr>
<td>9</td>
<td>Chan1 OVL</td>
</tr>
<tr>
<td>8</td>
<td>Chan1 OVL</td>
</tr>
<tr>
<td>7</td>
<td>Chan2 OVL</td>
</tr>
<tr>
<td>6</td>
<td>Chan2 OVL</td>
</tr>
<tr>
<td>5</td>
<td>Chan2 OVL</td>
</tr>
<tr>
<td>4</td>
<td>Chan2 OVL</td>
</tr>
<tr>
<td>3</td>
<td>Chan3 OVL</td>
</tr>
<tr>
<td>2</td>
<td>Chan3 OVL</td>
</tr>
<tr>
<td>1</td>
<td>Chan3 OVL</td>
</tr>
<tr>
<td>0</td>
<td>Chan3 OVL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>14</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>13</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>12</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>11</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>10</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>9</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>8</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>7</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>6</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>5</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>4</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>3</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>2</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>1</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>0</td>
<td>PLL Locked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>14</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>13</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>12</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>11</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>10</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>9</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>8</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>7</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>6</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>5</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>4</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>3</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>2</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>1</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>0</td>
<td>PLL Locked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>14</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>13</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>12</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>11</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>10</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>9</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>8</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>7</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>6</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>5</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>4</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>3</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>2</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>1</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>0</td>
<td>PLL Locked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>14</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>13</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>12</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>11</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>10</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>9</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>8</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>7</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>6</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>5</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>4</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>3</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>2</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>1</td>
<td>PLL Locked</td>
</tr>
<tr>
<td>0</td>
<td>PLL Locked</td>
</tr>
</tbody>
</table>

To bits in Operation Status Condition Register:
- HWE
- DVLR
- MTE

To bits in Operation Status Condition Register:
- HWE
- DVLR
- MTE
The status register bits are described in more detail in the following tables:

- Table 44
- Table 42
- Table 52
- Table 53
- Table 55
- Table 47
- Table 48
- Table 50
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.

```python
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.

```vbnet
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.

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

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

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

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

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

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

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

When using the Agilent VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEEBlock 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 \texttt{:SYSTem:DS}P 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.

```plaintext
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN bit</td>
<td>bit 3</td>
<td>Is set whenever the instrument goes from a stop state to a single or running state.</td>
</tr>
<tr>
<td>WAIT TRIG bit</td>
<td>bit 5</td>
<td>Is set by the Trigger Armed Event Register and indicates that the trigger is armed.</td>
</tr>
<tr>
<td>MTE bit</td>
<td>bit 9</td>
<td>Comes from the Mask Test Event Registers.</td>
</tr>
<tr>
<td>OVLR bit</td>
<td>bit 11</td>
<td>Is set whenever a 50Ω input overload occurs.</td>
</tr>
<tr>
<td>HWE bit</td>
<td>bit 12</td>
<td>Comes from the Hardware Event Registers.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN bit</td>
<td>bit 3</td>
<td>Is set whenever the instrument is not stopped.</td>
</tr>
<tr>
<td>WAIT TRIG bit</td>
<td>bit 5</td>
<td>Is set by the Trigger Armed Event Register and indicates that the trigger is armed.</td>
</tr>
<tr>
<td>MTE bit</td>
<td>bit 9</td>
<td>Comes from the Mask Test Event Registers.</td>
</tr>
<tr>
<td>OVLR bit</td>
<td>bit 11</td>
<td>Is set whenever a 50Ω input overload occurs.</td>
</tr>
<tr>
<td>HWE bit</td>
<td>bit 12</td>
<td>Comes from the Hardware Event Registers.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 1 OVL</td>
<td>bit 0</td>
<td>Overload has occurred on Channel 1 input.</td>
</tr>
<tr>
<td>Channel 2 OVL</td>
<td>bit 1</td>
<td>Overload has occurred on Channel 2 input.</td>
</tr>
<tr>
<td>Channel 3 OVL</td>
<td>bit 2</td>
<td>Overload has occurred on Channel 3 input.</td>
</tr>
<tr>
<td>Channel 4 OVL</td>
<td>bit 3</td>
<td>Overload has occurred on Channel 4 input.</td>
</tr>
<tr>
<td>External Trigger OVL</td>
<td>bit 4</td>
<td>Overload has occurred on External Trigger input.</td>
</tr>
<tr>
<td>Channel 1 Fault</td>
<td>bit 6</td>
<td>Fault has occurred on Channel 1 input.</td>
</tr>
<tr>
<td>Channel 2 Fault</td>
<td>bit 7</td>
<td>Fault has occurred on Channel 2 input.</td>
</tr>
<tr>
<td>Channel 3 Fault</td>
<td>bit 8</td>
<td>Fault has occurred on Channel 3 input.</td>
</tr>
<tr>
<td>Channel 4 Fault</td>
<td>bit 9</td>
<td>Fault has occurred on Channel 4 input.</td>
</tr>
<tr>
<td>External Trigger Fault</td>
<td>bit 10</td>
<td>Fault has occurred on External Trigger input.</td>
</tr>
</tbody>
</table>
Hardware Event Event Register (:HWEReGister[:EVENt])

This register hosts the Bat On bit (bit 0).
- The Bat On bit is set whenever the instrument is operating on battery power.
Hardware Event Condition Register (\texttt{:HWERegister:CONDition})

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

- The \texttt{:HWERegister:CONDition?} query returns the value of the Hardware Event Condition Register.
- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.
- The Bat On bit is set whenever the instrument is operating on battery power.
9  Status Reporting

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

The Mask Test Event Event Register register hosts these bits:

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>bit 0</td>
<td>Is set when the mask test is complete.</td>
</tr>
<tr>
<td>Fail</td>
<td>bit 1</td>
<td>Is set when there is a mask test failure.</td>
</tr>
<tr>
<td>Started</td>
<td>bit 8</td>
<td>Is set when mask testing is started.</td>
</tr>
<tr>
<td>Auto Mask</td>
<td>bit 10</td>
<td>Is set when auto mask creation is completed.</td>
</tr>
</tbody>
</table>

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

Status Reporting Decision Chart

1. Do you want to do status reporting?
   - yes: Reset the instrument and clear the status registers:
     ```
     myScope.WriteString "*RST"
     myScope.WriteString "*CLS"
     ```
   - no: Do you want to send a Service Request (SRQ) interrupt to the controller?
     - no: Your programs can read the status registers instead.
     - yes: Do you want to report events monitored by the Standard Event Status Register?
       - yes: Use the "ESE" common command to enable the bits you want to use to generate the ESB summary bit in the Status Byte Register.
       - no: Use the "SRE" common command to enable the bits you want to generate the RQS/MSS bit to set bit 6 in the Status Byte Register and send an SRQ to the computer. If events are monitored by the Standard Event Status Register, also enable ESB with the "SRE" command.
     
2. When an interrupt occurs, interrupt handler should serial poll STB with:
   ```
   varR = myScope.IO.ReadSTB
   ```
   To read the Status Byte Register, use the following:
   ```
   myScope.WriteString "*STB?"
   varR = myScope.ReadNumber
   MsgBox "STB: 0x" + Hex(varR)
   ```
   This displays the hexadecimal value of the Status Byte Register.

3. Determine which bits in the Status Byte Register are set.

4. To read the Standard Event Status Register, use the following:
   ```
   myScope.WriteString "*ESR?"
   varR = myScope.ReadNumber
   MsgBox "ESR: 0x" + Hex(varR)
   ```

5. To read the Standard Event Status Register, use the following:
   ```
   myScope.WriteString "*OPC?"
   varR = myScope.ReadNumber
   MsgBox "OPC: 0x" + Hex(varR)
   ```

6. Use the following to see if an operation is complete:
   ```
   myScope.WriteString "*OPC?"
   varR = myScope.ReadNumber
   MsgBox "OPC: 0x" + Hex(varR)
   ```

7. Use the following to read the contents of the status byte:
   ```
   myScope.WriteString "*STB?"
   varR = myScope.ReadNumber
   MsgBox "STB: 0x" + Hex(varR)
   ```

END
Synchronizing Acquisitions

Synchronization in the Programming Flow  778
Blocking Synchronization  779
Polling Synchronization With Timeout  780
Synchronizing with a Single-Shot Device Under Test (DUT)  782
Synchronization with an Averaging Acquisition  784

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.
Synchronizing Acquisitions

Synchronization in the Programming Flow

Most remote programming follows these three general steps:

1. Set up the oscilloscope and device under test (see page 778).
2. Acquire a waveform (see page 778).
3. Retrieve results (see page 778).

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.

<table>
<thead>
<tr>
<th></th>
<th>Blocking Wait</th>
<th>Polling Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use When</td>
<td>You know the oscilloscope will trigger based on the oscilloscope setup and device under test.</td>
<td>You know the oscilloscope may or may not trigger on the oscilloscope setup and device under test.</td>
</tr>
<tr>
<td>Advantages</td>
<td>No need for polling. Fastest method.</td>
<td>Remote interface will not timeout. No need for device clear if no trigger.</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>Remote interface may timeout. Device clear only way to get control of oscilloscope if there is no trigger.</td>
<td>Slower method. Requires polling loop. Requires known maximum wait time.</td>
</tr>
<tr>
<td>Implementation Details</td>
<td>See &quot;Blocking Synchronization&quot; on page 779.</td>
<td>See &quot;Polling Synchronization With Timeout&quot; on page 780.</td>
</tr>
</tbody>
</table>

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 779 and "Polling Synchronization With Timeout" on page 780 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.

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 780 with the addition of checking for the armed event status.

```vbs
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 "*: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 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
10  Synchronizing Acquisitions
11
More About Oscilloscope Commands

Command Classifications  788
Valid Command/Query Strings  789
Query Return Values  810
All Oscilloscope Commands Are Sequential  811
Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of Agilent InfiniiVision oscilloscopes, commands are classified by the following categories:

- "Core Commands" on page 788
- "Non-Core Commands" on page 788
- "Obsolete Commands" on page 788

Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Agilent InfiniiVision oscilloscopes. Core commands are unlikely to be modified in the future. If you restrict your programs to core commands, the programs should work across product offerings in the future, assuming appropriate programming methods are employed.

Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all Agilent InfiniiVision oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Agilent's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

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 7, “Obsolete and Discontinued Commands,” starting on page 689
- As well as: Chapter 6, “Commands A-Z,” starting on page 655
Valid Command/Query Strings

- "Program Message Syntax" on page 789
- "Command Tree" on page 793
- "Duplicate Mnemonics" on page 807
- "Tree Traversal Rules and Multiple Commands" on page 808

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 790), 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.
The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument. The "Command Tree" on page 793 illustrates how all the mnemonics can be joined together to form a complete header.

":DISPlay:LABel ON" is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, ".:DISPlay:LABel?". Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:
- "Simple Command Headers" on page 791
- "Compound Command Headers" on page 791
- "Common Command Headers" on page 792

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 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 792 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.

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.

<table>
<thead>
<tr>
<th>Long Form</th>
<th>Short form</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANGe</td>
<td>RANG</td>
</tr>
<tr>
<td>PATTern</td>
<td>PATT</td>
</tr>
<tr>
<td>TIMebase</td>
<td>TIM</td>
</tr>
<tr>
<td>DELay</td>
<td>DEL</td>
</tr>
<tr>
<td>TYPE</td>
<td>TYPE</td>
</tr>
</tbody>
</table>

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.

Command Tree

The command tree shows all of the commands and the relationships of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree. When a program message terminator (<NL>, linefeed-ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the root of the command tree.

:(root)  
  :ACQuire (see page 181)  
  :AALias (see page 183)  
  :COMComplete (see page 184)  
  :COUNT (see page 185)  
  :DAALias (see page 186)  
  :MODE (see page 187)  
  :POINts (see page 188)  
  :RSIGnal (see page 189)  
  :SEGmented  
    :ANALyze (see page 190)  
    :COUNT (see page 191)  
    :INDex (see page 192)  
  :SRATe (see page 195)  
  :TYPE (see page 196)  
  :ACTivity (see page 142)  
  :AER (Arm Event Register) (see page 143)  
  :AUToscale (see page 144)  
    :AMODE (see page 146)  
    :CHANnels (see page 147)  
  :BLANk (see page 148)  
  :BUS<n> (see page 198)
More About Oscilloscope Commands

- :BIT<m> (see page 200)
- :BIT$ (see page 201)
- :CLEar (see page 203)
- :DISPly (see page 204)
- :LABel (see page 205)
- :MASK (see page 206)
- :CALibrate (see page 207)
- :DATE (see page 209)
- :LABel (see page 210)
- :OUTPut (see page 211)
- :STARt (see page 212)
- :STATus (see page 213)
- :SWITch (see page 214)
- :TEMPerature (see page 215)
- :TIME (see page 216)
- :CDISplay (see page 149)
- :CHANnel<n> (see page 217)
  - :BWLimit (see page 220)
  - :COUPling (see page 221)
  - :DISPly (see page 222)
  - :IMPedance (see page 223)
  - :INVert (see page 224)
  - :LABel (see page 225)
  - :OFFSet (see page 226)
  - :PROBe (see page 227)
    - :HEAD[:TYPE] (see page 228)
    - :ID (see page 229)
    - :SKew (see page 230)
    - :STYPe (see page 231)
    - :PROTection (see page 232)
  - :RANGe (see page 233)
  - :SCALe (see page 234)
  - :UNITs (see page 235)
  - :VERNier (see page 236)
- :DIGital<n> (see page 237)
- :DISPlay (see page 239)
- :LABel (see page 240)
- :POSition (see page 241)
- :SIZE (see page 242)
- :THReshold (see page 243)
- :DIGitize (see page 150)
- :DISPlay (see page 244)
  - :CLEar (see page 246)
  - :DATA (see page 247)
  - :LABel (see page 249)
  - :LABList (see page 250)
  - :PERSistence (see page 251)
- :SOURce (see page 252)
- :VECTors (see page 253)
- :EXTernal (see page 254)
  - :BWLimit (see page 256)
  - :IMPedance (see page 257)
  - :PROBe (see page 258)
    - :ID (see page 259)
    - :STYPe (see page 260)
  - :PROTection (see page 261)
  - :RANGe (see page 262)
  - :UNITs (see page 263)
- :FUNCtion (see page 264)
  - :CENTer (see page 267)
  - :DISPlay (see page 268)
  - :GOFT
    - :OPERation (see page 269)
    - :SOURce1 (see page 270)
    - :SOURce2 (see page 271)
  - :OFFSet (see page 272)
  - :OPERation (see page 273)
  - :RANGe (see page 274)
  - :REFerence (see page 275)
  - :SCALe (see page 276)
More About Oscilloscope Commands

- :SOURce1 (see page 277)
- :SOURce2 (see page 278)
- :SPAN (see page 279)
- :WINDow (see page 280)
- :HARDcopy (see page 281)
- :AREA (see page 283)
- :APRinter (see page 284)
- :FACTors (see page 285)
- :FFEed (see page 286)
- :INKSaver (see page 287)
- :LAYout (see page 288)
- :PALette (see page 289)
- [:PRINter]
  - :LIST (see page 290)
  - [:START] (see page 291)
- :HWEenable (Hardware Event Enable Register) (see page 152)
- :HWERegister
  - :CONDition (Hardware Event Condition Register) (see page 154)
  - [:EVENt] (Hardware Event Event Register) (see page 156)
- :LISTer (see page 292)
- :DATA (see page 293)
- :DISPlay (see page 294)
- :MARKer (see page 295)
  - :MODE (see page 297)
  - :X1Position (see page 298)
  - :X1Y1source (see page 299)
  - :X2Position (see page 300)
  - :X2Y2source (see page 301)
  - :XDELta (see page 302)
  - :Y1Position (see page 303)
  - :Y2Position (see page 304)
  - :YDELta (see page 305)
- :MEASure (see page 306)
  - :CLEar (see page 314)
  - :COUNter (see page 315)
• :DEFine (see page 316)
• :DELAY (see page 319)
• :DUTY (see page 321)
• :FALLtime (see page 322)
• :FREQuency (see page 323)
• :NWIDTH (see page 324)
• :OVERshoot (see page 325)
• :PERiod (see page 327)
• :PHASE (see page 328)
• :PRESHoot (see page 329)
• :PWIDTH (see page 330)
• :RISetime (see page 334)
• :RESULTS (see page 331)
• :SDEViation (see page 335)
• :SHOW (see page 336)
• :SOURce (see page 337)
• :STATistics (see page 339)
  • :INCRement (see page 340)
  • :RESET (see page 341)
• :TEDGe (see page 342)
• :TVALue (see page 344)
• :VAMplitude (see page 346)
• :VAverage (see page 347)
• :VBASE (see page 348)
• :VMAX (see page 349)
• :VMIN (see page 350)
• :VPP (see page 351)
• :VRATio (see page 352)
• :VRMS (see page 353)
• :VTIme (see page 354)
• :VTOP (see page 355)
• :WINDow (see page 356)
• :XMAX (see page 357)
• :XMIN (see page 358)
• :MERGe (see page 158)
11 More About Oscilloscope Commands

- :MTEnable (Mask Test Event Enable Register) (see page 159)
- :MTERegister[:EVENt] (Mask Test Event Event Register) (see page 161)
- :MTESt (see page 359)
  - :AMASK
    - :CREate (see page 364)
    - :SOURCe (see page 365)
    - :UNITs (see page 366)
    - :XDELta (see page 367)
    - :YDELta (see page 368)
  - :COUNt
    - :FWAVeforms (see page 369)
    - :RESet (see page 370)
    - :TIME (see page 371)
    - :WAVeforms (see page 372)
  - :DATA (see page 373)
  - :DELlete (see page 374)
  - :ENABle (see page 375)
  - :LOCK (see page 376)
  - :OUTPut (see page 377)
  - :RMODe (see page 378)
    - :FACTion
      - :MEASure (see page 379)
      - :PRINt (see page 380)
      - :SAVE (see page 381)
      - :STOP (see page 382)
  - :SIGMa (see page 383)
  - :TIME (see page 384)
  - :WAVeforms (see page 385)
  - :SCALe
    - :BIND (see page 386)
    - :X1 (see page 387)
    - :XDELta (see page 388)
    - :Y1 (see page 389)
    - :Y2 (see page 390)
  - :SOURce (see page 391)
• :TITLE (see page 392)
• :OPEE (Operation Status Enable Register) (see page 163)
• :OPERegister
  • :CONDition (Operation Status Condition Register) (see page 165)
  • [:EVENt] (Operation Status Event Register) (see page 167)
• :OVLenable (Overload Event Enable Register) (see page 169)
• :OVLRegister (Overload Event Register) (see page 171)
• :POD<n> (see page 393)
  • :DISPlay (see page 394)
  • :SIZE (see page 395)
  • :THReshold (see page 396)
• :RECall
  • :FIlename (see page 399)
  • :IMAGe (see page 400)
    • [:STARt] (see page 400)
  • :MASK (see page 401)
    • [:STARt] (see page 401)
  • :PWD (see page 402)
  • :SETup (see page 403)
    • [:STARt] (see page 403)
• :RUN (see page 174)
• :SAVE
  • :FIlename (see page 406)
  • :IMAGe (see page 407)
    • [:STARt] (see page 407)
  • :AREA (see page 408)
  • :FACTors (see page 409)
  • :FORMat (see page 410)
  • :IGColors (see page 411)
  • :PALette (see page 412)
  • :LISTer (see page 413)
    • [:STARt] (see page 413)
  • :MASK (see page 414)
    • [:STARt] (see page 414)
  • :PWD (see page 415)
11 More About Oscilloscope Commands

- :SETup (see page 416)
  - [:START] (see page 416)
- :WAVeform (see page 417)
  - [:START] (see page 417)
  - :FORMat (see page 418)
  - :LENGth (see page 419)
  - :SEGMented (see page 420)
- :SBUS (see page 421)
- :CAN
  - :COUNt
    - :ERRor (see page 423)
    - :OVERload (see page 424)
    - :RESet (see page 425)
    - :TOTal (see page 426)
    - :UTILization (see page 427)
- :DISPlay (see page 428)
- :FLEXray
  - :COUNt
    - :NULL? (see page 429)
    - :RESet (see page 430)
    - :SYNC? (see page 431)
    - :TOTal? (see page 432)
- :I2S
  - :BASE (see page 433)
- :IIC
  - :ASIZE (see page 434)
- :LIN
  - :PARity (see page 435)
- :M1553
  - :BASE (see page 436)
- :MODE (see page 437)
- :SPI
  - :BITorder (see page 438)
  - :WIDTh (see page 439)
- :UART
• :BASE (see page 440)
• :COUNt
  • :ERRor (see page 441)
  • :RESet (see page 442)
  • :RXFRames (see page 443)
  • :TXFRames (see page 444)
• :FRAMing (see page 445)
• :SERial (see page 175)
• :SINGle (see page 176)
• :STATus (see page 177)
• :STOP (see page 178)
• :SYSTem (see page 446)
  • :DATE (see page 447)
  • :DSP (see page 448)
  • :ERRor (see page 449)
  • :LOCK (see page 450)
  • :PRECision (see page 451)
• :PROTection
  • :LOCK (see page 434)
• :SETup (see page 453)
• :TIME (see page 455)
• :TER (Trigger Event Register) (see page 179)
• :TIMebase (see page 456)
  • :MODE (see page 458)
  • :POSition (see page 459)
  • :RANGe (see page 460)
  • :REFClock (see page 461)
  • :REFerence (see page 462)
  • :SCALE (see page 463)
• :VERNier (see page 464)
• :WINDow
  • :POSition (see page 465)
  • :RANGe (see page 466)
  • :SCALE (see page 467)
• :TRIGger (see page 468)
More About Oscilloscope Commands

- :HFReject (see page 472)
- :HOLDoff (see page 473)
- :LFIFor (see page 474)
- :MODE (see page 475)
- :NREJec (see page 476)
- :PATTern (see page 477)
- :SWEep (see page 479)
- :CAN (see page 480)
  - :ACKNowledge (see page 740)
  - :PATTern
    - :DATA (see page 482)
    - :LENGth (see page 483)
    - :ID (see page 484)
    - :MODE (see page 485)
  - :SAMPlenpoint (see page 486)
  - :SIGNal
    - :BAUDrate (see page 487)
    - :DEFinition (see page 488)
  - :SOURce (see page 489)
  - :TRIGger (see page 490)
  - :DURation (see page 492)
  - :GREaterthan (see page 493)
  - :LESSthan (see page 494)
  - :PATTern (see page 495)
  - :QUALifier (see page 496)
  - :RANGe (see page 497)
  - :EBURst (see page 498)
  - :COUNt (see page 499)
  - :IDLE (see page 500)
  - :SLOPe (see page 501)
  - [:EDGE] (see page 502)
  - :COUPling (see page 503)
  - :LEVel (see page 504)
  - :REJect (see page 505)
  - :SLOPe (see page 506)
- :SOURce (see page 507)
- :FLEXray (see page 508)
- :AUToset (see page 509)
- :BAUDrate (see page 510)
- :CHANnel (see page 511)
- :ERRor
  - :TYPE (see page 512)
- :EVENt
  - :TYPE (see page 513)
- :FRAMe
  - :CCBase (see page 514)
  - :CCRepetition (see page 515)
  - :ID (see page 516)
  - :TYPE (see page 517)
- :SOURce (see page 518)
- :TRIGger (see page 519)
- :GLITch (see page 520)
- :GREaterthan (see page 522)
- :LESSthan (see page 523)
- :LEVel (see page 524)
- :POLarity (see page 525)
- :QUALifier (see page 526)
- :RANGe (see page 527)
- :SOURce (see page 528)
- :HFReject (see page 472)
- :HOLDoff (see page 473)
- :I2S (see page 529)
  - :ALIGNment (see page 531)
  - :AUDio (see page 532)
  - :CLOCk
    - :SLOPe (see page 533)
  - :PATTern
    - :DATA (see page 534)
    - :FORMat (see page 536)
  - :RANGe (see page 537)
More About Oscilloscope Commands

- :RWDth (see page 539)
- :SOURce
  - :CLOCk (see page 540)
  - :DATA (see page 541)
  - :WSELect (see page 542)
- :TRIGger (see page 543)
- :TWIDth (see page 545)
- :WSLow (see page 546)
- :IIC (see page 547)
- :PATTern
  - :ADDReSS (see page 548)
  - :DATA (see page 549)
  - :DATa2 (see page 550)
- :SOURce
  - :CLOCk (see page 551)
  - :DATA (see page 552)
- :TRIGger
  - :QUALifier (see page 553)
  - [:TYPE] (see page 554)
- :LIN (see page 556)
- :ID (see page 558)
- :PATTern
  - :DATA (see page 559)
    - :LENGTH (see page 561)
    - :FORMAT (see page 562)
  - :SAMPlepoint (see page 563)
- :SIGNal
  - :BAUDrate (see page 564)
  - :DEFinition (see page 741)
- :SOURce (see page 565)
- :STANdard (see page 566)
- :SYNCbreak (see page 567)
- :TRIGger (see page 568)
- :M1553 (see page 569)
- :AUTosetup (see page 570)
• :PATTern
  • :DATA (see page 571)
• :RTA (see page 572)
• :SOURce
  • :LOWer (see page 573)
  • :UPPer (see page 574)
• :TYPE (see page 575)
• :MODE (see page 475)
• :NREJect (see page 476)
• :PATTern (see page 477)
• :SEQUence (see page 576)
  • :COUNt (see page 577)
• :EDGE (see page 578)
• :FIND (see page 579)
• :PATTern (see page 580)
• :RESet (see page 581)
• :TIMer (see page 582)
• :TRIGger (see page 583)
• :SPI (see page 584)
  • :CLOCk
    • :SLOPe (see page 585)
    • :TIMEout (see page 586)
  • :FRAMing (see page 587)
• :PATTern
  • :DATA (see page 588)
  • :WIDTh (see page 589)
• :SOURce
  • :CLOCk (see page 590)
  • :DATA (see page 591)
  • :FRAME (see page 592)
• :SWEep (see page 479)
• :TV (see page 593)
  • :LINE (see page 594)
• :MODE (see page 595)
• :POLarity (see page 596)
More About Oscilloscope Commands

- :SOURce (see page 597)
- :STANdard (see page 598)
- :TVMode (see page 743)
- :UART (see page 599)
- :BASE (see page 601)
- :BAUDrate (see page 602)
- :BITorder (see page 603)
- :BURSt (see page 604)
- :DATA (see page 605)
- :IDLE (see page 606)
- :PARity (see page 607)
- :QUALifier (see page 609)
- :POLarity (see page 608)
- :SOURCE
  - :RX (see page 610)
  - :TX (see page 611)
  - :TYPE (see page 612)
  - :WIDTH (see page 613)
- :USB (see page 614)
- :SOURCE
  - :DMINus (see page 615)
  - :DPLus (see page 616)
  - :SPEed (see page 617)
  - :TRIGger (see page 618)
- :VIEW (see page 180)
- :WAVEform (see page 619)
  - :BYTeorder (see page 627)
  - :COUNt (see page 628)
  - :DATA (see page 629)
  - :FORMat (see page 631)
  - :POINts (see page 632)
  - :MODE (see page 634)
  - :PREamble (see page 636)
  - :SEGmented
    - :COUNt (see page 639)
More About Oscilloscope Commands

- :TTAG (see page 640)
- :SOURce (see page 641)
  - :SUBSource (see page 645)
- :TYPE (see page 646)
- :UNSigned (see page 647)
- :VIEW (see page 648)
- :XINCcrement (see page 649)
- :XORigin (see page 650)
- :XREFerence (see page 651)
- :YINCrement (see page 652)
- :YORigin (see page 653)
- :YREFerence (see page 654)

Common Commands (IEEE 488.2)

- *CLS (see page 117)
- *ESE (see page 118)
- *ESR (see page 120)
- *IDN (see page 122)
- *LRN (see page 123)
- *OPC (see page 124)
- *OPT (see page 125)
- *RCL (see page 127)
- *RST (see page 128)
- *SAV (see page 131)
- *SRE (see page 132)
- *STB (see page 134)
- *TRG (see page 136)
- *TST (see page 137)
- *WAI (see page 138)

Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGe may be used to change the vertical range or to change the horizontal range:

:CHANnel1:RANGe .4

Sets the vertical range of channel 1 to 0.4 volts full scale.

:TIMebase:RANGe 1
Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMebase are subsystem selectors and determine which range is being modified.

**Tree Traversal Rules and Multiple Commands**

Command headers are created by traversing down the Command Tree (see page 793). A legal command header would be :TIMebase:RANGe. This is referred to as a *compound header*. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces.

The following rules apply to traversing the tree:

- A leading colon (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command lets you access that subsystem until a leading colon or a program message terminator (<NL>) or EOI true is found.

- In the command tree, use the last mnemonic in the compound header as the reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMebase:). That is the point where the parser resides. Any command below that point can be sent within the current program message without sending the mnemonics which appear above them (for example, POSition).

The output statements in the examples are written using the Agilent VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

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

For example:

```
myScope.WriteString " :TIMebase:RANGe 0.5; POSition 0"
```

**NOTE**

The colon between TIMebase and RANGe is necessary because TIMebase:RANGe is a compound command. The semicolon between the RANGe command and the POSition command is the required program message unit separator. The POSition command does not need TIMebase preceding it because the TIMebase:RANGe command sets the parser to the TIMebase node in the tree.
Example 2: Program Message Terminator Sets Parser Back to Root

myScope.WriteString "::TIMEbase::REFerence CENTer;POSition 0.00001"

or

myScope.WriteString "::TIMEbase::REFerence CENTer"
myScope.WriteString "::TIMEbase::POSition 0.00001"

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.

NOTE

A second way to send these commands is by placing TIMebase: before the POSition command as shown in the second part of example 2. The space after POSition is required.

Example 3: Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

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

For example:

myScope.WriteString "::TIMEbase::REFerence CENTer;::DISPLAY:VECTors ON"

NOTE

The leading colon before DISPLAY:VECTors ON tells the parser to go back to the root of the command tree. The parser can then see the DISPLAY:VECTors ON command. The space between REFerence and CENTer is required; so is the space between VECTors and ON.

Multiple commands may be any combination of compound and simple commands.
Query Return Values

Command headers immediately followed by a question mark (?) are queries. Queries are used to get results of measurements made by the instrument or to find out how the instrument is currently configured.

After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued.

When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMebase:RANGe? places the current time base setting in the output queue. When using the Agilent VISA COM library in Visual Basic, the controller statements:

```vbnet
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.
11 More About Oscilloscope Commands
12

Programming Examples

VISA COM Examples  814
VISA Examples  847
SICL Examples  893

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 814
- "VISA COM Example in C#" on page 824
- "VISA COM Example in Visual Basic .NET" on page 836

VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Reference the Agilent VISA COM library:
   a. Choose Tools>References... from the main menu.
   b. In the References dialog, check the "VISA COM 3.0 Type Library".
   c. Click OK.
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.

```vbnet
Option Explicit

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

' MAIN PROGRAM
' -------------------------------------------------------------------
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
' The commands sent to the oscilloscope are written in both long and
' short form. Both forms are acceptable.
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
```

```
' Agilent VISA COM Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates most of the commonly used programming
' features of your Agilent oscilloscopes.
' -------------------------------------------------------------------
```
If you are using a different signal or different channels, these commands may not work as explained in the comments.

Sub Main()

On Error GoTo VisaComError

' Create the VISA COM I/O resource.
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

' GPIB.
Set myScope.IO = myMgr.Open("GPIB0::7::INSTR")

' LAN.
Set myScope.IO = myMgr.Open("TCPIP0::a-mso6102-90541::inst0::INSTR")

' USB.
Set myScope.IO = myMgr.Open("USB0::2391::5970::30D3090541::0::INSTR")

' Initialize - Initialization will start the program with the oscilloscope in a known state.
Initialize

' Capture - After initialization, you must make waveform data available to analyze. To do this, capture the data using the DIGITIZE command.
Capture

' Analyze - Once the waveform has been captured, it can be analyzed. There are many parts of a waveform to analyze. This example shows some of the possible ways to analyze various parts of a waveform.
Analyze

Exit Sub

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

End Sub

Initialize

Initialize will start the program with the oscilloscope in a known state. This is required because some uninitialized conditions could cause the program to fail or not perform as expected.

In this example, we initialize the following:
- Oscilloscope
- Channel 1 range
- Display Grid
- Timebase reference, range, and delay
- Trigger mode and type
Private Sub Initialize()

    On Error GoTo VisaComError

    ' Clear the interface.
    myScope.IO.Clear

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

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

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

    ' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
    ' range value is 8 times the volts per division.
    ' Set the vertical range to 8 volts.
    myScope.WriteString ":CHANNEL1:RANGE 8"

    ' TIME_RANGE - Sets the full scale horizontal time in seconds. The
    ' range value is 10 times the time per division.
    ' Set the time range to 0.002 seconds.
    myScope.WriteString ":TIM:RANG 2e-3"

    ' TIME_REFERENCE - Possible values are LEFT and CENTER.
    ' - LEFT sets the display reference on time division from the left.
    ' - CENTER sets the display reference to the center of the screen.
    ' Set reference to center.
    myScope.WriteString ":TIMEBASE:REFERENCE CENTER"

    ' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
    ' TV trigger. Any channel can be selected.
    myScope.WriteString ":TRIGGER:TV:SOURCE CHANNEL1"

    ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
    ' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
    ' Set the trigger mode to EDGE.
    myScope.WriteString ":TRIGGER:MODE EDGE"
' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.
' Set the slope to positive.
myScope.WriteString "*:TRIGGER:EDGE:SLOPE POSITIVE"

' The following commands are not executed and are shown for reference purposes only. To execute these commands, uncomment them.
' RUN_STOP - (not executed in this example)
' - RUN starts the acquisition of data for the active waveform display.
' - STOP stops the data acquisition and turns off AUTOSTORE.
' myScope.WriteString ":RUN" ' Start data acquisition.
' myScope.WriteString ":STOP" ' Stop the data acquisition.

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

' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.
' Set time base mode to main.
' myScope.WriteString "*:TIMEBASE:MODE MAIN"

Exit Sub

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

End Sub

' Capture
' -------------------------------------------------------------------
' We will capture the waveform using the digitize command.
' -------------------------------------------------------------------

Private Sub Capture()

On Error GoTo VisaComError

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

' ACQUIRE_COMPLETE - Specifies the minimum completion criteria for
an acquisition. The parameter determines the percentage of time
buckets needed to be "full" before an acquisition is considered
to be complete.
myScope.WriteString "*:ACQUIRE:COMPLETE 100"

' DIGITIZE - Used to acquire the waveform data for transfer over
the interface. Sending this command causes an acquisition to
take place with the resulting data being placed in the buffer.
NOTE! The DIGITIZE command is highly recommended for triggering modes other than SINGLE. This ensures that sufficient data is available for measurement. If DIGITIZE is used with single mode, the completion criteria may never be met. The number of points gathered in Single mode is related to the sweep speed, memory depth, and maximum sample rate. For example, take an oscilloscope with a 1000-point memory, a sweep speed of 10 us/div (100 us total time across the screen), and a 20 MSa/s maximum sample rate. 1000 divided by 100 us equals 10 MSa/s. Because this number is less than or equal to the maximum sample rate, the full 1000 points will be digitized in a single acquisition. Now, use 1 us/div (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s; because this is greater than the maximum sample rate by 5 times, only 400 points (or 1/5 the points) can be gathered on a single trigger. Keep in mind when the oscilloscope is running, communication with the computer interrupts data acquisition. Setting up the oscilloscope over the bus causes the data buffers to be cleared and internal hardware to be reconfigured. If a measurement is immediately requested, there may have not been enough time for the data acquisition process to collect data, and the results may not be accurate. An error value of 9.9E+37 may be returned over the bus in this situation.

myScope.WriteString ":DIGITIZE CHAN1"

Exit Sub

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

Private Sub Analyze()
    On Error GoTo VisaComError
    SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message that contains the current state of the instrument. Its format is a definite-length binary block, for example, #80002204<setup string><NL>
    where the setup string is 2204 bytes in length.
    myScope.WriteString ":SYSTEM:SETUP?"
    varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
    CheckForInstrumentErrors  ' After reading query results.
    Output setup string to a file:
Dim strPath As String
strPath = "c:scope\config\setup.dat"
Close #1 ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult ' Write data.
Close #1 ' Close file.

' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString "DISPLAY:DATA? BMP, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
    Kill strPath
End If
Close #1 ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , byteData ' Write data.
Close #1 ' Close file.
myScope.IO.Timeout = 5000

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write
' it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:scope\config\setup.dat"
Open strPath For Binary Access Read As #1 ' Open file for input.
Get #1, , varSetupString ' Read data.
Close #1 ' Close file.
' Write setup string back to oscilloscope using "SYSTEM:SETUP"
' command:
myScope.WriteIEEEBlock "SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure.
myScope.WriteString "MEASURE:SOURCE CHANNEL1"

' Query for frequency.
myScope.WriteString "MEASURE:FREQUENCY?"
varQueryResult = myScope.ReadNumber ' Read frequency.
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(varQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
myScope.WriteString "MEASURE:DUTYCYCLE?"
varQueryResult = myScope.ReadNumber ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + _
    FormatNumber(varQueryResult, 3) + "%"
' Query for risetime.
myScope.WriteString "':MEASURE:RISETIME?"
varQueryResult = myScope.ReadNumber ' Read risetime.
MsgBox "Risetime:" + vbCrLf + _
    FormatNumber(varQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
myScope.WriteString "':MEASURE:VPP?"
varQueryResult = myScope.ReadNumber ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' Query for Vmax.
myScope.WriteString "':MEASURE:VMAX?"
varQueryResult = myScope.ReadNumber ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' "':WAVEFORM:DATA?" query. Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString "':WAVEFORM:SOURCE CHAN1"

' WAVE_POINTS - Specifies the number of points to be transferred
' using the "':WAVEFORM:DATA?" query.
myScope.WriteString "':WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output. This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString "':WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' With WORD format, use most significant byte first order.
myScope.WriteString "':WAVEform:BYTorder MSBFirst"

' Data in range 0 to 255.
'myScope.WriteString "':WAVEFORM:FORMAT BYTE"
lngVSteps = 256
intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
' FORMAT : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
' TYPE : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
' POINTS : int32 - number of data points transferred.
' COUNT : int32 - 1 and is always 1.
' XINCREMENT : float64 - time difference between data points.
' XORIGIN : float64 - always the first data point in memory.
' XREFERENCE : int32 - specifies the data point associated with x-origin.
' YINCREMENT : float32 - voltage difference between data points.
' YORIGIN : float32 - value is the voltage at center screen.
' YREFERENCE : int32 - specifies the data point where y-origin occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":\WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + FormatNumber(dblXIncrement * 1000000) + _
" us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
" FormatNumber(dblXOrigin * 1000000) + _
" us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
" CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
" FormatNumber(sngYIncrement * 1000) + _
" mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
" FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
" CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
" FormatNumber(lngVSteps * sngYIncrement / 8) + _
" V" + vbCrLf
strOutput = strOutput + "Offset = " + _
FormatNumber(sngYOrigin) * V* + vbCrLf
strOutput = strOutput + "Sec/Div = " + __
FormatNumber(lngPoints * dblXIncrement / 10 * __
1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + __
FormatNumber(((lngPoints / 2) * __
dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":\WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
' <header><waveform_data><NL>
'
' Where:
' <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
For lngI = 0 To UBound(varQueryResult) __
    Step (UBound(varQueryResult) / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 + __
        varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
strOutput = strOutput + "Data point " + __
    CStr(lngI / intBytesPerData) + *, " + __
    FormatNumber((lngDataValue - lngYReference) * sngYIncrement + __
    sngYOrigin) + " V", " + __
    FormatNumber(((lngI / intBytesPerData - lngXReference) * __
    dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double
' Query time at 1st rising edge on ch1.
myScope.WriteString "":MEASURE:TEDGE? +1, CHAN1"

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString "":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.

' Query time at 1st rising edge on ch1.
myScope.WriteString "":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

Exit Sub

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

Private Sub CheckForInstrumentErrors()
    On Error GoTo VisaComError
    Dim strErrVal As String
    Dim strOut As String
    myScope.WriteString "SYSTEM:ERROR?"  ' Query any errors data.
    strErrVal = myScope.ReadString  ' Read: Errnum,"Error String".
    While Val(strErrVal) <> 0  ' End if find: 0,"No Error".
        strOut = strOut + "INST Error: " + strErrVal
        myScope.WriteString "":SYSTEM:ERROR?"  ' Request error message.
        strErrVal = myScope.ReadString  ' Read error message.
    Wend
    If Not strOut = "" Then
        MsgBox strOut, vbExclamation, "INST Error Messages"
    End If
    Exit Sub
End Sub
myScope.FlushWrite (False)
myScope.FlushRead

End If

Exit Sub

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

End Sub

**VISA COM Example in C#**

To compile and run this example in Microsoft Visual Studio 2005:

1  Open Visual Studio.
2  Create a new Visual C#, Windows, Console Application project.
3  Cut-and-paste the code that follows into the C# source file.
4  Edit the program to use the VISA address of your oscilloscope.
5  Add a reference to the VISA COM 3.0 Type Library:
   a  Right-click the project you wish to modify (not the solution) in the
      Solution Explorer window of the Microsoft Visual Studio
      environment.
   b  Choose **Add Reference....**
   c  In the Add Reference dialog, select the **COM** tab.
   d  Select **VISA COM 3.0 Type Library**; then click **OK**.
6  Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

/*
 * Agilent VISA COM Example in C#
 * -----------------------------------------------------------
 * This program illustrates most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -----------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;
namespace InfiniiVision
{
   class VisaComInstrumentApp
   {

   
}
private static VisaComInstrument myScope;

public static void Main(string[] args)
{
    try
    {
        myScope = new VisaComInstrument("USB0::2391::5957::MY47250010::0::INSTR");

        Initialize();

        /* The extras function contains miscellaneous commands that
         * do not need to be executed for the proper operation of
         * this example. The commands in the extras function are
         * shown for reference purposes only.
         */
        // Extra();  // Uncomment to execute the extra function.
        Capture();
        Analyze();
    }
    catch (System.ApplicationException err)
    {
        Console.WriteLine("*** VISA Error Message : " + err.Message);
    }
    catch (System.SystemException err)
    {
        Console.WriteLine("*** System Error Message : " + err.Message);
    }
    catch (System.Exception err)
    {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
    }
    finally
    {
        myScope.Close();
    }
}

/*
 * Initialize()
 * --------------------------------------------------------------
 * This function initializes both the interface and the
 * oscilloscope to a known state.
 */
private static void Initialize()
{
    string strResults;

    /* RESET - This command puts the oscilloscope into a known
     * state. This statement is very important for programs to
     * work as expected. Most of the following initialization
     * commands are initialized by *RST. It is not necessary to
     * reinitialize them unless the default setting is not suitable
     * for your application.
     */
    myScope.DoCommand("*RST");  // Reset the to the defaults.
myScope.DoCommand("*CLS"); // Clear the status data structures.

/* IDN - Ask for the device's *IDN string. */
strResults = myScope.DoQueryString("*IDN?");

// Display results.
Console.Write("Result is: {0}", strResults);

/* AUTOSCALE - This command evaluates all the input signals
* and sets the correct conditions to display all of the
* active signals. */
myScope.DoCommand(":AUToscale");

/* CHANNEL_PROBE - Sets the probe attenuation factor for the
* selected channel. The probe attenuation factor may be from
* 0.1 to 1000. */
myScope.DoCommand(":CHANnel1:PROBe 10");

/* CHANNEL_RANGE - Sets the full scale vertical range in volts.
* The range value is eight times the volts per division. */
myScope.DoCommand(":CHANnel1:RANGe 8");

/* TIME_RANGE - Sets the full scale horizontal time in seconds.
* The range value is ten times the time per division. */
myScope.DoCommand(":TIMebase:RANGe 2e-3");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
* - LEFT sets the display reference one time division from
*   the left.
* - CENTER sets the display reference to the center of the
*   screen. */
myScope.DoCommand(":TIMebase:REFerence CENTer");

/* TRIGGER_SOURCE - Selects the channel that actually produces
* the TV trigger. Any channel can be selected. */
myScope.DoCommand(":TRIGger:TV:SOURCe CHANnel1");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
* PATTERN, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
* UART, or USB. */
myScope.DoCommand(":TRIGger:MODE EDGE");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
* trigger to either POSITIVE or NEGATIVE. */
myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");

/*
private static void Extra()
{
    myScope.DoCommand(":RUN");
    myScope.DoCommand(":STOP");

    myScope.DoCommand(":BLANk CHANnel1");
    myScope.DoCommand(":VIEW CHANnel1");

    myScope.DoCommand(":TIMebase:MODE MAIN");
}

private static void Capture()
{
    myScope.DoCommand(":ACQuire:TYPE NORMal");

    myScope.DoCommand(":ACQuire:COMPlete 100");

    myScope.DoCommand(":DIGITIZE");
}
NOTE! The use of the DIGITIZE command is highly recommended as it will ensure that sufficient data is available for measurement. Keep in mind when the oscilloscope is running, communication with the computer interrupts data acquisition. Setting up the oscilloscope over the bus causes the data buffers to be cleared and internal hardware to be reconfigured.

If a measurement is immediately requested there may not have been enough time for the data acquisition process to collect data and the results may not be accurate. An error value of 9.9E+37 may be returned over the bus in this situation.

myScope.DoCommand(":DIGitize CHANnell");

private static void Analyze()
{
    byte[] ResultsArray; // Results array.
    int nBytes; // Number of bytes returned from instrument.

    /* SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a program message that contains the current state of the instrument. Its format is a definite-length binary block, for example,
       #800002204<setup string><NL>
       where the setup string is 2204 bytes in length.
    */
    Console.WriteLine("Saving oscilloscope setup to " + "c:\scope\config\setup.dat");
    if (File.Exists("c:\scope\config\setup.dat"))
        File.Delete("c:\scope\config\setup.dat");
    ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
    nBytes = ResultsArray.Length;
    Console.WriteLine("Read oscilloscope setup ({0} bytes).", nBytes);
    // Write setup string to file.
    File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray);
    Console.WriteLine("Wrote setup string ({0} bytes) to file.", nBytes);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string to the oscilloscope. */
byte[] DataArray;

    // Read setup string from file.
    DataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
    Console.WriteLine("Read setup string ({0} bytes) from file.",
        DataArray.Length);

    // Restore setup string.
    myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
    Console.WriteLine("Restored setup string.");

    /* IMAGE_TRANSFER - In this example, we query for the screen
     * data with the ":DISPLAY:DATA?" query. The .png format
     * data is saved to a file in the local file system.
     */
    Console.WriteLine("Transferring screen image to " +
        "c:\\scope\\data\\screen.png");
    if (File.Exists("c:\\scope\\data\\screen.png")
        File.Delete("c:\\scope\\data\\screen.png");

    // Increase I/O timeout to fifteen seconds.
    myScope.SetTimeoutSeconds(15);

    // Get the screen data in PNG format.
    ResultsArray = myScope.DoQueryIEEEBlock(
        "*:DISPLAY:DATA? PNG, SCReen, COlor*");
    nBytes = ResultsArray.Length;
    Console.WriteLine("Read screen image ({0} bytes).", nBytes);

    // Store the screen data in a file.
    File.WriteAllBytes("c:\\scope\\data\\screen.png",
        ResultsArray);
    Console.WriteLine("Wrote screen image ({0} bytes) to file.",
        nBytes);

    // Return I/O timeout to five seconds.
    myScope.SetTimeoutSeconds(5);

    /* MEASURE - The commands in the MEASURE subsystem are used to
     * make measurements on displayed waveforms.
     */

    // Set source to measure.
    myScope.DoCommand(":MEASure:SOURce CHANnel1");

    // Query for frequency.
    double fResults;
    fResults = myScope.DoQueryValue(":MEASure:FREQuency?");
    Console.WriteLine("The frequency is: {0:F4} kHz",
        fResults / 1000);

    // Query for peak to peak voltage.
    fResults = myScope.DoQueryValue(":MEASure:VPP?");
    Console.WriteLine("The peak to peak voltage is: (0:F2) V",
        fResults);

    /* WAVEFORM_DATA - Get waveform data from oscilloscope. To
* obtain waveform data, you must specify the WAVEFORM parameters for the waveform data prior to sending the "WAVEFORM:DATA?" query.

* Once these parameters have been sent, the "WAVEFORM:PREAMBLE?" query provides information concerning the vertical and horizontal scaling of the waveform data.

* With the preamble information you can then use the "WAVEFORM:DATA?" query and read the data block in the correct format.

*/

/* WAVE_FORMAT - Sets the data transmission mode for waveform data output. This command controls how the data is formatted when sent from the oscilloscope and can be set to WORD or BYTE format. */

// Set waveform format to BYTE.
myScope.DoCommand("WAVEform:FORMat BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred. The number of time points available is returned by the "ACQUIRE:POINTS?" query. This can be set to any binary fraction of the total time points available. */

myScope.DoCommand("WAVEform:POINts 1000");

/* GET_PREAMBLE - The preamble contains all of the current WAVEFORM settings returned in the form <preamble block><NL> where the <preamble block> is:
* FORMAT : int16 - 0 = BYTE, 1 = WORD, 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 the x-origin.
* YINCREMENT : float32 - voltage difference between data points.
* YORIGIN : float32 - value of the voltage at center screen.
* YREFERENCE : int32 - data point where y-origin occurs. */

Console.WriteLine("Reading preamble.");
double[] fResultsArray;
fResultsArray = myScope.DoQueryValues("WAVEform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMAT: (0:e), {0:e}", fFormat);

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: (0:e)", fType);

double fPoints = fResultsArray[2];
Console.WriteLine("Preamble POINTs: (0:e)", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNT: (0:e)", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: (0:e)", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: (0:e)", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Preamble XREFerence: (0:e)", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: (0:e)", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: (0:e)", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Preamble YREFerence: (0:e)", fYreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVeform:SOURce" command. */

/* READ_WAVE_DATA - The wave data consists of two parts: the
 * header, and the actual waveform data followed by a
 * New Line (NL) character. The query data has the following
 * format:
 * 
 * <header><waveform data block><NL>
 * 
 * Where:
 * 
 * <header> = #800002048 (this is an example header)
 * 
 * The ":8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the
 * ":WAVEFORM:POINTS" command. You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query. */

// Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock("*:WAVeform:DATA?;

nBytes = ResultsArray.Length;
Console.WriteLine("Read waveform data ((0) bytes).", nBytes);

// Make some calculations from the preamble data.
double fVdiv = 32 * fYincrement;
double fOffset = fYorigin;
double fSdiv = fPoints * fXincrement / 10;
double fDelay = (fPoints / 2) * fXincrement + fXorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fVdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < nBytes; i = i + (nBytes / 20))
{
    Console.WriteLine("Data point {0:d} = {1:f6} Volts at " + /*(float)ResultsArray[i] - fYreference) * fYincrement + fYorigin," + ((float)i - fXreference) * fXincrement + fXorigin);
}

/*--------------------------------------------------------*/
/* SAVE_WAVE_DATA - saves the waveform data to a CSV format */
/* file named "waveform.csv". */
/*--------------------------------------------------------*/
if (File.Exists("c:\scope\data\waveform.csv"))
    File.Delete("c:\scope\data\waveform.csv");
StreamWriter writer =
    File.CreateText("c:\scope\data\waveform.csv");
for (int i = 0; i < nBytes; i++)
{
    writer.WriteLine("{0:E}, {1:f6}",
        ((float)i - fXreference) * fXincrement + fXorigin,
        ((float)ResultsArray[i] - fYreference) * fYincrement + fYorigin);
}
wrtier.Close();
Console.WriteLine("Waveform data ({0} points) written to " + /*c:\scope\data\waveform.csv.", nBytes);
}
}

class VisaComInstrument
{
    private ResourceManagerClass m_ResourceManager;
    private FormattedIO488Class m_IoObject;
    private string m_strVisaAddress;

    // Constructor.
    public VisaComInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA COM IO object.
        OpenIO();
    

// Clear the interface.
m_IoObject.IO.Clear();

public void DoCommand(string strCommand)
{
    // Send the command.
    m_IoObject.WriteString(strCommand, true);

    // Check for instrument errors.
    CheckForInstrumentErrors(strCommand);
}

public string DoQueryString(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result string.
    string strResults;
    strResults = m_IoObject.ReadString();

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return results string.
    return strResults;
}

public double DoQueryValue(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result number.
    double fResult;
    fResult = (double)m_IoObject.ReadNumber(
        IEEEASCIIType.ASCIIType_R8, true);

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return result number.
    return fResult;
}

public double[] DoQueryValues(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result numbers.
    double[] fResultsArray;
    fResultsArray = (double[])m_IoObject.ReadList(
        IEEEASCIIType.ASCIIType_R8, "", ";");
// Check for instrument errors.
CheckForInstrumentErrors(strQuery);

// Return result numbers.
return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the results array.
    byte[] ResultsArray;
    ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
        IEEEBinaryType.BinaryType_UI1, false, true);

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return results array.
    return ResultsArray;
}

public void DoCommandIEEEBlock(string strCommand,
    byte[] DataArray)
{
    // Send the command.
    m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

    // Check for instrument errors.
    CheckForInstrumentErrors(strCommand);
}

private void CheckForInstrumentErrors(string strCommand)
{
    string strInstrumentError;
    bool bFirstError = true;

    // Repeat until all errors are displayed.
    do
    {
        // Send the ":SYSTem:ERRor?" query, and get the result string.
        m_IoObject.WriteString(":SYSTem:ERRor?", true);
        strInstrumentError = m_IoObject.ReadString();

        // If there is an error, print it.
        if (strInstrumentError.ToString() != "+0,"No error"
            {"n"
            }
            if (bFirstError)
            { // Print the command that caused the error.
                Console.WriteLine("ERROR(s) for command '{0'}: ",
                    strCommand);
                bFirstError = false;
            }
        Console.Write(strInstrumentError);
} while (strInstrumentError.ToString() != "+0,"No error\n"");

private void OpenIo()
{
    m_ResourceManager = new ResourceManagerClass();
    m_IoObject = new FormattedIO488Class();

    // Open the default VISA COM IO object.
    try
    {
        m_IoObject.IO =
            (IMessage)m_ResourceManager.Open(m_strVisaAddress,
                AccessMode.NO_LOCK, 0, "*");
    } catch (Exception e)
    {
        Console.WriteLine("An error occurred: {0}", e.Message);
    }
}

public void SetTimeoutSeconds(int nSeconds)
{
    m_IoObject.IO.Timeout = nSeconds * 1000;
}

public void Close()
{
    try
    {
        m_IoObject.IO.Close();
    } catch ()
    {
        try
        {
            Marshal.ReleaseComObject(m_IoObject);
        } catch ()
        {
            try
            {
                Marshal.ReleaseComObject(m_ResourceManager);
            } catch ()
            {
                Console.WriteLine("An error occurred: {0}", e.Message);
            }
        }
    }
}
VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2005:

2. Create a new Visual Basic, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA COM 3.0 Type Library:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose Add Reference....
   c. In the Add Reference dialog, select the COM tab.
   d. Select VISA COM 3.0 Type Library; then click OK.
   e. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose Properties; then, select "InfiniiVision.VisaComInstrumentApp" as the Startup object.

6. Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices
Namespace InfiniiVision
Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument
    Public Shared Sub Main(ByVal args As String())
        Try
            myScope = New _
            VisaComInstrument("USB0::2391::5957::MY47250010::0::INSTR")
            Initialize()

            ' The extras function contains miscellaneous commands that
' do not need to be executed for the proper operation of
' this example. The commands in the extras function are
' shown for reference purposes only.

' Extra(); // Uncomment to execute the extra function.
Capture()
Analyze()
Catch err As System.ApplicationException
Catch err As System.SystemException
Catch err As System.Exception
    System.Diagnostics.Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error:" + err.Message)
Finally
    myScope.Close()
End Try
End Sub

' Initialize()
' --------------------------------------------------------------
' This function initializes both the interface and the
' oscilloscope to a known state.

Private Shared Sub Initialize()
    Dim strResults As String

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

    ' Reset to the defaults.
    myScope.DoCommand("*RST")

    ' Clear the status data structures.
    myScope.DoCommand("*CLS")

    ' IDN - Ask for the device's *IDN string.
    strResults = myScope.DoQueryString("*IDN?")

    ' Display results.
    Console.Write("Result is: {0}", strResults)

    ' AUTOSCALE - This command evaluates all the input signals
    ' and sets the correct conditions to display all of the
    ' active signals.
    myScope.DoCommand(":AUToscale")

    ' CHANNEL_PROBE - Sets the probe attenuation factor for the
    ' selected channel. The probe attenuation factor may be from
    ' 0.1 to 1000.
    myScope.DoCommand(":CHANn1:PROBe 10")

    ' CHANNEL_RANGE - Sets the full scale vertical range in volts.
The range value is eight times the volts per division.
myScope.DoCommand(":*CHANnel1:RANGe 8")

TIME_RANGE - Sets the full scale horizontal time in seconds. 
The range value is ten times the time per division.
myScope.DoCommand(":*TIMebase:RANGe 2e-3")

TIME_REFERENCE - Possible values are LEFT and CENTER:
- LEFT sets the display reference one time division from
  the left.
- CENTER sets the display reference to the center of the
  screen.
myScope.DoCommand(":*TIMebase:REFerence CENTER")

TRIGGER_SOURCE - Selects the channel that actually produces
the TV trigger. Any channel can be selected.
myScope.DoCommand(":*TRIGger:TV:SOURCe CHANnel1")

TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
PATTern, CAN, DURection, IIC, LIN, SEQuence, SPI, TV,
UART, or USB.
myScope.DoCommand(":*TRIGger:MODE EDGE")

TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
trigger to either POSITIVE or NEGATIVE.
myScope.DoCommand(":*TRIGger:EDGE:SLOPe POSitive")

End Sub

Extra()

--------------------------------------------------------------

The commands in this function are not executed and are shown
for reference purposes only. To execute these commands, call
this function from main.

Private Shared Sub Extra()
  ' RUN_STOP (not executed in this example):
  ' - RUN starts the acquisition of data for the active
    waveform display.
  ' - STOP stops the data acquisition and turns off AUTOSTORE.
  '
  myScope.DoCommand(":RUN")
  myScope.DoCommand(":STOP")
  ' VIEW_BLANK (not executed in this example):
  ' - VIEW turns on (starts displaying) an active channel or
    pixel memory.
  ' - BLANK turns off (stops displaying) a specified channel or
    pixel memory.
  '
  myScope.DoCommand(":BLANk CHAnnell")
  myScope.DoCommand(":VIEW CHAnnell")
' TIME_MODE (not executed in this example) - Set the time base
' mode to MAIN, DELAYED, XY or ROLL.
'
myScope.DoCommand(":TIMebase:MODE MAIN")
End Sub

' Capture()
' ---------------------------------------------------------------------------
' This function prepares the scope for data acquisition and then
' uses the DIGITIZE MACRO to capture some data.

Private Shared Sub Capture()
' AQUIRE_TYPE - Sets the acquisition mode. There are three
' acquisition types NORMAL, PEAK, or AVERAGE.
myScope.DoCommand(":ACQuire:TYPE NORMal")

' 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.DoCommand(":ACQuire:COMPLETE 100")

' DIGITIZE - Used to acquire the waveform data for transfer
' over the interface. Sending this command causes an
' acquisition to take place with the resulting data being
' placed in the buffer.

' NOTE! The use of the DIGITIZE command is highly recommended
' as it will ensure that sufficient data is available for
' measurement. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data
' buffers to be cleared and internal hardware to be
' reconfigured.
' If a measurement is immediately requested there may not have
' been enough time for the data acquisition process to collect
' data and the results may not be accurate. An error value of
' 9.9E+37 may be returned over the bus in this situation.
myScope.DoCommand(":DIGitize CHANnell")

End Sub

' Analyze()
' ---------------------------------------------------------------------------
' In this example we will do the following:
' - Save the system setup to a file for restoration at a later
' time.
' - Save the oscilloscope display to a file which can be
' printed.
' - Make single channel measurements.

Private Shared Sub Analyze()
' Results array.
Dim ResultsArray As Byte()

' Number of bytes returned from instrument.
Dim nBytes As Integer

' SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a program message that contains the current state of the instrument. Its format is a definite-length binary block, for example,
' #800002204<setup string><NL>
' where the setup string is 2204 bytes in length.
Console.WriteLine("Saving oscilloscope setup to " + 
' "c:\scope\config\setup.dat")
If File.Exists("c:\scope\config\setup.dat") Then
    File.Delete("c:\scope\config\setup.dat")
End If

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nBytes = ResultsArray.Length
Console.WriteLine("Read oscilloscope setup ({0} bytes).", nBytes)

' Write setup string to file.
File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
Console.WriteLine("Wrote setup string ({0} bytes) to file.", nBytes)

' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string to the oscilloscope.
Dim DataArray As Byte()

' Read setup string from file.
DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
Console.WriteLine("Read setup string ({0} bytes) from file.", DataArray.Length)

' Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
Console.WriteLine("Restored setup string.")

' IMAGE_TRANSFER - In this example, we query for the screen data with the ":DISPlay:DATA?" query. The .png format data is saved to a file in the local file system.
Console.WriteLine("Transferring screen image to " + 
' "c:\scope\data\screen.png")
If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
End If

' Increase I/O timeout to fifteen seconds.
myScope.SetTimeoutSeconds(15)

' Get the screen data in PNG format.
ResultsArray = _
    myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, SCReen, COlor")
nBytes = ResultsArray.Length
Console.WriteLine("Read screen image ({0} bytes).", nBytes)

' Store the screen data in a file.
File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
Console.WriteLine("Wrote screen image ({0} bytes) to file.", _ nBytes)

' Return I/O timeout to five seconds.
myScope.SetTimeoutSeconds(5)

' MEASURE - The commands in the MEASURE subsystem are used to make measurements on displayed waveforms.

' Set source to measure.
myScope.DoCommand(":MEASure:SOURce CHANnel1")

' Query for frequency.
Dim fResults As Double
fResults = myScope.DoQueryValue(":MEASure:FREQuency?")
Console.WriteLine("The frequency is: (0:F4) kHz", _ fResults / 1000)

' Query for peak to peak voltage.
fResults = myScope.DoQueryValue(":MEASure:VPP?"")
Console.WriteLine("The peak to peak voltage is: (0:F2) V", _ fResults)

' WAVEFORM_DATA - Get waveform data from oscilloscope. To obtain waveform data, you must specify the WAVEFORM parameters for the waveform data prior to sending the "*:WAVEFORM:DATA?" query.

' Once these parameters have been sent, the "*:WAVEFORM:PREAMBLE?" query provides information concerning the vertical and horizontal scaling of the waveform data.

' With the preamble information you can then use the "*:WAVEFORM:DATA?" query and read the data block in the correct format.

' WAVE_FORMAT - Sets the data transmission mode for waveform data output. This command controls how the data is formatted when sent from the oscilloscope and can be set to WORD or BYTE format.

' Set waveform format to BYTE.
myScope.DoCommand("*:WAVeform:FORMat BYTE")

' WAVE_POINTS - Sets the number of points to be transferred.
' The number of time points available is returned by the "ACQUIRE:POINTS?" query. This can be set to any binary fraction of the total time points available.
myScope.DoCommand("*:WAVeform:POINts 1000")

' GET_PREAMBLE - The preamble contains all of the current WAVEFORM settings returned in the form <preamble block><NL>
' where the <preamble block> is:
' FORMAT : int16 - 0 = BYTE, 1 = WORD, 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 the x-origin.
YINCREMENT : float32 - voltage difference between data points.
YORIGIN : float32 - value of the voltage at center screen.
YREFERENCE : int32 - data point where y-origin occurs.

Console.WriteLine("Reading preamble.")
Dim fResultsArray As Double() = myScope.DoQueryValues(":\WAVeform:\PREamble?")

Dim fFormat As Double = fResultsArray(0)
Console.WriteLine("Preamble FORMAT: {0:e}", fFormat)

Dim fType As Double = fResultsArray(1)
Console.WriteLine("Preamble TYPE: {0:e}", fType)

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Preamble POINTs: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Preamble COUNt: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Preamble XREference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YREference: {0:e}", fYreference)

- QUERY\_WAVE\_DATA - Outputs waveform records to the controller over the interface that is stored in a buffer previously specified with the ":WAVEform:SOURce" command.
- READ\_WAVE\_DATA - The wave data consists of two parts: the header, and the actual waveform data followed by a New Line (NL) character. The query data has the following format:
  
  <header><waveform data block><NL>
Where:

<header> = #800002048 (this is an example header)

The "#8" may be stripped off of the header and the remaining numbers are the size, in bytes, of the waveform data block. The size can vary depending on the number of points acquired for the waveform which can be set using the ".:WAVEFORM:POINTS" command. You may then read that number of bytes from the oscilloscope; then, read the following NL character to terminate the query.

Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":.WAVEform:DATA?")
nBytes = ResultsArray.Length
Console.WriteLine("Read waveform data ({0} bytes).", nBytes)

Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = {0:f}", fOffset)
Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
Console.WriteLine("Delay = {0:e}", fDelay)

Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < nBytes
    Console.WriteLine("Data point {0:d} = {1:f6} Volts at " + _
        "(2:f10) Seconds", i, _
        (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
        fYorigin, (CSng(i) - fXreference) * fXincrement + fXorigin)
    i = i + (nBytes / 20)
End While

SAVE_WAVE_DATA - saves the waveform data to a CSV format file named "waveform.csv".
If File.Exists("c:\scope\data\waveform.csv") Then
    File.Delete("c:\scope\data\waveform.csv")
End If

Dim writer As StreamWriter = _
    File.CreateText("c:\scope\data\waveform.csv")
For index As Integer = 0 To nBytes - 1
    writer.WriteLine("{0:E}, {1:f6}", _
        (CSng(index) - fXreference) * fXincrement + fXorigin, _
        (CSng(ResultsArray(index)) - fYreference) * fYincrement + fYorigin)
Next
writer.Close()
Console.WriteLine("Waveform data ({0} points) written to " + _
Class VisaComInstrument
Private mResourceManager As ResourceManagerClass
Private m_IoObject As FormattedIO488Class
Private m_strVisaAddress As String

' Constructor.
Public Sub New(ByVal strVisaAddress As String)
' Save VISA address in member variable.
m_strVisaAddress = strVisaAddress

' Open the default VISA COM IO object.
OpenIo()

' Clear the interface.
m_IoObject.IO.Clear()
End Sub

Public Sub DoCommand(ByVal strCommand As String)
' Send the command.
m_IoObject.WriteString(strCommand, True)

' Check for instrument errors.
CheckForInstrumentErrors(strCommand)
End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
' Send the query.
m_IoObject.WriteString(strQuery, True)

' Get the result string.
Dim strResults As String
strResults = m_IoObject.ReadString()

' Check for instrument errors.
CheckForInstrumentErrors(strQuery)

' Return results string.
Return strResults
End Function

Public Function DoQueryValue(ByVal strQuery As String) As Double
' Send the query.
m_IoObject.WriteString(strQuery, True)

' Get the result number.
Dim fResult As Double
fResult = _
CDbl(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

' Check for instrument errors.
CheckForInstrumentErrors(strQuery)

' Return result number.
Return fResult
End Function

Public Function DoQueryValues(ByVal strQuery As String) As Double()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result numbers.
    Dim fResultsArray As Double()
    fResultsArray = _
        m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, "", ";")

    ' Check for instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return result numbers.
    Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String) As Byte()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the results array.
    Dim ResultsArray As Byte()
    ResultsArray = _
        m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
            False, True)

    ' Check for instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return results array.
    Return ResultsArray
End Function

Public Sub DoCommandIEEEBlock(ByVal strCommand As String, ByVal DataArray As Byte())
    ' Send the command.
    m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

    ' Check for instrument errors.
    CheckForInstrumentErrors(strCommand)
End Sub

Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
    Dim strInstrumentError As String
    Dim bFirstError As Boolean = True

    ' Repeat until all errors are displayed.
    Do
        ' Send the ":SYSTem:ERRor?" query, and get the result string.
        m_IoObject.WriteString("":SYSTem:ERRor?", True)
        strInstrumentError = m_IoObject.ReadString()
' If there is an error, print it.
If strInstrumentError.ToString() <> "+0,""No error"" _
& Chr(10) & "" Then
    If bFirstError Then
        ' Print the command that caused the error.
        Console.WriteLine("ERROR(s) for command '{0}': ", _
            strCommand)
        bFirstError = False
    End If
    console.WriteLine(strInstrumentError)
End If
Loop While strInstrumentError.ToString() <> "+0,""No error"" _
& Chr(10) & ""
End Sub

Private Sub OpenIo()
    m_ResourceManager = New ResourceManagerClass()
    m_IoObject = New FormattedIO488Class()

    ' Open the default VISA COM IO object.
    Try
        m_IoObject.IO = _
            DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
                AccessMode.NO_LOCK, 0, ""), IMessage)
    Catch e As Exception
        Console.WriteLine("An error occurred: {0}", e.Message)
    End Try
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
    m_IoObject.IO.Timeout = nSeconds * 1000
End Sub

Public Sub Close()
    Try
        m_IoObject.IO.Close()
    Catch
        End Try

    Try
        Marshal.ReleaseComObject(m_IoObject)
    Catch
        End Try

    Try
        Marshal.ReleaseComObject(m_ResourceManager)
    Catch
        End Try

End Class
End Namespace
VISA Examples

- "VISA Example in C" on page 847
- "VISA Example in Visual Basic" on page 856
- "VISA Example in C#" on page 866
- "VISA Example in Visual Basic .NET" on page 879

VISA Example in C

To compile and run this example in Microsoft Visual Studio 2005:

3. In the Win32 Application Wizard, click Next >. Then, check Empty project, and click Finish.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2005, right-click the Source Files folder, choose Add > Add Existing Item..., select the example.c file, and click Add.
6. Edit the program to use the VISA address of your oscilloscope.
7. Choose Project > Properties.... In the Property Pages dialog, update these project settings:
   c. Click OK to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose Tools > Options....
   b. In the Options dialog, select VC++ Directories under Projects and Solutions.
   c. Show directories for Include files, and add the include directory (for example, Program Files\VISA\winnt\include).
   d. Show directories for Library files, and add the library files directory (for example, Program Files\VISA\winnt\lib\msc).
   e. Click OK to close the Options dialog.
9. Build and run the program.

/*
 * Agilent VISA Example in C
 */

---

Programming Examples 12
This program illustrates most of the commonly-used programming features of your Agilent oscilloscope. This program is to be built as a WIN32 console application. Edit the RESOURCE line to specify the address of the applicable device.

#include <stdio.h> /* For printf(). */
#include <visa.h> /* Agilent VISA routines. */

/* GPIB */
/* #define RESOURCE "GPIB0::7::INSTR" */

/* LAN */
/* #define RESOURCE "TCPIP0::a-mso6102-90541::inst0::INSTR" */

/* USB */
define RESOURCE "USB0::2391::5970::30D3090541::0::INSTR"
#define WAVE_DATA_SIZE 5000
#define TIMEOUT 5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE 300000

/* Function prototypes */
void initialize(void); /* Initialize the oscilloscope. */
void extra(void); /* Miscellaneous commands not executed, shown for reference purposes. */
void capture(void); /* Digitize data from oscilloscope. */
void analyze(void); /* Make some measurements. */
void get_waveform(void); /* Download waveform data from oscilloscope. */
void save_waveform(void); /* Save waveform data to a file. */
void retrieve_waveform(void); /* Load waveform data from a file. */

/* Global variables */
ViSession defaultRM, vi; /* Device session ID. */
char buf[256] = {0}; /* Buffer for IDN string. */
unsigned char waveform_data[WAVE_DATA_SIZE]; /* Array for waveform data. */
double preamble[10]; /* Array for preamble. */

void main(void)
{
    /* Open session. */
    viOpenDefaultRM(&defaultRM);
    viOpen(defaultRM, RESOURCE, VI_NULL, VI_NULL, &vi);
    printf("Oscilloscope session initialized!\n");

    /* Clear the interface. */
    viClear(vi);
    initialize();

    /* The extras function contains miscellaneous commands that do not need to be executed for the proper operation of this example. */
* The commands in the extras function are shown for reference purposes only.
*/
/* extra(); */ /* <-- Uncomment to execute the extra function */
capture();
analyze();

/* Close session */
viClose(vi);
viClose(defaultRM);
printf ("Program execution is complete...\n");
}

/* initialize
* ------------------------------------------------------------------
* This function initializes both the interface and the oscilloscope
* to a known state.
*/
void initialize (void)
{
    /* RESET - This command puts the oscilloscope in a known state.
     * Without this command, the oscilloscope settings are unknown.
     * This command is very important for program control.
     * Many of the following initialization commands are initialized
     * by this command. It is not necessary to reinitialize them
     * unless you want to change the default setting.
     */
    viPrintf(vi, "*RST\n");

    /* Write the *IDN? string and send an EOI indicator, then read
    * the response into buf.
    */
    viQueryf(vi, "*IDN?\n", "%t", buf);
    printf("%s\n", buf);
    
    /* AUTOSCALE - This command evaluates all the input signals and
    * sets the correct conditions to display all of the active signals.
    */
    viPrintf(vi, ":\nAUTOSCALE\n");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
    * selected channel. The probe attenuation factor may be from
    * 0.1 to 1000.
    */
    viPrintf(vi, ":\nCHAN1:PROBE 10\n");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
    * The range value is eight times the volts per division.
    */
    viPrintf(vi, ":\nCHANNEL1:RANGE 8\n");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.
    */
* The range value is ten times the time per division.
*/
viPrintf(vi, "::RANG 2e-3\n\n");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
* - LEFT sets the display reference one time division from the
* left.
* - CENTER sets the display reference to the center of the screen.
*/
viPrintf(vi, "::REFERENCE CENTER\n\n");

/* TRIGGER_SOURCE - Selects the channel that actually produces the
* TV trigger. Any channel can be selected.
*/
viPrintf(vi, "::SOURCE CHANNEL1\n\n");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch, PATTern,
* CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
*/
viPrintf(vi, "::MODE EDGE\n\n");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
* to either POSITIVE or NEGATIVE.
*/
viPrintf(vi, "::EDGE:SLOPE POSITIVE\n\n");

}  
/* extra
* ------------------------------------------------------------------
* The commands in this function are not executed and are shown for
* reference purposes only. To execute these commands, call this
* function from main.
*/

void extra (void)
{

/* RUN_STOP (not executed in this example):
* - RUN starts the acquisition of data for the active waveform
* display.
* - STOP stops the data acquisition and turns off AUTOSTORE.
*/
viPrintf(vi, "::RUN\n\n");
viPrintf(vi, "::STOP\n\n");

/* VIEW_BLANK (not executed in this example):
* - VIEW turns on (starts displaying) an active channel or pixel
* memory.
* - BLANK turns off (stops displaying) a specified channel or
* pixel memory.
*/
viPrintf(vi, "::CHANNEL1\n\n");
viPrintf(vi, "::VIEW CHANNEL1\n\n");

/* TIME_MODE (not executed in this example) - Set the time base
* mode to MAIN, DELAYED, XY or ROLL.
*/
viPrintf(vi, ":TIMEBASE:MODE MAIN\n";)
}

/*
 * capture
 * --------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

capture (void)
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
    viPrintf(vi, ":ACQUIRE:TYPE NORMAL\n";)
    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition. The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
    viPrintf(vi, ":ACQUIRE:COMPLETE 100\n";)
    /* DIGITIZE - Used to acquire the waveform data for transfer over
     * the interface. Sending this command causes an acquisition to
     * take place with the resulting data being placed in the buffer.
     */
    viPrintf(vi, ":DIGITIZE CHAN1\n";)
}

/*
 * analyze
 * --------------------------------------------------------------
 * In this example we will do the following:
 * - Save the system setup to a file for restoration at a later time.
 * - Save the oscilloscope display to a file which can be printed.
 * - Make single channel measurements.
 */

analyze (void)
{
    double frequency, vpp; /* Measurements. */
    double vdiv, off, sdiv, delay; /* Values calculated from preamble
                                     * data. */
    }
int i; /* Loop counter. */
unsigned char setup_string[SETUP_STR_SIZE]; /* Array for setup string. */

int setup_size;
FILE *fp;
unsigned char image_data[IMG_SIZE]; /* Array for image data. */
int img_size;

/* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program * message that contains the current state of the instrument. Its * format is a definite-length binary block, for example, * #800002204<setup string><NL> * where the setup string is 2204 bytes in length. */
setup_size = SETUP_STR_SIZE;
/* Query and read setup string. */
viQueryf(vi, " :SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
printf("Read setup string query (%d bytes).\n", setup_size);
/* Write setup string to file. */
fp = fopen ("c:\scope\config\setup.dat", "wb");
setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size, fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to file.\n", setup_size);

/* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string * to the oscilloscope. */
/* Read setup string from file. */
fp = fopen ("c:\scope\config\setup.dat", "rb");
setup_size = fread (setup_string, sizeof(unsigned char), SETUP_STR_SIZE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file.\n", setup_size);
/* Restore setup string. */
viPrintf(vi, " :SYSTEM:SETUP #8%08d", setup_size);
viBufWrite(vi, setup_string, setup_size, &setup_size);
viPrintf(vi, "\n");
printf("Restored setup string (%d bytes).\n", setup_size);

/* IMAGE_TRANSFER - In this example we will query for the image * data with ":DISPLAY:DATA?" to read the data and save the data * to the file "image.dat" which you can then send to a printer. */
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 30000);
printf("Transferring image to c:\scope\data\screen.bmp\n");
img_size = IMG_SIZE;
viQueryf(vi, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n", &img_size, image_data);
printf("Read display data query (%d bytes).\n", img_size);
/* Write image data to file. */
fp = fopen ("c:\scope\data\screen.bmp", "wb");
img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
fclose (fp);
printf("Wrote image data (%d bytes) to file.\n", img_size);
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 5000);
/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

/* Set source to measure. */
viPrintf(vi, "MEASURE:SOURCE CHANNEL1\n");

/* Query for frequency. */
viQueryf(vi, "MEASURE:FREQUENCY?\n", "%lf", &frequency);
printf("The frequency is: %.4f kHz\n", frequency / 1000);

/* Query for peak to peak voltage. */
viQueryf(vi, "MEASURE:VPP\n", "%lf", &vpp);
printf("The peak to peak voltage is: %.2f V\n", vpp);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. */
get_waveform();

/* Make some calculations from the preamble data. */
vdiv = 32 * preamble[7];
off = preamble[8];

/* Print them out... */
printf("Scope Settings for Channel 1:\n");
printf("Volts per Division = %f\n", vdiv);
printf("Offset = %f\n", off);
printf("Seconds per Division = %f\n", sdiv);
printf("Delay = %f\n", delay);

/* print out the waveform voltage at selected points */
for (i = 0; i < 1000; i = i + 50)
  printf("Data Point %d = %6.2f Volts at %10f Seconds\n", i,
    ((float)waveform_data[i] - preamble[9]) * preamble[7] +
    preamble[8],
    ((float)i - preamble[6]) * preamble[4] + preamble[5]);

save_waveform(); /* Save waveform data to disk. */
retrive_waveform(); /* Load waveform data from disk. */
}

/* get_waveform
 * ------------------------------------------------------------------
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
  int waveform_size;

  /* WAVEFORM_DATA - To obtain waveform data, you must specify the
   * WAVEFORM parameters for the waveform data prior to sending the
   * "*WAVEFORM:DATA?" query. */
Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?" query provides information concerning the vertical and horizontal scaling of the waveform data.

With the preamble information you can then use the ":WAVEFORM:DATA?" query and read the data block in the correct format.

/*
 * WAVE_FORMAT - Sets the data transmission mode for waveform data output. This command controls how the data is formatted when sent from the oscilloscope and can be set to WORD or BYTE format.
 */

viPrintf(vi, ":WAVEFORM:FORMAT BYTE\n");

/* WAVE_POINTS - Sets the number of points to be transferred. The number of time points available is returned by the "*ACQUIRE:POINTS?" query. This can be set to any binary fraction of the total time points available. */

viPrintf(vi, ":WAVEFORM:POINTS 1000\n");

/* GET_PREAMBLE - The preamble contains all of the current WAVEFORM settings returned in the form <preamble block><NL> where the <preamble block> is:
 * FORMAT : int16 - 0 = BYTE, 1 = WORD, 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 the x-origin.
 * YINCREMENT : float32 - voltage difference between data points.
 * YORIGIN : float32 - value of the voltage at center screen.
 * YREFERENCE : int32 - data point where y-origin occurs.
 */

printf("Reading preamble\n"); viQueryf(vi, ":WAVEFORM:PREAMBLE?\n", "%10lf\n", preamble);

/*
 * QUERY_WAVE_DATA - Outputs waveform records to the controller over the interface that is stored in a buffer previously
 */
viPrintf(vi, ":^WAVEFORM:DATA?\n"); /* Query waveform data. */

READ_WAVE_DATA - The wave data consists of two parts: the header, and the actual waveform data followed by an New Line (NL) character. The query data has the following format:

- <header><waveform data block><NL>

Where:

- <header> = #800002048 (this is an example header)
- The "#8" may be stripped off of the header and the remaining numbers are the size, in bytes, of the waveform data block.
- The size can vary depending on the number of points acquired for the waveform which can be set using the ":^WAVEFORM:POINTS" command. You may then read that number of bytes from the oscilloscope; then, read the following NL character to terminate the query.

waveform_size = WAVE_DATA_SIZE;

READ waveform data. */
viScanf(vi, "%#b\n", &waveform_size, waveform_data);
if ( waveform_size == WAVE_DATA_SIZE )
{
    printf("Waveform data buffer full: ");
    printf("May not have received all points.\n");
}
else
{
    printf("Reading waveform data... size = %d\n", waveform_size);
}

save_waveform

This function saves the waveform data from the get_waveform function to disk. The data is saved to a file called "wave.dat".

save_waveform()
{
    FILE *fp;
    fp = fopen("c:\\scope\\data\\wave.dat", "wb");
    /* Write preamble. */
    fwrite(preamble, sizeof(preamble[0]), 10, fp);
    /* Write actually waveform data. */
    fwrite(waveform_data, sizeof(waveform_data[0]), (int)preamble[2], fp);
    fclose(fp);
}

*/
* retrieve_waveform
* ------------------------------------------------------------------
* This function retrieves previously saved waveform data from a
* file called "wave.dat".
* /

void retrieve_waveform(void)
{
    FILE *fp;

    fp = fopen("c:\scope\data\wave.dat", "rb");
    /* Read preamble. */
    fread(preamble, sizeof(preamble[0]), 10, fp);
    /* Read the waveform data. */
    fread(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],
         fp);
    fclose(fp);
}

VISA Example in Visual Basic

To run this example in Visual Basic for Applications:

1 Start the application that provides Visual Basic for Applications (for
   example, Microsoft Excel).

2 Press ALT+F11 to launch the Visual Basic editor.

3 Add the visa32.bas file to your project:
   a Choose File>Import File....
   b Navigate to the header file, visa32.bas (installed with Agilent IO
      Libraries Suite and found in the Program Files\VISA\winnt\include
directory), select it, and click Open.

4 Choose Insert>Module.

5 Cut-and-paste the code that follows into the editor.

6 Edit the program to use the VISA address of your oscilloscope, and save
   the changes.

7 Run the program.

   ' Agilent VISA Example in Visual Basic
   ' -------------------------------------------------------------------
   ' This program illustrates a few commonly-used programming
   ' features of your Agilent oscilloscope.
   ' -------------------------------------------------------------------

Option Explicit

Public err As Long   ' Error returned by VISA function calls.
Public drm As Long   ' Session to Default Resource Manager.
Public vi As Long    ' Session to instrument.
' Declare variables to hold numeric values returned by viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long
Public Const DblArraySize = 20
Public dblArray(DblArraySize) As Double

' Declare fixed length string variable to hold string value returned by viVScanf/viVQueryf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

' Main Program
' -------------------------------------------------------------------
Sub Main()

' Open the default resource manager session.
err = viOpenDefaultRM(drm)
If (err <> VI_SUCCESS) Then HandleVISAError drm

' Open the session using the oscilloscope's VISA address.
err = viOpen(drm, _
        "USB0::2391::5970::30D3090541::0::INSTR", 0, 15000, vi)
If (err <> VI_SUCCESS) Then HandleVISAError drm

' 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()

' Set probe attenuation factor (from 0.1 to 1000).
' -------------------------------------------------------------------
_DoCommand ":CHANnel1:PROBe 10"
_Debug.Print "Channel 1 probe attenuation factor: " + _
_DoQueryString("":CHANnel1:PROBe?"")

' Use auto-scale to automatically configure oscilloscope.
' -------------------------------------------------------------------
_DoCommand ":AUToscale"

' Set the trigger mode to EDGE.
_DoCommand ":TRIGger:MODE EDGE"
_Debug.Print "Trigger mode: " + _
_DoQueryString("":TRIGger:MODE?"")

' Set EDGE trigger parameters.
_DoCommand ":TRIGger:EDGE:SOURCe CHANnell"
_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 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 to NORMal.
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 data using :DIGitize.
' -----------------------------------------------------------------
DoCommand " :DIGitize"

End Sub
'Analyze the captured waveform.
'------------------------------------------------------------------
Private Sub Analyze()

' Make a couple of measurements.
'------------------------------------------------------------------
  DoCommand "*:MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString("*:MEASure:SOURce?"
  DoCommand "*:MEASure:VAMPlitude"
  dblQueryResult = DoQueryNumber("*:MEASure:VAMPlitude?"
  MsgBox "Vertical amplitude: " + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"
  DoCommand "*:MEASure:FREQuency"
  dblQueryResult = DoQueryNumber("*:MEASure:FREQuency?"
  MsgBox "Frequency: " + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

'Download the screen image.
'------------------------------------------------------------------
' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = _
      DoQueryIEEEBlock_Bytes("*:DISPlay:DATA? PNG, SCREEN, COLOR"
  Debug.Print "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?"
'Set the desired number of waveform points.
  DoCommand "*:WAVeform:POINts 1000"
Debug.Print "Waveform points desired: " + _
    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 singYIncrement As Single
Dim singYOrigin As Single
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)
singYIncrement = dblArray(7)
singYOrigin = 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 = 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"
End If

Debug.Print "Waveform points desired: " + _
FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
Format(sngYOrigin, "Scientific")

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))
    Print #hFile, _
    Format((lngI - lngXReference) * dblXIncrement + _
    dblXOrigin, "Scientific") + "," + _
    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."

End Sub

Private Sub DoCommand(command As String)
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 + vbCrLf, 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))

retCount = DblArraySize

' Read numbers.
err = viVScanf(vi, "%lf" + vbCrLf, 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 + vbCrLf, 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))

' Get unsigned integer bytes.
Sleep 2000 ' Delay before reading data.
err = viVScanf(vi, "%#b" + vbCrLf, 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 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 If

End Sub
VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2005:

2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Agilent's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Click Add and then click Add Existing Item...
   c. Navigate to the header file, visa32.cs (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, but do not click the Open button.
   d. Click the down arrow to the right of the Add button, and choose Add as Link.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```csharp
/*
 * Agilent VISA Example in C#
 * --------------------------------------------------------------------------------
 * This program illustrates most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * --------------------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;

namespace InfiniiVision
{
    class VisaInstrumentApp
    {
        private static VisaInstrument oscp;

        public static void Main(string[] args)
```
try
{
    oscp = new VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR");
    Initialize();

    /* The extras function contains miscellaneous commands that
    * do not need to be executed for the proper operation of
    * this example. The commands in the extras function are
    * shown for reference purposes only.
    */
    // Extra(); // Uncomment to execute the extra function.
    Capture();
    Analyze();
}
catch (System.ApplicationException err)
{
    Console.WriteLine("*** VISA Error Message : " + err.Message);
}
catch (System.SystemException err)
{
    Console.WriteLine("*** System Error Message : " + err.Message);
}
catch (System.Exception err)
{
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
}
finally
{
    oscp.Close();
}

/*
 * Initialize()
 * --------------------------------------------------------------
 * This function initializes both the interface and the
 * oscilloscope to a known state.
 */
private static void Initialize()
{
    StringBuilder strResults;

    /* RESET - This command puts the oscilloscope into a known
    * state. This statement is very important for programs to
    * work as expected. Most of the following initialization
    * commands are initialized by *RST. It is not necessary to
    * reinitialize them unless the default setting is not suitable
    * for your application.
    */
    oscp.DoCommand("*RST"); // Reset the to the defaults.
    oscp.DoCommand("*CLS"); // Clear the status data structures.

    /* IDN - Ask for the device's *IDN string.

    */
*/
strResults = oscp.DoQueryString("*IDN?");

// Display results.
Console.WriteLine("Result is: {0}", strResults);

/* AUTOSCALE - This command evaluates all the input signals
 * and sets the correct conditions to display all of the
 * active signals.
 */
oscp.DoCommand(":*AUToscale");

/* CHANNEL_PROBE - Sets the probe attenuation factor for the
 * selected channel. The probe attenuation factor may be from
 * 0.1 to 1000.
 */
oscp.DoCommand(":*CHAnnel1:PROBe 10");

/* CHANNEL_RANGE - Sets the full scale vertical range in volts.
 * The range value is eight times the volts per division.
 */
oscp.DoCommand(":*CHAnnel1:RANGE 8");

/* TIME_RANGE - Sets the full scale horizontal time in seconds.
 * The range value is ten times the time per division.
 */
oscp.DoCommand(":*TIMebase:RANGE 2e-3");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
 * - LEFT sets the display reference one time division from
 *   the left.
 * - CENTER sets the display reference to the center of the
 *   screen.
 */
oscp.DoCommand(":*TIMebase:REFerence CENTer");

/* TRIGGER_SOURCE - Selects the channel that actually produces
 * the TV trigger. Any channel can be selected.
 */
oscp.DoCommand(":*TRIGger:TV:SOURCe CHANnell");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
 * PATTERN, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
 * UART, or USB.
 */
oscp.DoCommand(":*TRIGger:MODE EDGE");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
 * trigger to either POSITIVE or NEGATIVE.
 */
oscp.DoCommand(":*TRIGger:EDGE:SLOPe POSitive");

}
private static void Extra()
{
    /* RUN_STOP (not executed in this example): */
    /* - RUN starts the acquisition of data for the active */
    /* waveform display. */
    /* - STOP stops the data acquisition and turns off AUTOSTORE. */
    oscp.DoCommand(":RUN");
    oscp.DoCommand(":STOP");

    /* VIEW_BLANK (not executed in this example): */
    /* - VIEW turns on (starts displaying) an active channel or */
    /* pixel memory. */
    /* - BLANK turns off (stops displaying) a specified channel or */
    /* pixel memory. */
    oscp.DoCommand(":BLANK CHANnel1");
    oscp.DoCommand(":VIEW CHANnel1");

    /* TIME_MODE (not executed in this example) - Set the time base */
    /* mode to MAIN, DELAYED, XY or ROLL. */
    oscp.DoCommand(":TIMebase:MODE MAIN");
}

private static void Capture()
{
    /* AQUIRE_TYPE - Sets the acquisition mode. There are three */
    /* acquisition types NORMAL, PEAK, or AVERAGE. */
    oscp.DoCommand(":ACQuire:TYPE NORMal");

    /* 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. */
    oscp.DoCommand(":ACQuire:COMPlete 100");

    /* DIGITIZE - Used to acquire the waveform data for transfer */
    /* over the interface. Sending this command causes an */
    /* acquisition to take place with the resulting data being */
    /* placed in the buffer. */
    /* */

    /* NOTE! The use of the DIGITIZE command is highly recommended */
    /* as it will ensure that sufficient data is available for */
    /* measurement. Keep in mind when the oscilloscope is running,
* communication with the computer interrupts data acquisition.  
* Setting up the oscilloscope over the bus causes the data  
* buffers to be cleared and internal hardware to be  
* reconfigured.  
* If a measurement is immediately requested there may not have  
* been enough time for the data acquisition process to collect  
* data and the results may not be accurate. An error value of  
* 9.9E+37 may be returned over the bus in this situation. */

oscp.DoCommand(":DIGitize CHANnel1");

/*
* Analyze()
* --------------------------------------------------------------
* In this example we will do the following:
* - Save the system setup to a file for restoration at a later  
*   time.
* - Save the oscilloscope display to a file which can be  
*   printed.
* - Make single channel measurements.
*/
private static void Analyze()
{
    byte[] ResultsArray; // Results array.  
    int nLength; // Number of bytes returned from instrument.

    /* SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a  
     * program message that contains the current state of the  
     * instrument. Its format is a definite-length binary block,  
     * for example,  
     * #800002204<setup string><NL>  
     * where the setup string is 2204 bytes in length. */
    Console.WriteLine("Saving oscilloscope setup to " +  
                    "c:\\scope\\config\\setup.dat");
    if (File.Exists("c:\\scope\\config\\setup.dat"))
        File.Delete("c:\\scope\\config\\setup.dat");

    // Query and read setup string.
    nLength = oscp.DoQueryIEEEBlock(":SYSTem:SETup?",  
                                      out ResultsArray);
    Console.WriteLine("Read oscilloscope setup ({0} bytes).",  
                      nLength);

    // Write setup string to file.
    File.WriteAllBytes("c:\\scope\\config\\setup.dat",  
                       ResultsArray);
    Console.WriteLine("Wrote setup string ({0} bytes) to file.",  
                      nLength);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup  
     * string to the oscilloscope. */
    byte[] DataArray;
    int nBytesWritten;
// Read setup string from file.
DataArray = File.ReadAllBytes("c:\scope\config\setup.dat");
Console.WriteLine("Read setup string ({0} bytes) from file.",
    DataArray.Length);

// Restore setup string.
nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTem:SETup",
    DataArray);
Console.WriteLine("Restored setup string ({0} bytes).",
    nBytesWritten);

/* IMAGE_TRANSFER - In this example, we query for the screen
 * data with the ":DISPLAY:DATA?" query. The .png format
 * data is saved to a file in the local file system.
 */
Console.WriteLine("Transferring screen image to " +
    "c:\\scope\\data\\screen.png");
if (File.Exists("c:\\scope\\data\\screen.png"))
    File.Delete("c:\\scope\\data\\screen.png");

// Increase I/O timeout to fifteen seconds.
oscp.SetTimeoutSeconds(15);

// Get the screen data in PNG format.
nLength = oscp.DoQueryIEEEBlock(
    ":DISPLAY:DATA? PNG, SCReen, COlor", out ResultsArray);
Console.WriteLine("Read screen image ({0} bytes).", nLength);

// Store the screen data in a file.
File.WriteAllBytes("c:\\scope\\data\\screen.png",
    ResultsArray);
Console.WriteLine("Wrote screen image ({0} bytes) to file.",
    nLength);

// Return I/O timeout to five seconds.
oscp.SetTimeoutSeconds(5);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

// Set source to measure.
oscp.DoCommand(":MEASure:SOURce CHANnel1");

// Query for frequency.
double fResults;
fResults = oscp.DoQueryValue(":MEASure:FREQuency?");
Console.WriteLine("The frequency is: {0:F4} kHz",
    fResults / 1000);

// Query for peak to peak voltage.
fResults = oscp.DoQueryValue(":MEASure:VPP?");
Console.WriteLine("The peak to peak voltage is: {0:F2} V",
    fResults);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. To
 * obtain waveform data, you must specify the WAVEFORM

* parameters for the waveform data prior to sending the
  * ":WAVEFORM:DATA?" query.

* Once these parameters have been sent, the
  * ":WAVEFORM:PREAMBLE?" query provides information concerning
  * the vertical and horizontal scaling of the waveform data.

* With the preamble information you can then use the
  * ":WAVEFORM:DATA?" query and read the data block in the
  * correct format.
*/

/* WAVE_FORMAT - Sets the data transmission mode for waveform
* data output. This command controls how the data is
* formatted when sent from the oscilloscope and can be set
* to WORD or BYTE format.
*/
oscp.DoCommand(":WAVeform:FORMat BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred.
* The number of time points available is returned by the
* "ACQUIRE:POINTS?" query. This can be set to any binary
* fraction of the total time points available.
*/
oscp.DoCommand(":WAVeform:POINts 1000");

/* GET_PREAMBLE - The preamble contains all of the current
* WAVEFORM settings returned in the form <preamble block><NL>
* where the <preamble block> is:
* FORMAT : int16 - 0 = BYTE, 1 = WORD, 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 the x-origin.
* YINCREMENT : float32 - voltage difference between data
*               points.
* YORIGIN : float32 - value of the voltage at center
*            screen.
* YREFERENCE : int32 - data point where y-origin occurs.
*/
Console.WriteLine("Reading preamble.");

double[] fResultsArray;
fResultsArray = oscp.DoQueryValues(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMAT: (0:e), fFormat");

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: (0:e), fType");
double fPoints = fResultsArray[2];
Console.WriteLine("Preamble POINTs: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNT: {0:e}", fCount);

double fXIncrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: {0:e}", fXIncrement);

double fXOrigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: {0:e}", fXOrigin);

double fXReference = fResultsArray[6];
Console.WriteLine("Preamble XREFerence: {0:e}", fXReference);

double fYIncrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: {0:e}", fYIncrement);

double fYOrigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: {0:e}", fYOrigin);

double fYReference = fResultsArray[9];
Console.WriteLine("Preamble YREFerence: {0:e}", fYReference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVeform:SOURce" command.
 */

/* READ_WAVE_DATA - The wave data consists of two parts: the
 * header, and the actual waveform data followed by a
 * New Line (NL) character. The query data has the following
 * format:
 *   <header><waveform data block><NL>
 * Where:
 *   <header> = #800002048 (this is an example header)
 *   The "#8" may be stripped off of the header and the remaining
 *   numbers are the size, in bytes, of the waveform data block.
 *   The size can vary depending on the number of points acquired
 *   for the waveform which can be set using the
 *   ":WAVEFORM:POINTS" command. You may then read that number
 *   of bytes from the oscilloscope; then, read the following NL
 *   character to terminate the query.
 */

// Read waveform data.
nLength = oscp.DoQueryIEEEBlock("::WAVeform:DATA?", out ResultsArray);
Console.WriteLine("Read waveform data ({0} bytes).", nLength);

// Make some calculations from the preamble data.
double fYDiv = 32 * fYIncrement;
double fOffset = fYorigin;
double fSdiv = fPoints * fXincrement / 10;
double fDelay = (fPoints / 2) * fXincrement + fXorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fVdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < 1000; i = i + 50)
    Console.WriteLine("Data point {0:d} = {1:f2} Volts at " + "{2:f10} Seconds", i,
        ((float)ResultsArray[i] - fYreference) * fYincrement + fYorigin,
        ((float)i - fXreference) * fXincrement + fXorigin);

/* SAVE_WAVE_DATA - saves the waveform data to a CSV format
* file named "waveform.csv". */
if (File.Exists("c:\scope\data\waveform.csv"))
    File.Delete("c:\scope\data\waveform.csv");
StreamWriter writer =
    File.CreateText("c:\scope\data\waveform.csv");
for (int i = 0; i < 1000; i++)
    writer.WriteLine("{0:E}, {1:f6}",
        ((float)i - fXreference) * fXincrement + fXorigin,
        ((float)ResultsArray[i] - fYreference) * fYincrement + fYorigin);
writer.Close();

class VisaInstrument
{
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA resource manager.
        OpenResourceManager();

        // Open a VISA resource session.
        OpenSession();

        // Clear the interface.
        int nViStatus;
        nViStatus = visa32.viClear(m_nSession);
public void DoCommand(string strCommand)
{
    // Send the command.
    VisaSendCommandOrQuery(strCommand);

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strCommand);
}

public int DoCommandIEEEBlock(string strCommand, byte[] DataArray)
{
    // Send the command to the device.
    string strCommandAndLength;
    int nViStatus, nLength, nBytesWritten;

    nLength = DataArray.Length;
    strCommandAndLength = String.Format("{0} #8{1:D8}",
     strCommand, nLength);

    // Write first part of command to formatted I/O write buffer.
    nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength);
    CheckVisaStatus(nViStatus);

    // Write the data to the formatted I/O write buffer.
    nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
     out nBytesWritten);
    CheckVisaStatus(nViStatus);

    // Write command termination character.
    nViStatus = visa32.viPrintf(m_nSession, "\n");
    CheckVisaStatus(nViStatus);

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strCommand);

    return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    StringBuilder strResults = new StringBuilder(1000);
    strResults = VisaGetResultString();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
}
public double DoQueryValue(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultValue();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
}

public double[] DoQueryValues(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double[] fResultsArray;
    fResultsArray = VisaGetResultValues();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return fResultsArray;
}

public int DoQueryIEEEBlock(string strQuery,
                            out byte[] ResultsArray)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    int length; // Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(out ResultsArray);

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return length;
}

private void CheckForInstrumentErrors(string strCommand)
{
    // Check for instrument errors.
    StringBuilder strInstrumentError = new StringBuilder(1000);
    bool bFirstError = true;
    do
    {
        VisaSendCommandOrQuery(“:SYSTem:ERRor?”);
strInstrumentError = VisaGetResultString();

if (strInstrumentError.ToString() != "+0,"No error"
) {
    if (bFirstError)
    {
        Console.WriteLine("ERROR(s) for command '{0}': ",
                        strCommand);
        bFirstError = false;
    }    
    Console.Write(strInstrumentError);
}

} while (strInstrumentError.ToString() != "+0,"No error"
);

private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
    // Send command or query to the device.
    string strWithNewline = String.Format("\n", strCommandOrQuery);
    int nViStatus;
    nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
    CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetResultString()
{
    StringBuilder strResults = new StringBuilder(1000);

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
    CheckVisaStatus(nViStatus);

    return strResults;
}

private double VisaGetResultValue()
{
    double fResults = 0;

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
    CheckVisaStatus(nViStatus);

    return fResults;
}

private double[] VisaGetResultValues()
{
    double[] fResultsArray;
    fResultsArray = new double[10];

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, ",%10lf\n",
    // ... (rest of the code continues...
private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
    // Results array, big enough to hold a PNG.
    ResultsArray = new byte[300000];
    int length; // Number of bytes returned from instrument.

    // Set the default number of bytes that will be contained in
    // the ResultsArray to 300,000 (300kB).
    length = 300000;

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
                               ResultsArray);
    CheckVisaStatus(nViStatus);

    // Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
    CheckVisaStatus(nViStatus);
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
    CheckVisaStatus(nViStatus);

    return length;
}

private void OpenResourceManager()
{
    int nViStatus;
    nViStatus = visa32.viOpenDefaultRM(out this.m_nResourceManager);
    if (nViStatus < visa32.VI_SUCCESS)
        throw new ApplicationException("Failed to open Resource Manager");
}

private void OpenSession()
{
    int nViStatus;
    nViStatus = visa32.viOpen(this.m_nResourceManager,
                               this.m_strVisaAddress, visa32.VI_NO_LOCK,
                               visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
    CheckVisaStatus(nViStatus);
}

public void SetTimeoutSeconds(int nSeconds)
{
    int nViStatus;
    nViStatus = visa32.viSetAttribute(this.m_nSession,
                                       visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
    CheckVisaStatus(nViStatus);
}
public void CheckVisaStatus(int nViStatus)
{
    // If VISA error, throw exception.
    if (nViStatus < visa32.VI_SUCCESS)
    {
        StringBuilder strError = new StringBuilder(256);
        visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
            strError);
        throw new ApplicationException(strError.ToString());
    }
}

public void Close()
{
    if (m_nSession != 0)
        visa32.viClose(m_nSession);
    if (m_nResourceManager != 0)
        visa32.viClose(m_nResourceManager);
}

VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2005:
1 Open Visual Studio.
2 Create a new Visual Basic, Windows, Console Application project.
3 Cut-and-paste the code that follows into the Visual Basic .NET source file.
4 Edit the program to use the VISA address of your oscilloscope.
5 Add Agilent's VISA header file to your project:
   a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b Choose Add and then choose Add Existing Item...
   c Navigate to the header file, visa32.vb (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, but do not click the Open button.
   d Click the down arrow to the right of the Add button, and choose Add as Link.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.
   e Right-click the project again and choose Properties; then, select "InfiniiVision.VisaInstrumentApp" as the Startup object.
6 Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

' Agilent VISA Example in Visual Basic .NET
' -------------------------------------------------------------------
' This program illustrates most of the commonly-used programming
' features of your Agilent oscilloscope.
' -------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
    Class VisaInstrumentApp
        Private Shared oscp As VisaInstrument

        Public Shared Sub Main(ByVal args As String())
            Try
                oscp = New VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR")
                Initialize()
                ' The extras function contains miscellaneous commands that
                ' do not need to be executed for the proper operation of
                ' this example. The commands in the extras function are
                ' shown for reference purposes only.

                ' Extra() ' Uncomment to execute the extra function.
                Capture()
                Analyze()
            Catch err As System.ApplicationException
                MsgBox("**** Error : " & err.Message, vbExclamation, _
                    "VISA Error Message")
                Exit Sub
            Catch err As System.SystemException
                MsgBox("**** Error : " & err.Message, vbExclamation, _
                    "System Error Message")
                Exit Sub
            Catch err As System.Exception
                Debug.Fail("Unexpected Error")
                MsgBox("**** Error : " & err.Message, vbExclamation, _
                    "Unexpected Error")
                Exit Sub
            Finally
                oscp.Close()
            End Try
        End Sub

    End Class
End Namespace

' Initialize()
' -------------------------------------------------------------------
' This function initializes both the interface and the
' oscilloscope to a known state.
Private Shared Sub Initialize()
  Dim strResults As StringBuilder

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

  ' Reset the to the defaults.
  oscp.DoCommand("*RST")
  ' Clear the status data structures.
  oscp.DoCommand("*CLS")

  ' IDN - Ask for the device’s *IDN string.
  strResults = oscp.DoQueryString("*IDN?")
  ' Display results.
  Console.Write("Result is: {0}", strResults)

  ' AUTOSCALE - This command evaluates all the input signals
  ' and sets the correct conditions to display all of the
  ' active signals.
  oscp.DoCommand(":AUToscale")

  ' CHANNEL_PROBE - Sets the probe attenuation factor for the
  ' selected channel. The probe attenuation factor may be from
  ' 0.1 to 1000.
  oscp.DoCommand(":CHANnel1:PROBe 10")

  ' CHANNEL_RANGE - Sets the full scale vertical range in volts.
  ' The range value is eight times the volts per division.
  oscp.DoCommand(":CHANnel1:RANGe 8")

  ' TIME_RANGE - Sets the full scale horizontal time in seconds.
  ' The range value is ten times the time per division.
  oscp.DoCommand(":TIMebase:RANGe 2e-3")

  ' TIME_REFERENCE - Possible values are LEFT and CENTER:
  ' - LEFT sets the display reference one time division from
  '   the left.
  ' - CENTER sets the display reference to the center of the
  '   screen.
  oscp.DoCommand(":TIMebase:REFerence CENTER")

  ' TRIGGER_SOURCE - Selects the channel that actually produces
  ' the TV trigger. Any channel can be selected.
  oscp.DoCommand(":TRIGger:TV:SOURCe CHAnnel1")

  ' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
  ' PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
  ' UART, or USB.
  oscp.DoCommand(":TRIGger:MODE EDGE")

  ' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
  ' trigger to either POSITIVE or NEGATIVE.
osc.opn.DoCommand(":^TRIGger:EDGE:SLOPe POSitive")

End Sub

' Extra()
' --------------------------------------------------------------
' The commands in this function are not executed and are shown
' for reference purposes only. To execute these commands, call
' this function from main.

Private Shared Sub Extra()

' RUN_STOP (not executed in this example):
' - RUN starts the acquisition of data for the active
' waveform display.
' - STOP stops the data acquisition and turns off AUTOSTORE.
osc.opn.DoCommand(":^RUN")
ocsp.DoCommand(":^STOP")

' VIEW_BLANK (not executed in this example):
' - VIEW turns on (starts displaying) an active channel or
' pixel memory.
' - BLANK turns off (stops displaying) a specified channel or
' pixel memory.
osc.opn.DoCommand(":^BLANk CHANnel1")
ocsp.DoCommand(":^VIEW CHANnel1")

' TIME_MODE (not executed in this example) - Set the time base
' mode to MAIN, DELAYED, XY or ROLL.
oscop.DoCommand(":^TIMebase:MODE MAIN")

End Sub

' Capture()
' --------------------------------------------------------------
' This function prepares the scope for data acquisition and then
' uses the DIGITIZE MACRO to capture some data.

Private Shared Sub Capture()

' AQUIRE_TYPE - Sets the acquisition mode. There are three
' acquisition types NORMAL, PEAK, or AVERAGE.
ocsp.DoCommand(":^ACQuire:TYPE NORMal")

' 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.
ocsp.DoCommand(":^ACQuire:COMPLETE 100")

' DIGITIZE - Used to acquire the waveform data for transfer
' over the interface. Sending this command causes an
' acquisition to take place with the resulting data being
' placed in the buffer.

' NOTE! The use of the DIGITIZE command is highly recommended
as it will ensure that sufficient data is available for
measurement. Keep in mind when the oscilloscope is running,
communication with the computer interrupts data acquisition.
Setting up the oscilloscope over the bus causes the data
buffers to be cleared and internal hardware to be
reconfigured.
If a measurement is immediately requested there may not have
been enough time for the data acquisition process to collect
data and the results may not be accurate. An error value of
9.9E+37 may be returned over the bus in this situation.

opc.DoCommand(":DIGitize CHANn1")
End Sub

' Analyze()
' --------------------------------------------------------------
' In this example we will do the following:
' - Save the system setup to a file for restoration at a later
time.
' - Save the oscilloscope display to a file which can be
printed.
' - Make single channel measurements.

Private Shared Sub Analyze()

' Results array.
Dim ResultsArray As Byte()
' Number of bytes returned from instrument.
Dim nLength As Integer

' SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a
' program message that contains the current state of the
' instrument. Its format is a definite-length binary block,
' for example,
' #800002204<setup string><NL>
' where the setup string is 2204 bytes in length.
Console.WriteLine("Saving oscilloscope setup to " + _
+ "c:\scope\config\setup.dat")
If File.Exists("c:\scope\config\setup.dat") Then
  File.Delete("c:\scope\config\setup.dat")
End If

' Query and read setup string.
Dim DataArray As Byte()
Dim nBytesWritten As Integer
nLength = opc.DoQueryIEEEBlock(":SYSTem:SETup?", ResultsArray)
Console.WriteLine("Read oscilloscope setup ({0} bytes).", _
nLength)

' Write setup string to file.
File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
nLength)

' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
' string to the oscilloscope.
Dim DataArray As Byte()
Dim nBytesWritten As Integer
' Read setup string from file.
DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
Console.WriteLine("Read setup string {0} bytes) from file.", _
    DataArray.Length)

' Restore setup string.
nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTem:SETup", _
    DataArray)
Console.WriteLine("Restored setup string {0} bytes).", _
    nBytesWritten)

' IMAGE_TRANSFER - In this example, we query for the screen
' data with the ":DISPLAY:DATA?" query. The .png format
' data is saved to a file in the local file system.
Console.WriteLine("Transferring screen image to ", _
    "c:\scope\data\screen.png")
If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
End If

' Increase I/O timeout to fifteen seconds.
oscps.SetTimeoutSeconds(15)

' Get the screen data in PNG format.
nLength = _
    oscp.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, SCReen, COlor", _
        ResultsArray)
Console.WriteLine("Read screen image {0} bytes).", nLength)

' Store the screen data in a file.
File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
Console.WriteLine("Wrote screen image {0} bytes) to file.", _
    nLength)

' Return I/O timeout to five seconds.
oscps.SetTimeoutSeconds(5)

' MEASURE - The commands in the MEASURE subsystem are used to
' make measurements on displayed waveforms.

' Set source to measure.
oscps.DoCommand(":MEASure:SOURce CHANnel1")

' Query for frequency.
Dim fResults As Double
fResults = oscps.DoQueryValue(":MEASure:FREQuency?"
Console.WriteLine("The frequency is: {0:F4} kHz", _
    fResults / 1000)

' Query for peak to peak voltage.
fResults = oscps.DoQueryValue(":MEASure:VPP?"
Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
    fResults)

' WAVEFORM_DATA - Get waveform data from oscilloscope. To
' obtain waveform data, you must specify the WAVEFORM
parameters for the waveform data prior to sending the
"*:WAVeform:DATA?" query.

Once these parameters have been sent, the
"*:WAVeform:PREAMble?" query provides information concerning
the vertical and horizontal scaling of the waveform data.

With the preamble information you can then use the
"*:WAVeform:DATA?" query and read the data block in the
correct format.

WAVE_FORMAT - Sets the data transmission mode for waveform
data output. This command controls how the data is
formatted when sent from the oscilloscope and can be set
to WORD or BYTE format.

Set waveform format to BYTE.
oscP.DoCommand("*:WAVeform:FORMat BYTE")

WAVE_POINTS - Sets the number of points to be transferred.
The number of time points available is returned by the
"*ACQUIRE:POINTS?" query. This can be set to any binary
fraction of the total time points available.
oscP.DoCommand("*:WAVeform:POINts 1000")

GET_PREAMBLE - The preamble contains all of the current
WAVEFORM settings returned in the form <preamble block><NL>
where the <preamble block> is:
  FORMAT : int16 - 0 = BYTE, 1 = WORD, 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 the x-origin.
  YINCREMENT : float32 - voltage difference between data
         points.
  YORIGIN : float32 - value of the voltage at center
         screen.
  YREFERENCE : int32 - data point where y-origin occurs.
Console.WriteLine("Reading preamble.")
Dim fResultsArray As Double() = oscp.DoQueryValues("*:WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
Console.WriteLine("Preamble FORMat: (0:e)", fFormat)

Dim fType As Double = fResultsArray(1)
Console.WriteLine("Preamble TYPE: (0:e)", fType)

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Preamble POINts: (0:e)", fPoints)
Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Preamble COUNT: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Preamble XINCREMENT: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Preamble XORIGIN: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Preamble XREFERENCE: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Preamble YINCREMENT: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YORIGIN: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YREFERENCE: {0:e}", fYreference)

' QUERY_WAVE_DATA - Outputs waveform records to the controller
' over the interface that is stored in a buffer previously
' specified with the ":WAVEform:SOURce" command.

' READ_WAVE_DATA - The wave data consists of two parts: the
' header, and the actual waveform data followed by a
' New Line (NL) character. The query data has the following
' format:
'  
'  <header><waveform data block><NL>
'  
'  Where:
'  
'  <header> = #800002048 (this is an example header)
'  
'  The "#8" may be stripped off of the header and the remaining
'  numbers are the size, in bytes, of the waveform data block.
'  The size can vary depending on the number of points acquired
'  for the waveform which can be set using the
'  ":WAVEform:POINTS" command. You may then read that number
'  of bytes from the oscilloscope; then, read the following NL
'  character to terminate the query.

' Read waveform data.
nLength = oscp.DoQueryIEEEBlock("::WAVEform:DATA?", ResultsArray)
Console.WriteLine("Read waveform data ({0} bytes)", nLength)

' Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

' Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = (0:f)", fOffset)
Console.WriteLine("Seconds per Division = (0:e)", fSdiv)
Console.WriteLine("Delay = (0:e)", fDelay)

' Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < 1000
    Console.WriteLine("Data point (0:d) = (1:f2) Volts at " + _
        "(2:f10) Seconds", i, _
        CSng(ResultsArray(i)) - fYreference) * fYincrement + _
        fYorigin, _
        CSng(i) - fXreference) * fXincrement + fXorigin)
    i = i + 50
End While

' SAVE_WAVE_DATA - saves the waveform data to a CSV format
' file named "waveform.csv".
If File.Exists("c:\scope\data\waveform.csv") Then
    File.Delete("c:\scope\data\waveform.csv")
End If

Dim writer As StreamWriter = _
    File.CreateText("c:\scope\data\waveform.csv")
For index As Integer = 0 To 999
    writer.WriteLine("{0:E}, {1:f6}", _
        CSng(index) - fXreference) * fXincrement + fXorigin, _
        CSng(ResultsArray(index)) - fYreference) * fYincrement _
        + fYorigin)
Next
writer.Close()
End Sub
End Class

Class VisaInstrument
    Private m_nResourceManager As Integer
    Private m_nSession As Integer
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)
        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress
        ' Open the default VISA resource manager.
        OpenResourceManager()
        ' Open a VISA resource session.
        OpenSession()

        ' Clear the interface.
        Dim nViStatus As Integer
        nViStatus = visa32.viClear(m_nSession)
    End Sub

    Public Sub DoCommand(ByVal strCommand As String)
        ' Send the command.
        VisaSendCommandOrQuery(strCommand)
End Class
' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strCommand)
End Sub

Public Function DoCommandIEEEBlock(ByVal strCommand As String, ByVal DataArray As Byte()) As Integer
' Send the command to the device.
Dim strCommandAndLength As String
Dim nViStatus As Integer
Dim nLength As Integer
Dim nBytesWritten As Integer

nLength = DataArray.Length
strCommandAndLength = [String].Format("{0} #8{1:D8}", _
strCommand, nLength)

' Write first part of command to formatted I/O write buffer.
nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
CheckVisaStatus(nViStatus)

' Write the data to the formatted I/O write buffer.
nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength, _
    nBytesWritten)
CheckVisaStatus(nViStatus)

' Write command termination character.
nViStatus = visa32.viPrintf(m_nSession, "" & Chr(10) & ")"
CheckVisaStatus(nViStatus)

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strCommand)

Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) As StringBuilder
' Send the query.
VisaSendCommandOrQuery(strQuery)

' Get the result string.
Dim strResults As New StringBuilder(1000)
strResults = VisaGetResultString()

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)

' Return string results.
Return strResults
End Function

Public Function DoQueryValue(ByVal strQuery As String) As Double
' Send the query.
VisaSendCommandOrQuery(strQuery)

' Get the result string.
Dim fResults As Double
fResults = VisaGetResultValue()
'
' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)
'
' Return string results.
Return fResults
End Function

Public Function DoQueryValues(ByVal strQuery As String) As Double()
' Send the query.
VisaSendCommandOrQuery(strQuery)
'
' Get the result string.
Dim fResultsArray As Double()
fResultsArray = VisaGetResultValues()
'
' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)
'
' Return string results.
Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String, _
ByRef ResultsArray As Byte()) As Integer
' Send the query.
VisaSendCommandOrQuery(strQuery)
'
' Get the result string.
Dim length As Integer
' Number of bytes returned from instrument.
length = VisaGetResultIEEEBlock(ResultsArray)
'
' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)
'
' Return string results.
Return length
End Function

Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
' Check for instrument errors.
Dim strInstrumentError As New StringBuilder(1000)
Dim bFirstError As Boolean = True
Do
    VisaSendCommandOrQuery(":^SYSTem:ERRor?")
    strInstrumentError = VisaGetResultString()
    If strInstrumentError.ToString() <> "+0," & Chr(10) & "" Then
        If bFirstError Then
            Console.WriteLine("ERROR(s) for command '{0}': ", _
                strCommand)
            bFirstError = False
        End If
    End If
    bFirstError = False
Loop
End Sub
Console.Write(strInstrumentError)
End If
Loop While strInstrumentError.ToString() <> _
"+0,""No error"" & Chr(10) & ""
End Sub

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
As String)
' Send command or query to the device.
Dim strWithNewline As String
strWithNewline = [String].Format("{0}" & Chr(10) & ", ", _
strCommandOrQuery)
Dim nViStatus As Integer
nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetResultString() As StringBuilder
Dim strResults As New StringBuilder(1000)
' Read return value string from the device.
Dim nViStatus As Integer
nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
CheckVisaStatus(nViStatus)
Return strResults
End Function

Private Function VisaGetResultValue() As Double
Dim fResults As Double = 0
' Read return value string from the device.
Dim nViStatus As Integer
nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
CheckVisaStatus(nViStatus)

Return fResults
End Function

Private Function VisaGetResultValues() As Double()
Dim fResultsArray As Double()
fResultsArray = New Double(9) {}
' Read return value string from the device.
Dim nViStatus As Integer
nViStatus = visa32.viScanf(m_nSession, ", ", 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 = vis32.viScanf(m_nSession, "%#b", length, _
ResultsArray)
CheckVisaStatus(nViStatus)

Write and read buffers need to be flushed after IEEE block?
Dim nViStatus As Integer
nViStatus = vis32.viFlush(m_nSession, vis32.VI_WRITE_BUF)
CheckVisaStatus(nViStatus)
nViStatus = vis32.viFlush(m_nSession, vis32.VI_READ_BUF)
CheckVisaStatus(nViStatus)

Return length
End Function

Private Sub OpenResourceManager()
Dim nViStatus As Integer
nViStatus = vis32.viOpenDefaultRM(Me.m_nResourceManager)
If nViStatus < vis32.VI_SUCCESS Then
    Throw New _
        ApplicationException("Failed to open Resource Manager")
End If
End Sub

Private Sub OpenSession()
Dim nViStatus As Integer
nViStatus = vis32.viOpen(Me.m_nResourceManager, _
    Me.m_strVisaAddress, vis32.VI_NO_LOCK, _
    vis32.VI_TMO_IMMEDIATE, Me.m_nSession)
CheckVisaStatus(nViStatus)
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
Dim nViStatus As Integer
nViStatus = vis32.viSetAttribute(Me.m_nSession, _
    vis32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
CheckVisaStatus(nViStatus)
End Sub

Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
    ' If VISA error, throw exception.
    If nViStatus < vis32.VI_SUCCESS Then
        Dim strError As New StringBuilder(256)
        vis32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
        Throw New ApplicationException(strError.ToString())
    End If
End Sub

Public Sub Close()
    If m_nSession <> 0 Then
        vis32.viClose(m_nSession)
    End If
If m_nResourceManager <> 0 Then
    visa32.viClose(m_nResourceManager)
End If
End Sub
End Class
End Namespace
SICL Examples

- "SICL Example in C" on page 893
- "SICL Example in Visual Basic" on page 902

SICL Example in C

To compile and run this example in Microsoft Visual Studio 2005:

3. In the Win32 Application Wizard, click Next >. Then, check Empty project, and click Finish.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2005, right-click the Source Files folder, choose Add > Add Existing Item..., select the example.c file, and click Add.
6. Edit the program to use the SICL address of your oscilloscope.
7. Choose Project > Properties.... In the Property Pages dialog, update these project settings:
   c. Click OK to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose Tools > Options....
   b. In the Options dialog, select VC++ Directories under Projects and Solutions.
   c. Show directories for Include files, and add the include directory (for example, Program Files\Agilent\ IO Libraries Suite\include).
   d. Show directories for Library files, and add the library files directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
   e. Click OK to close the Options dialog.
9. Build and run the program.

/*  
* Agilent SICL Example in C  
* -----------------------------------------------  
* This program illustrates most of the commonly-used programming  
* features of your Agilent oscilloscope.  
* This program is to be built as a WIN32 console application.  
*/
* Edit the DEVICE_ADDRESS line to specify the address of the applicable device. */

#include <stdio.h> /* For printf(). */
#include "sicl.h" /* SICL routines. */

/* #define DEVICE_ADDRESS "gpib0,7" */ /* GPIB */
/* #define DEVICE_ADDRESS "lan[a-mso6102-90541]:inst0" */ /* LAN */
#define DEVICE_ADDRESS "usb0[2391::5970::30D3090541::0]" /* USB */

#define WAVE_DATA_SIZE 5000
#define TIMEOUT 5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE 300000

/* Function prototypes */
void initialize(void); /* Initialize the oscilloscope. */
void extra(void); /* Miscellaneous commands not executed, shown for reference purposes. */
void capture(void); /* Digitize data from oscilloscope. */
void analyze(void); /* Make some measurements. */
void get_waveform(void); /* Download waveform data from oscilloscope. */
void save_waveform(void); /* Save waveform data to a file. */
void retrieve_waveform(void); /* Load waveform data from a file. */

/* Global variables */
INST id; /* Device session ID. */
char buf[256] = {0}; /* Buffer for IDN string. */

/* Array for waveform data. */
unsigned char waveform_data[WAVE_DATA_SIZE];
double preamble[10]; /* Array for preamble. */

void main(void)
{
    /* Install a default SICL error handler that logs an error message and exits. On Windows 98SE or Windows Me, view messages with the SICL Message Viewer. For Windows 2000 or XP, use the Event Viewer. */
    ionerror(I_ERROR_EXIT);

    /* Open a device session using the DEVICE_ADDRESS */
    id = iopen(DEVICE_ADDRESS);

    if (id == 0)
    {
        printf ("Oscilloscope iopen failed!
"));
    }
    else
    {
        printf ("Oscilloscope session initialized!
"));

        /* Set the I/O timeout value for this session to 5 seconds. */
        itimeout(id, TIMEOUT);
/* Clear the interface. */
iclear(id);
iremote(id);
}

initialize();

/* The extras function contains miscellaneous commands that do not
* need to be executed for the proper operation of this example.
* The commands in the extras function are shown for reference
* purposes only.
*/
/* extra(); */ /* <-- Uncomment to execute the extra function */
capture();
analyze();

/* Close the device session to the instrument. */
iclose(id);
printf("Program execution is complete...\n");

/* For WIN16 programs, call _siclcleanup before exiting to release
* resources allocated by SICL for this application. This call is
* a no-op for WIN32 programs.
*/
_siclcleanup();
}

/* initialize
* -------------------------------------------------------------
* This function initializes both the interface and the oscilloscope
* to a known state.
*/

void initialize (void)
{
/* RESET - This command puts the oscilloscope in a known state.
* Without this command, the oscilloscope settings are unknown.
* This command is very important for program control.
* Many of the following initialization commands are initialized
* by this command. It is not necessary to reinitialize them
* unless you want to change the default setting.
*/
iprintf(id, "*RST\n");

/* Write the *IDN? string and send an EOI indicator, then read
* the response into buf.
ipromptf(id, "*IDN?\n", "%t", buf);
printf("%s\n", buf);
*/

/* AUTOSCALE - This command evaluates all the input signals and
* sets the correct conditions to display all of the active signals.*/
Programming Examples

*/
iprintf(id, ":AUTOSCALE\n");

/* CHANNEL_PROBE - Sets the probe attenuation factor for the
* selected channel. The probe attenuation factor may be from
* 0.1 to 1000.
*/
iprintf(id, ":CHAN1:PROBE 10\n");

/* CHANNEL_RANGE - Sets the full scale vertical range in volts.
* The range value is eight times the volts per division.
*/
iprintf(id, ":CHANNEL1:RANGE 8\n");

/* TIME_RANGE - Sets the full scale horizontal time in seconds.
* The range value is ten times the time per division.
*/
iprintf(id, ":TIM:RANG 2e-3\n");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
* - LEFT sets the display reference one time division from the
*   left.
* - CENTER sets the display reference to the center of the screen.
*/
iprintf(id, ":TIMEBASE:REFERENCE CENTER\n");

/* TRIGGER_SOURCE - Selects the channel that actually produces the
* TV trigger. Any channel can be selected.
*/
iprintf(id, ":TRIGGER:TV:SOURCE CHANNEL1\n");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLItCh, PATTern,
* CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
*/
iprintf(id, ":TRIGGER:MODE EDGE\n");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
* to either POSITIVE or NEGATIVE.
*/
iprintf(id, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
* extra
* ------------------------------------------------------------------
* The commands in this function are not executed and are shown for
* reference purposes only. To execute these commands, call this
* function from main.
*/
void extra (void)
{
    /* RUN_STOP (not executed in this example):
    * - RUN starts the acquisition of data for the active waveform
    *   display.
    * - STOP stops the data acquisition and turns off AUTOSTORE.
    */
iprintf(id, "\":RUN\n\n");
iprintf(id, "\":STOP\n\n");

/* VIEW_BLANK (not executed in this example):
 * - VIEW turns on (starts displaying) an active channel or pixel
 * memory.
 * - BLANK turns off (stops displaying) a specified channel or
 * pixel memory.
 */
iprintf(id, "\":BLANK CHANNEL1\n\n");
iprintf(id, "\":VIEW CHANNEL1\n\n");

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

/*
 * capture
 * -------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
iprintf(id, "\":ACQUIRE:TYPE NORMAL\n\n");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition. The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
iprintf(id, "\":ACQUIRE:COMPLETE 100\n\n");

    /* DIGITIZE - Used to acquire the waveform data for transfer over
     * the interface. Sending this command causes an acquisition to
     * take place with the resulting data being placed in the buffer.
     */

    /* NOTE! The use of the DIGITIZE command is highly recommended
     * as it will ensure that sufficient data is available for
     * measurement. Keep in mind when the oscilloscope is running,
     * communication with the computer interrupts data acquisition.
     * Setting up the oscilloscope over the bus causes the data
     * buffers to be cleared and internal hardware to be reconfigured.
     * If a measurement is immediately requested there may not have
     * been enough time for the data acquisition process to collect
     * data and the results may not be accurate. An error value of
     * 9.9E+37 may be returned over the bus in this situation.
     */
iprintf(id, "\":DIGITIZE CHAN1\n\n");
}
void analyze (void)
{
    double frequency, vpp; /* Measurements. */
    double vdiv, off, sdiv, delay; /* Calculated from preamble data. */
    int i; /* Loop counter. */
    /* Array for setup string. */
    unsigned char setup_string[SETUP_STR_SIZE];
    int setup_size;
    FILE *fp;
    unsigned char image_data[IMG_SIZE]; /* Array for image data. */
    int img_size;
    /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program */
    /* message that contains the current state of the instrument. Its */
    /* format is a definite-length binary block, for example, */
    /* #800002204<setup string><NL> */
    /* where the setup string is 2204 bytes in length. */
    setup_size = SETUP_STR_SIZE;
    /* Query and read setup string. */
    fprintf(id, ":SYSTEM:SETUP?\n", ":\n", &setup_size, setup_string);
    printf("Read setup string query (%d bytes).\n", setup_size);
    /* Write setup string to file. */
    fp = fopen ("c:\scope\config\setup.dat", "wb");
    setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
                      fp);
    fclose (fp);
    printf("Wrote setup string (%d bytes) to file.\n", setup_size);
    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string */
    /* to the oscilloscope. */
    /* Read setup string from file. */
    fp = fopen ("c:\scope\config\setup.dat", "rb");
    setup_size = fread (setup_string, sizeof(unsigned char),
                         SETUP_STR_SIZE, fp);
    fclose (fp);
    printf("Read setup string (%d bytes) from file.\n", setup_size);
    /* Restore setup string. */
    fprintf(id, ":SYSTEM:SETUP #8%08d", setup_size);
    fwrite(id, setup_string, setup_size, 1, &setup_size);
    printf("Restored setup string (%d bytes).\n", setup_size);
    /* IMAGE_TRANSFER - In this example we will query for the image */
    /* data with ":DISPLAY:DATA?" to read the data and save the data */
    /* to the file "image.dat" which you can then send to a printer. */
itimeout(id, 30000);
printf("Transferring image to c:\scope\data\screen.bmp\n");
img_size = IMG_SIZE;
ipromptf(id, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", 
    &img_size, image_data);
printf("Read display data query (%d bytes)\n", img_size);
/* Write image data to file. */
fp = fopen ("c:\scope\data\screen.bmp", "wb");
img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
fclose (fp);
printf("Wrote image data (%d bytes) to file.\n", img_size);
itimeout(id, 5000);

/* MEASURE - The commands in the MEASURE subsystem are used to 
   * make measurements on displayed waveforms. */

/* Set source to measure. */
iprintf(id, ":MEASURE:SOURCE CHANNEL1\n");

/* Query for frequency. */
ipromptf(id, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
printf("The frequency is: %.4f kHz\n", frequency / 1000);

/* Query for peak to peak voltage. */
ipromptf(id, ":MEASURE:VPP?\n", "%lf", &vpp);
printf("The peak to peak voltage is: %.2f V\n", vpp);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. */
get_waveform();

/* Make some calculations from the preamble data. */
vdiv = 32 * preamble[7];
off = preamble[8];

/* Print them out... */
printf ("Scope Settings for Channel 1:\n");
printf ("Volts per Division = %f\n", vdiv);
printf ("Offset = %f\n", off);
printf ("Seconds per Division = %f\n", sdiv);
printf ("Delay = %f\n", delay);

/* print out the waveform voltage at selected points */
for (i = 0; i < 1000; i = i + 50)
  printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i, 
    (float)waveform_data[i] - preamble[9]) * preamble[7] + 
    preamble[8], 
    (float)i - preamble[6]) * preamble[4] + preamble[5]);

save_waveform(); /* Save waveform data to disk. */
retrieve_waveform(); /* Load waveform data from disk. */

/*
void get_waveform (void)
{
    int waveform_size;

    /* WAVEFORM_DATA - To obtain waveform data, you must specify the
        WAVEFORM parameters for the waveform data prior to sending the
        "::WAVEFORM::DATA?" query.
        
        Once these parameters have been sent, the "::WAVEFORM::PREAMBLE?"
        query provides information concerning the vertical and horizontal
        scaling of the waveform data.
        
        With the preamble information you can then use the
        "::WAVEFORM::DATA?" query and read the data block in the
        correct format.
        */
    
    /* WAVE_FORMAT - Sets the data transmission mode for waveform data
        output. This command controls how the data is formatted when
        sent from the oscilloscope and can be set to WORD or BYTE format.
        */
    
    /* Set waveform format to BYTE. */
    iprintf(id, "::WAVEFORM::FORMAT BYTE\n");

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

    /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
        settings returned in the form <preamble block><NL> where the
        <preamble block> is:
        
        FORMAT : int16 - 0 = BYTE, 1 = WORD, 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 the x-origin.
        YINCREMENT : float32 - voltage difference between data points.
        YORIGIN : float32 - value of the voltage at center screen.
        YREFERENCE : int32 - data point where y-origin occurs.
        */
    printf("Reading preamble\n");
    ipromptf(id, "::WAVEFORM::PREAMBLE?\n", "%10lf\n", preamble);
    /*
    printf("Preamble FORMAT: %e\n", preamble[0]);
printf("Preamble TYPE: %e\n", preamble[1]);
printf("Preamble POINTS: %e\n", preamble[2]);
printf("Preamble COUNT: %e\n", preamble[3]);
printf("Preamble XINCREMENT: %e\n", preamble[4]);
printf("Preamble XORIGIN: %e\n", preamble[5]);
printf("Preamble XREFERENCE: %e\n", preamble[6]);
printf("Preamble YINCREMENT: %e\n", preamble[7]);
printf("Preamble YORIGIN: %e\n", preamble[8]);
printf("Preamble YREFERENCE: %e\n", preamble[9]);
*/

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEFORM:SOURCE" command.
 */
 printf(id, ":WAVEFORM:DATA?\n"); /* Query waveform data. */

/* READ_WAVE_DATA - The wave data consists of two parts: the header,
 * and the actual waveform data followed by an New Line (NL)
 * character. The query data has the following format:
 *
 * <header><waveform data block><NL>
 *
 * Where:
 *
 * <header> = #800002048 (this is an example header)
 *
 * The ":8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the ":WAVEFORM:POINTS"
 * command. You may then read that number of bytes from the
 * oscilloscope; then, read the following NL character to
 * terminate the query.
 */

waveform_size = WAVE_DATA_SIZE;
/* Read waveform data. */
 iscanf(id, ":%b\n", &waveform_size, waveform_data);
if ( waveform_size == WAVE_DATA_SIZE )
{
 printf("Waveform data buffer full: ");
 printf("May not have received all points.\n");
} else
{
 printf("Reading waveform data... size = %d\n", waveform_size);
}

/*/ save_waveform
 */

save_waveform(void)


```c
{  
  FILE *fp;

  fp = fopen("c:\scope\data\wave.dat", "wb");
  /* Write preamble. */
  fwrite(preamble, sizeof(preamble[0]), 10, fp);
  /* Write actually waveform data. */
  fwrite(waveform_data, sizeof(waveform_data[0]),
        (int)preamble[2], fp);
  fclose (fp);
}

/*
 * retrieve_waveform
 * --------------------------------------------------------------
 * This function retrieves previously saved waveform data from a
 * file called "wave.dat".
 */

void retrieve_waveform(void)
{
  FILE *fp;

  fp = fopen("c:\scope\data\wave.dat", "rb");
  /* Read preamble. */
  fread (preamble, sizeof(preamble[0]), 10, fp);
  /* Read the waveform data. */
  fread (waveform_data, sizeof(waveform_data[0]),
         (int)preamble[2], fp);
  fclose (fp);
}
```

**SICL Example in Visual Basic**

To run this example in Visual Basic for Applications:

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Add the sicl32.bas file to your project:
   a. Choose **File>Import File...**
   b. Navigate to the header file, sicl32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\Agilent\IO Libraries Suite\include directory), select it, and click **Open**.
4. Choose **Insert>Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the SICL address of your oscilloscope, and save the changes.
7. Run the program.
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 B{array}size = 500000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned ' by ivscanf.
Public strQueryResult As String * 200

' Main Program
' ---------------

Sub Main()

On Error GoTo ErrorHandler

' Open a device session using the SICL_ADDRESS.
id = iopen("lan[130.29.69.12]:inst0")
Call itimeout(id, 5000)

' Initialize - start from a known state.
Initialize

' Capture data.
Capture

' Analyze the captured waveform.
Analyze

' Close the vi session and the resource manager session.
Call iclose(id)
Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End
End Sub

' Initialize the oscilloscope to a known state.
Private Sub Initialize()

    On Error GoTo ErrorHandler

    ' Clear the interface.
    Call iclear(id)

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

    Exit Sub

ErrorHandler:

    MsgBox "*** Error: " + Error, vbExclamation
    End

End Sub

Private Sub Capture()

    On Error GoTo ErrorHandler

    ' Use auto-scale to automatically configure oscilloscope.
    ' -----------------------------------------------------------------
    DoCommand ":AUToscale"

    ' Save oscilloscope configuration.
    ' -----------------------------------------------------------------
    Dim lngSetupStringSize As Long
    lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
    Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

    ' Output setup string to a file:
    Dim strPath As String
    strPath = "c:\scope\config\setup.dat"

    ' Open file for output.
    Dim hFile As Long
    hFile = FreeFile
    Open strPath For Binary Access Write Lock Write As hFile
    Dim lngI As Long
    For lngI = 0 To lngSetupStringSize - 1
        Put hFile, , byteArray(lngI) ' Write data.
    Next lngI
    Close hFile ' Close file.
' Or, configure the settings with individual commands:
' -----------------------------------------------------------------

' Set trigger mode and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
  DoQueryString(":\TRIGger:MODE?"

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
Debug.Print "Trigger edge source: " + _
  DoQueryString(":\TRIGger:EDGE:SOURce?"

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
  DoQueryString(":\TRIGger:EDGE:LEVel?"

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
  DoQueryString(":\TRIGger:EDGE:SLOPe?"

' Set vertical scale and offset.
DoCommand ":CHANnell:SCALe 0.5"
Debug.Print "Channel 1 vertical scale: " + _
  DoQueryString(":\CHANnell:SCALe?"

DoCommand ":CHANnell:OFFSet 1.5"
Debug.Print "Channel 1 vertical offset: " + _
  DoQueryString(":\CHANnell:OFFSet?"

' Set horizontal scale and offset.
DoCommand ":TIMebase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
  DoQueryString(":\TIMebase:SCALe?"

DoCommand ":TIMebase:POSition 0.0"
Debug.Print "Timebase position: " + _
  DoQueryString(":\TIMebase:POSition?"

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
DoCommand ":ACQuire:TYPE NORMal"
Debug.Print "Acquire type: " + _
  DoQueryString(":\ACQuire:TYPE?"

' Or, configure by loading a previously saved setup.
' -----------------------------------------------------------------
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile) ' Length of file.
Get hFile, , byteArray ' Read data.
Close hFile ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup" ' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":\SYSTem:SETup", lngSetupFileSize)
Debug.Print "Setup bytes restored: " + CStr(lngRestored)

' Acquire data.
' -----------------------------------------------------------------
DoCommand "::DIGitize"

Exit Sub

ErrorHandler:

MsgBox "*** Error :" + Error, vbExclamation
End

End Sub

' Analyze the captured waveform.
' -------------------------------------------------------------------

Private Sub Analyze()

On Error GoTo ErrorHandler

' Make a couple of measurements.
' -----------------------------------------------------------------
DoCommand "::MEASure:SOURce CHANnel1"
Debug.Print "Measure source: " + _
   DoQueryString("::MEASure:SOURce?")
DoCommand "::MEASure:VAMPlitude"
dblQueryResult = DoQueryNumber("::MEASure:VAMPlitude?")
MsgBox "Vertical amplitude:" + vbCrLf + _
   FormatNumber(dblQueryResult, 4) + " V"
DoCommand "::MEASure:FREQuency"
dblQueryResult = DoQueryNumber("::MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
   FormatNumber(dblQueryResult / 1000, 4) + " kHz"

' Download the screen image.
' -----------------------------------------------------------------
' Get screen image.
Dim lngBlockSize As Long
lngBlockSize = _
   DoQueryIEEEBlock_Bytes("::DISplay:DATA? PNG, SCReen, COLor")
Debug.Print "Image IEEEBlock bytes: " + CStr(lngBlockSize)

' Save screen image to a file:
Dim strPath As String
strPath = "c:\scope\data\screen.png"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 10 To lngBlockSize - 1 ' Skip past 10-byte header.
   Put hFile, , byteArray(lngI) ' Write data.
Next lngI
Close hFile ' Close file.
MsgBox "Screen image written to " + strPath

' Download waveform data.
' -----------------------------------------------------------------
Dim lngPoints As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double
Dim dblYReference As Double

' Set the waveform source.
DoCommand " :WAVEform:SOURce CHANnel1"
Debug.Print "Waveform source: " + 
   DoQueryString(" :WAVEform:SOURce?")

' Get the number of waveform points:
' How do you get max depth like when saving CSV from front panel?
dblQueryResult = DoQueryNumber(" :WAVEform:POINts?")
lngPoints = dblQueryResult
Debug.Print "Waveform points, channel 1: " + 
   CStr(lngPoints)

' Display the waveform settings:
dblXIncrement = DoQueryNumber(" :WAVEform:XINCrement?")
Debug.Print "Waveform X increment, channel 1: " + 
   Format(dblXIncrement, "Scientific")
dblXOrigin = DoQueryNumber(" :WAVEform:XORigin?")
Debug.Print "Waveform X origin, channel 1: " + 
   Format(dblXOrigin, "Scientific")
dblYIncrement = DoQueryNumber(" :WAVEform:YINCrement?")
Debug.Print "Waveform Y increment, channel 1: " + 
   Format(dblYIncrement, "Scientific")
dblYOrigin = DoQueryNumber(" :WAVEform:YORigin?")
Debug.Print "Waveform Y origin, channel 1: " + 
   Format(dblYOrigin, "Scientific")
dblYReference = DoQueryNumber(" :WAVEform:YREFerence?")
Debug.Print "Waveform Y reference, channel 1: " + 
   Format(dblYReference, "Scientific")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand " :WAVEform:FORMat BYTE"
Debug.Print "Waveform format: " + 
   DoQueryString(" :WAVEform:FORMat?")
' Data in range 0 to 255.
Dim lngVSteps As Long
Dim intBytesPerData As Integer
lngVSteps = 256
intBytesPerData = 1

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(" :WAVEform:DATA?")
Debug.Print "Waveform data IEEEBlock bytes: " + CStr(lngNumBytes)
' Set up output file:
strPath = "C:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long
For lngI = 10 To lngNumBytes - 2 ' Skip past 10-byte header.
    lngDataValue = CLng(byteArray(lngI))
    ' Write time value, voltage value.
    Print #hFile, 
    Format(dblXOrigin + lngI * dblXIncrement, "Scientific") + 
    ", " + 
    FormatNumber((lngDataValue - dblYReference) * dblYIncrement + 
    dblYOrigin)
Next lngI

' Close output file.
Close hFile ' Close file.
MsgBox "Waveform format BYTE data written to " + 
"C:\scope\data\waveform_data.csv."

Exit Sub

ErrorHandler:
    MsgBox "*** Error :" + Error, vbExclamation
End Sub

Private Sub DoCommand(command As String)
    On Error GoTo ErrorHandler
    Call ivprintf(id, command + vbLf)
    CheckForInstrumentErrors command
    Exit Sub
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 fwrite(id, byteArray(), lngBlockSize, vbNull, retCount)
' retCount is now actual number of bytes written.
CheckForInstrumentErrors command
DoCommandIEEEBlock = retCount

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryString(query As String) As String

Dim actual As Long
On Error GoTo ErrorHandler

Dim ret_val As Integer
Dim strResult As String * 200

Call ivprintf(id, query + vbCrLf)
Call ivscanf(id, "%200t", strResult)
CheckForInstrumentErrors query
DoQueryString = strResult

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryNumber(query As String) As Double

On Error GoTo ErrorHandler

Dim dblResult As Double

Call ivprintf(id, query + vbCrLf)
Call ivscanf(id, "%lf" + vbCrLf, dblResult)
CheckForInstrumentErrors query
DoQueryNumber = dblResult

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End
End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

    On Error GoTo ErrorHandler

    ' Send query.
    Call ivprintf(id, query + vbLf)
    ' Read definite-length block bytes.
    Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)
    ' retCount is now actual number of bytes returned by read.
    CheckForInstrumentErrors query
    DoQueryIEEEBlock_Bytes = retCount

    Exit Function

ErrorHandler:

    MsgBox "**** Error : " + Error, vbExclamation
End

End Function

Private Sub CheckForInstrumentErrors(strCmdOrQuery As String)

    On Error GoTo ErrorHandler

    Dim strErrVal As String * 200
    Dim strOut As String

    Do
        Call ivprintf(id, "SYSTem:ERRor?" + vbLf) ' Request error message.
        Call ivscanf(id, "%200t", strErrVal) ' Read: Errno,"Error String".
        If Val(strErrVal) <> 0 Then
            strOut = strOut + "INST Error: " + RTrim(strErrVal) + vbLf
        End If
    Loop While Val(strErrVal) <> 0 ' End if find: 0,"No Error".

    If Not strOut = "" Then
        MsgBox strOut, vbExclamation, "INST Error Messages," + _
            strCmdOrQuery
        Call iflush(id, I_BUF_DISCARD_READ Or I_BUF_DISCARD_WRITE)
    End If

    Exit Sub

ErrorHandler:

    MsgBox "**** Error: " + Error, vbExclamation

End Sub
Index

Symbols

+9.9E+37, infinity representation, 810
+9.9E+37, measurement error, 312

Numerics

0 (zero) values in waveform data, 629
1 (one) values in waveform data, 629
10 MHz REF BNC, enabling/disabling, 461
10 MHz reference signal, 189
50% trigger level, 474
82350A GPIB interface, 4

A

AC coupling, trigger edge, 503
AC input coupling for specified channel, 221
accumulate activity, 142
acknowledge, 740
ACQuire commands, 181
acquire data, 150, 196
acquire mode on autoscale, 146
acquire reset conditions, 128
acquire sample rate, 195
ACQuire subsystem, 51
acquired data points, 188
acquisition anti-alias control, 183
acquisition count, 185
acquisition mode, 181, 187, 646
acquisition type, 181, 196
acquisition types, 621
active edges, 142
active printer, 284
activity logic levels, 142
activity on digital channels, 142
add function, 641
add math function, 273
add math function as g(i) source, 269
address field size, IIC serial decode, 434
address, IIC trigger pattern, 548
Addresses softkey, 38
AER (Arm Event Register), 143, 165, 167, 770
Agilent Connection Expert, 39
Agilent Interactive IO application, 43
Agilent IO Control icon, 39
Agilent IO Libraries Suite, 4, 35, 48, 50
Agilent IO Libraries Suite, installing, 36
ALB waveform data format, 418
alignment, I2S trigger, 531
ALL segments waveform save option, 420
alphabetical list of commands, 655
AMASK commands, 656
amplitude, vertical, 346
analog channel coupling, 221
analog channel display, 222
analog channel impedance, 223
analog channel input, 699
analog channel inversion, 224
analog channel labels, 225, 250
analog channel offset, 226
analog channel protection lock, 452
analog channel range, 233
analog channel scale, 234
analog channel source for glitch, 528
analog channel units, 235
analog channels only oscilloscopes, 4
analog probe attenuation, 227
analog probe head type, 228
analog probe sensing, 700
analog probe skew, 230, 698
analyzing captured data, 47
angle brackets, 109
annotate channels, 225
anti-alias control, 183
AREA commands, 656
area for hardcopy print, 283
area for saved image, 408
Arm Event Register [AER], 143, 165, 167, 770
arrange waveforms, 702
ASCII format, 631
ASCII format for data transfer, 625
ASCII string, quoted, 225
auto setup for M1553 trigger, 428
autoscale for FLEXray trigger, 509
autotask commands, 656
binary block data, 109, 453, 629
BINary waveform data format, 418
bind levels for masks, 386
bit order, 603
bit order, SPI decode, 438
bit selection command, bus, 200
bit weights, 120
bitmap display, 247
B1Order commands, 657
bits in Service Request Enable Register, 133
bits in Standard Event Status Enable Register, 119
bits in Status Byte Register, 135
bits selection command, bus, 201
block data, 109, 123, 247, 453
block response data, 54
blocking synchronization, 779
blocking wait, 778
BMP [bitmap] hardcopy format, 711
braces, 108
built-in measurements, 47
burst, minimum time before next, 500
bus bit selection command, 200
bus bits selection commands, 201
bus clear command, 203
bus commands, 199
BUS data format, 626
bus display, 204
bus label command, 205
bus mask command, 206
BUS<n> commands, 198
button disable, 450
BWLimit commands, 657
byte format for data transfer, 625, 631
BYeOrder, 627
Index

C

C, SICL library example, 893
C, VISA library example, 847
C#, VISA COM example, 824
C#, VISA example, 866
CAL PROTECT switch, 207, 214
calculating preshoot of waveform, 329
calculating the waveform overshoot, 325
calibrate, 209, 210, 214, 216
CALibrate commands, 207
calibrate date, 209
calibrate introduction, 207
calibrate label, 210
calibrate output, 211
calibrate start, 212
calibrate status, 213
calibrate switch, 214
calibrate temperature, 215
calibrate time, 216
CAN, 482
CAN acknowledge, 486, 740
CAN baud rate, 487
CAN commands, 658
CAN frame counters, reset, 425
CAN id pattern, 484
CAN signal definition, 488
CAN source, 489
CAN trigger, 483, 490
CAN trigger commands, 480
CAN trigger pattern id mode, 485
CAN triggerning, 469
capture data, 150
capturing data, 46
CDISplay, 149
center frequency set, 266, 267
center of screen, 654
center reference, 462
center screen, vertical value at, 272, 275
channel, 180, 225, 695, 697
channel coupling, 221
channel display, 222
channel input impedance, 223
channel inversion, 224
channel label, 225, 696
channel labels, 249, 250
channel numbers, 702
channel overload, 232
channel probe ID, 259
channel protection, 232
channel reset conditions, 128
channel selected to produce trigger, 528, 597
channel signal type, 231
channel skew for oscilloscope probe, 230, 698
channel status, 177, 702
channel threshold, 697
channel vernier, 236
channel, stop displaying, 148
CHANnel<n> commands, 217, 218
channels to autoscale, 147
channels, how autoscale affects, 144
characters to display, 448
classes of input signals, 280
classifications, command, 788
clear, 246
clear bus command, 203
CLEAR commands, 659
clear cumulative edge variables, 695
clear display, 149
clear markers, 314, 716
clear measurement, 314, 716
clear message queue, 117
clear method, 49
clear screen, 703
clear status, 117
clear waveform area, 244
clipped high waveform data value, 629
clipped low waveform data value, 629
clock, 551, 585, 586, 590
CLOCK commands, 659
clock slope, I2S, 533
CLOCK source, I2S, 540
CLS (Clear Status), 117
CME (Command Error) status bit, 119, 121
CMOS threshold voltage for digital channels, 243, 697
CMOS trigger threshold voltage, 742
code, ACQuire:COMplete, 184
code, ACQuire:SEGMENTed, 192
code, ACQuire:TYPE, 107
code, AUTOscale, 145
code, CHANnel<n>:LABel, 225
code, CHANnel<n>:PROBe, 227
code, CHANnel<n>:RANGe, 233
code, DIGitize, 150
code, DISPlay:DATA, 248
code, DISPlay:LABel, 249
code, DISPlay:ORDer, 702
code, MEASure:PERiod, 338
code, MEASure:RESULTs, 331
code, MEASure:TEDGE, 343
code, MTESt, 361
code, PODelement:Threshold, 396
code, RUN:/STOP, 174
code, SYSTem:SEtp, 453
code, TIMebase:DELAY, 739
code, TIMebase:MODE, 458
code, TIMebase:RANGe, 460
code, TIMebase:REFERENCE, 464
code, TRIGger:MODE, 475
code, TRIGger:SLOPe, 506
code, TRIGger:SOURCe, 507
code, VIEW and :BLANK, 180
code, :WAVeform, 642
code, :WAVeform:DATA, 629
code, :WAVeform:POINts, 633
code, :WAVeform:PREamble, 637
code, :WAVeform:SEGMENTed, 192
code, :RST, 130
code, SICL library example in C, 893
code, SICL library example in Visual Basic, 902
code, VISA COM library example in C, 824
code, VISA COM library example in Visual Basic, 814
code, VISA COM library example in Visual Basic .NET, 836
code, VISA library example in C, 847
code, VISA library example in C#, 866
code, VISA library example in Visual Basic, 856
code, VISA library example in Visual Basic .NET, 879
colon, root commands prefixed by, 141
color palette for hardcopy, 289
color palette for image, 412
Comma Separated Values (CSV) hardcopy format, 711
Comma Separated Values (CSV) waveform data format, 418
command classifications, 788
command errors detected in Standard Event Status, 121
command header, 790
command headers, common, 792
command headers, compound, 791
command headers, simple, 791
command strings, valid, 789
command tree, 793
commands by subsystem, 111
commands in alphabetical order, 655
commands quick reference, 59
commands sent over interface, 116
commands, more about, 787
commands, obsolete and discontinued, 689
common (*) commands, 112, 113, 116
common command headers, 792
completion criteria for an acquisition, 184, 185
compound command headers, 791
compound header, 808
computer control examples, 813
conditions for external trigger, 254
conditions, reset, 128
configurations, oscilloscope, 123, 127, 131, 453
Configure softkey, 38
connect oscilloscope, 37
connect sampled data points, 701
constants for making automatic measurements, 227
constants for scaling display factors, 227
constants for setting trigger levels, 227
Control softkey, 37, 38
coupler initialization, 46
copy display, 173
core commands, 788
count, 577, 628
COUNT commands, 659
count values, 185
count, Nth edge of burst, 499
counter, 315
coupling, 503
COUPling commands, 660
coupling for channels, 221
create automask, 364
CSV (Comma Separated Values) hardcopy format, 711
Index

enabling channel display, 222
enabling status register bits, 118, 132
end of string (EOS) terminator, 790
end of text (EOT) terminator, 790
end or identify (EOI), 790
enter pattern, 477
EOI (end or identify), 790
EOS (end of string) terminator, 790
EOT (end of text) terminator, 790
Epson, 709
equivalent-time acquisition mode, 182, 187
erase data, 149, 246
erase functions, 149
erase measurements, 716
erase screen, 703
ERRor commands, 662
error frame count (CAN), 423
error frame count (UART), 441
error messages, 449, 745
error number, 449
error queue, 449, 767
error, measurement, 312
ESB (Event Status Bit), 133, 135
ESE (Standard Event Status Enable Register), 118, 786
ESR (Standard Event Status Register), 120, 765
EVENT commands, 662
event status conditions occurred, 135
Event Status Enable Register (ESE), 118, 786
Event Status Register (ESR), 120, 179, 765
excample code, :ACQuire:COMPlete, 184
excample code, :ACQuire:SEGMenTed, 192
excample code, :ACQuire:TYPE, 197
excample code, :AUtOScale, 145
excample code, :CHANnel<n>:LABel, 225
excample code, :CHANnel<n>:PROP兹, 227
excample code, :CHANnel<n>:RANGE, 233
excample code, :DIGItize, 150
excample code, :DISPlay:DATA, 248
excample code, :DISPlay:LABel, 249
excample code, :DISPlay:ORDer, 702
excample code, :MEASURE:PERiod, 338
excample code, :MEASURE:RESults, 331
excample code, :MEASURE:TEDGe, 343
excample code, :MEASURE:TDM, 351
excample code, :MEASURE:TDM, 351
ex: :MEASURE:THReshold, 396
excample code, :RUN/STOP, 174
excample code, :SYStem:Setup, 453
excample code, :TIMebase:DElay, 739
excample code, :TIMebase:MODE, 458
excample code, :TIMebase:RANGE, 460
excample code, :TIMebase:REFerence, 462
excample code, :TRIGger:MODE, 475
excample code, :TRIGger:SLPe, 506
excample code, :TRIGger:SOURce, 507
excample code, :VIEW and :BLANK, 180
excample code, :WAVeform, 642
excample code, :WAVeform:DATA, 629
excample code, :WAVeform:POInts, 633
excample code, :WAVeform:PREamble, 637
ex: :WAVeform:SEGMenTed, 192
example code, *RST, 130
example programs, 4, 813
EXE (Execution Error) status bit, 119, 121
execution error detected in Standard Event
Status, 121
exponential notation, 108
external glitch trigger source, 528
external range, 262
external trigger, 254, 257, 258, 507, 704
EXTernal trigger commands, 254
external trigger input impedance, 257, 704
EXTernal trigger level, 504
external trigger overload, 261
external trigger probe attenuation factor, 258
external trigger probe ID, 259
external trigger probe sensing, 705
external trigger protection, 261
external trigger signal type, 260
EXTernal trigger source, 507
external trigger units, 263
F
FACTion commands, 662
FACTors commands, 662
fail (mask test) output, 377
failed waveforms in mask test, 369
failure, self test, 137
fall time measurement, 312, 322
falling edge, 477, 578
Fast Fourier Transform (FFT) functions, 266
FF values in waveform data, 629
FFT (Fast Fourier Transform) functions, 266
FTT (Fast Fourier Transform) operation, 273
FFT math function, 186
fifty ohm impedance, disable setting, 452
fifty percent trigger level, 474
FILENAME commands, 663
filename for hardcopy, 710
filename for recall, 399
filename for save, 406
filter for frequency reject, 505
filter for high frequency reject, 472
filter for noise reject, 476
filter used to limit bandwidth, 220, 256
filters to Fast Fourier Transforms, 280
find stage in sequence trigger, 579
fine horizontal adjustment (vernier), 464
done vertical adjustment (vernier), 236
finish pending device operations, 124
first point displayed, 651
FLATtop window for amplitude
measurements, 280
FLEXray commands, 663
FLEXray frame counters, reset, 430
FLEXray source, 518
FLEXray trigger, 519
FLEXray trigger autosetup, 509
FLEXray trigger commands, 508
FLEXray triggering, 469
format, 631, 636
FORMat commands, 663
format for block data, 123
format for generic video, 594, 598
format for hardcopy, 708, 711
format for image, 410
format for waveform data, 418
FormattedI0488 object, 49
tormfeed for hardcopy, 282, 286
formulas for data conversion, 623
frame, 592
FRAME commands, 663
frame counters (CAN), 423
frame counters (CAN), overload, 424
frame counters (CAN), reset, 425
frame counters (CAN), total, 426
frame counters (FlexRay), null, 429, 431
frame counters (FlexRay), reset, 430
frame counters (UART), 441
frame counters (UART), reset, 442
frame counters (UART), Rx frames, 443
frame counters (UART), Tx frames, 444
frame ID, FLEXray frame trigger, 516
frame type, FLEXray frame trigger, 517
framing, 587
FRAMing commands, 664
frequency measurement, 47, 312, 323
drequency resolution, 280
frequency span of display, 279
drequency versus dB, 266
front panel mode, 479
front panel Single key, 176
front panel Stop key, 178
front-panel lock, 450
du-scale horizontal time, 460, 466
du-scale vertical axis defined, 274
dunction, 180, 267, 268, 272, 273, 274, 275,
d, 278, 279, 280, 706, 707
FUNCTION commands, 264
dunction memory, 177
dunction turned on or off, 707
dunctions, 641
dunctions, erasing, 149
G
g(t) source, first input channel, 270
g(t) source, math operation, 269
g(t) source, second input channel, 271
gateway IP, 37
general trigger commands, 471
gENERic, 594, 598
generic video format, 594, 598
glitch duration, 527
glitch qualifier, 526
glitch source, 528
GLITch trigger commands, 520
glitch trigger duration, 522
glitch trigger polarity, 525
glitch trigger source, 522
GOFT commands, 664
graphics, 247
graticule area for hardcopy print, 283
graticule area for saved image, 408
graticule colors, invert for hardcopy, 287, 713
graticule colors, invert for image, 411
graticule data, 247
grayscale palette for hardcopy, 289
grayscale palette for image, 412
grayscale coding on hardcopy, 712
greater than qualifier, 526
greater than time, 493, 497, 522, 527
GREaterthan commands, 664
groups of digital channels, 393, 394, 396, 697

H
HANning window for frequency resolution, 280
hardcopy, 173, 282
HARDcopy commands, 281
hardcopy device, 709
hardcopy factors, 285, 409
hardcopy filename, 710
hardcopy format, 708, 711
hardcopy formfeed, 286
hardcopy grayscale, 712
hardcopy invert graticule colors, 287, 713
hardcopy layout, 288
hardcopy palette, 289
hardcopy print, area, 283
hardcopy printer driver, 714
hardware event condition register, 154
Hardware Event Condition Register (:HWERegister:CONDition), 154
Hardware Event Condition Register (:OPERegister:CONDition), 773
Hardware Event Enable Register (HWEnable), 152
hardware event enable register, 156
Hardware Event Register (:HWERegister[:EVENT]), 156, 772
head type, probe, 228
header, 790
high resolution acquisition type, 623
high-frequency reject filter, 472, 505
high-resolution acquisition type, 182
hold until operation complete, 124
holdoff time, 473
holes in waveform data, 629
horizontal adjustment, fine (vernier), 464
horizontal position, 465
horizontal scale, 463, 467
horizontal scaling, 636
horizontal time, 460, 466, 717
hostname, 37
HWEnable (Hardware Event Enable Register), 152
HWERegister:CONDition (Hardware Event Condition Register), 154, 773
HWERegister[:EVENT] (Hardware Event Register), 156, 772

I
I/O softkey, 37, 38
I1080L60HZ, 594, 598
I1080L60HZ, 594, 598
I2S alignment, 531
I2S audio channel, 532
I2S clock slope, 533
I2S CLOCK source, 540
I2S commands, 665
I2S DATA source, 541
I2S pattern data, 534
I2S pattern format, 536
I2S range, 537
I2S receiver width, 539
I2S serial decode base, 433
I2S transmit word size, 545
I2S trigger commands, 529
I2S trigger operator, 543
I2S triggering, 469
I2S word select (WS) low, 546
I2S word select (WS) source, 542
ID commands, 665
id mode, 485
identification number, 122
identification of options, 125
identifier, LIN, 558
IDLE commands, 665
idle until operation complete, 124
IDN (Identification Number), 122
IEEE 488.2 standard, 116
IColors commands, 666
I address, 548
I clock, 551
IC commands, 665
ID data, 549, 552
ID data 2, 550
IC serial decode address field size, 434
IIC trigger commands, 547
IC trigger qualifier, 553
IIC trigger type, 554
IIC triggering, 489
IMAGE commands, 666
image format, 410
image invert graticule colors, 411
image memory, 177, 252
image palette, 412
image, recall, 400
image, save, 407
image, save with inksaver, 411
 impedance, 223
IMPedance commands, 666
 impedance for external trigger input, 257, 704
infinity representation, 810
initialization, 46, 49
initialize, 128
initialize label list, 250
initialize acquisition, 150
inksaver, save image with, 411
input, 257, 704
input coupling for channels, 221
input impedance for channels, 223, 699
input impedance for external trigger, 257, 704
input inversion for specified channel, 224
insert label, 225
installed options identified, 125
instruction header, 790
instrument number, 122
instrument options identified, 125
instrument requests service, 135
instrument serial number, 175
instrument settings, 282
instrument status, 56
instrument type, 122
integrate math function, 266, 273, 641
INtegrate source for function, 278, 706
INTERN files, 252
internal low-pass filter, 220, 256
introduction to :ACQuire commands, 181
introduction to :BUS<n> commands, 199
introduction to :CALibrate commands, 207
introduction to :CHANnel<n> commands, 218
introduction to :DIGital<n> commands, 237
introduction to :DISPLAY commands, 244
introduction to :EXTernal commands, 254
introduction to :FUNCTION commands, 266
introduction to :HARDcopy commands, 282
introduction to :LISTer commands, 292
introduction to :MARKer commands, 296
introduction to :MEASure commands, 312
introduction to :POD<n> commands, 393
introduction to :RECall commands, 398
introduction to :SAVE commands, 405
introduction to :SBUS commands, 422
introduction to :SYSTEM commands, 446
introduction to :TIMEbase commands, 457
introduction to :TRIGger commands, 468
introduction to :WAVEform commands, 621
introduction to common (*) commands, 116
introduction to root (:) commands, 141
invert graticule colors for hardcopy, 287, 713
invert graticule colors for image, 411
inverted masks, bind levels, 386
inverting input for channels, 224
ID library, referencing, 48
IP address, 37
IP Options softkey, 38

K
key disable, 450
key press detected in Standard Event Status Register, 121
knob disable, 450
known state, 128

L
label, 240, 696
label command, bus, 205
LABel commands, 666
measurements, time between trigger and edge, 342
measurements, time between trigger and vertical value, 344
measurements, time between trigger and voltage level, 723
measurements, upper threshold value, 725
measurements, vertical amplitude, 346
measurements, vertical peak-to-peak, 351
measurements, voltage difference, 726
memory setup, 131, 453
merge, 158
message available bit, 135
message available bit clear, 117
message displayed, 135
message error, 745
message queue, 764
messages ready, 135
midpoint of thresholds, 327
MIL-STD 1553 serial decode, 436
MIL-STD 1553 triggering, 469
minimum duration, 493, 494, 497, 522
minimum vertical value measurement, 350
minimum vertical value, time of, 358, 720
mixed-signal oscilloscopes, 4
monochrome palette for image, 412
most significant byte first, 627
move, 266
move cursors, 721, 722
msbitr, 627
MSG (Message), 133, 135
MSO models, 4
MSS (Master Summary Status), 135
MTEenable (Mask Test Enable Register), 159
MTERegister[:EVEN] (Mask Test Event Register), 161, 774
MTEs commands, 359
multiple commands, 808
multiple queries, 55
multiply math function, 266, 273, 641
multiply math function as g(t) source, 269

O
obsolete and discontinued commands, 689
obsolete commands, 788
occurrence reported by magnitude, 723
offset, 266
OFFSet commands, 671
offset value for channel voltage, 226
offset value for selected function, 272, 275
one values in waveform data, 629
OPC (Operation Complete) command, 124
OPC (Operation Complete) status bit, 119, 121
OPEE (Operation Status Enable Register), 163
Open method, 49
operating configuration, 123, 453
operating state, 131
OPERation commands, 671
operation complete, 124
operation status condition register, 165
Operation Status Condition Register
:OPERRegister:CONDition, 165, 769
operation status conditions occurred, 135
Operation Status Enable Register (OPEE), 163
operation status event register, 167
Operation Status Event Register
:OPERRegister[:EVEN], 167, 768
operation, 266
operations for function, 273
OPERRegister:CONDition (Operation Status Condition Register), 165, 769
OPERRegister[:EVEN] (Operation Status Event Register), 167, 768
OPT (Option Identification), 125
optional syntax terms, 108
options, 125
order of digital channels on display, 702
order of output, 627
oscilloscope connection, opening, 49
oscilloscope connection, verifying, 39
oscilloscope external trigger, 254
oscilloscope models, 3
oscilloscope rate, 195
oscilloscope, connecting, 37
oscilloscope, initialization, 46
oscilloscope, operation, 4
oscilloscope, program structure, 46
oscilloscope, setting up, 37
oscilloscope, setup, 50
OUTPut commands, 671
output messages ready, 135
output queue, 124, 763
output queue clear, 117
output sequence, 627
output, mask test, 377
overlapped commands, 811
overload, 232, 261
Overload Enable Register (OVL), 169
Overload Event Register (:OVLRegister), 771
Overload Event Register (OVL), 771
overload frame count (CAN), 424
overload protection, 168, 171
overshoot of waveform, 325
overvoltage, 232, 261
OVL (Overload Event Enable Register), 169
OVL (Overload Event Register), 171
OVL bit, 156, 165, 187
OVLRegister (Overload Event Register), 771

P
P1080L24HZ, 594, 598
P1080L25HZ, 594, 598
P1080L50HZ, 598
P1080L60HZ, 571, 598
P480L60HZ, 594, 598
P720L60HZ, 594, 598
PAL, 594, 598
PALETTE commands, 671
palette for hardcopy, 289
palette for image, 412
PAL-M, 594, 598
parameters for delay measurement, 319
parametric measurements, 312
parity, 607
parity bits, LIN serial decode bus, 435
PARity commands, 671
parser, 141, 808
pass (mask test) output, 377
pass, self test, 137
path information, recall, 402
path information, save, 415
pattern, 477, 482, 484, 495, 548, 549, 550, 580, 588
pattern and edge, 477
PATern commands, 671
pattern data, I2S, 534
pattern data, LIN, 559
pattern duration, 493, 494, 522, 523
pattern format, I2S, 536
Index

pattern format, LIN, 562
pattern length, 483, 561
pattern trigger, 477
pattern triggering, 469
pattern width, 589
peak data, 623
peak detect, 196
peak detect acquisition type, 182, 623
peaks, 266
peak-to-peak vertical value measurement, 351
pending operations, 124
percent of waveform overshoot, 325
percent thresholds, 317
period measured to calculate phase, 328
period measurement, 47, 312, 327
persistence, waveform, 244, 251
phase measured between channels, 328
phase measurements, 342
pixel memory, 252
pixel memory, saving display to, 158
PLL Locked bit, 154, 165
pod, 393, 394, 395, 396, 641, 697
POD commands, 393
POD data format, 626
pod, stop displaying, 148
points, 188, 632, 634
POIints commands, 672
points in waveform data, 622
polarity, 596, 698
POLarity commands, 672
polarity for glitch trigger, 525
polling synchronization with timeout, 780
polling wait, 778
PON (Power On) status bit, 119, 121
portrait layout for hardcopy, 288
position, 241, 300, 459, 465
POSition commands, 672
position cursors, 721, 722
position in zoomed view, 465
position waveforms, 702
positive glitch trigger polarity, 525
positive pulse width, 330
positive pulse width measurement, 47
positive slope, 506, 585
positive slope, Nth edge in burst, 501
positive TV trigger polarity, 596
positive width, 330
preamble data, 636
preamble metadata, 621
precision analysis, 451
precision analysis record, 634
predefined logic threshold, 697
predefined threshold voltages, 742
present working directory, recall operations, 402
present working directory, save operations, 415
preset conditions, 128
preshoot measured on waveform, 329
previously stored configuration, 127
print command, 173
print job, start, 291
print mask test failures, 380
print query, 737
printer, 709
printer driver for hardcopy, 714
printer hardcopy format, 711
printing, 282
printing in grayscale, 712
probe, 504
probe attenuation affects channel voltage range, 233
probe attenuation factor (external trigger), 258
probe attenuation factor for selected channel, 227
PROBe commands, 673
probe head type, 228
probe ID, 228, 259
probe sense for oscilloscope, 700, 705
probe skew value, 230, 698
process sigma, mask test run, 383
program data, 790
program data syntax rules, 792
program initialization, 46
program message, 49, 116
program message syntax, 789
program message terminator, 790
program structure, 46
programming examples, 4, 813
protecting against calibration, 214
protection, 159, 171, 232, 261
PROTection commands, 673
protection lock, 452
pulse width, 324, 330
pulse width duration trigger, 522, 523, 527
pulse width measurement, 47, 312
pulse width trigger, 476
pulse width trigger level, 524
pulse width triggering, 469
PVD commands, 673
pwidth, 330
Q
qualifier, 527
QUALifier commands, 673
qualifier, trigger duration, 493, 494, 496
queries, multiple, 55
query error detected in Standard Event Status, 121
query responses, block data, 54
query responses, reading, 53
query results, reading into numeric variables, 54
query results, reading into string variables, 54
query return values, 810
query setup, 282, 296, 312, 453
query subsystem, 199, 237, 266
querying setup, 219
querying the subsystem, 470
queues, clearing, 775
quick reference, commands, 59
quoted ASCII string, 109
QYE (Query Error) status bit, 119, 121
R
range, 266, 466
RANGe commands, 673
range for channels, 233
range for duration trigger, 497
range for external trigger, 262
range for full-scale vertical axis, 274
range for glitch trigger, 527
range for time base, 460
range of offset values, 226
range qualifier, 526
range, 123, 537
ranges, value, 109
rate, 195
ratio of AC RMS values measured between channels, 352
raw acquisition record, 634
RCL (Recall), 127
read configuration, 123
read trace memory, 247
ReadIEEEBlock method, 49, 53, 55
ReadList method, 49, 53
ReadNumber method, 49, 53
readout, 717
ReadString method, 49, 53
real-time acquisition mode, 182, 187
recall, 127, 398, 453
RECAll commands, 398
recall filename, 399
recall image, 400
recall mask, 401
recall path information, 402
recall setup, 403
recalling and saving data, 244
receiver width, 123, 539
RECTangular window for transient signals, 280
reference, 266, 462
reference clock, 461
REFERence commands, 674
reference for time base, 739
reference signal (10 MHz), 189
reference signal mode, 461
registers, 120, 127, 131, 143, 152, 154, 156, 159, 161, 163, 165, 167, 169, 171
registers, clearing, 775
reject filter, 505
reject high frequency, 472
reject noise, 476
remote control examples, 813
Remote Terminal Address (RTA), M1553
trigger, 572
remove cursor information, 297
remove labels, 249
remove message from display, 448
reorder channels, 144
repetitive acquisitions, 174
report errors, 449
report transition, 342, 344
reporting status, 753
reporting the setup, 470
request service, 135
sweep mode, trigger
subsource, waveform source
subnet mask
sweep speed set to fast to measure fall
time, 334
switch disable, 450
switch, calibration protect, 214
sync break, LIN, 567
sync frame count (FlexRay), 431
syntax elements, 108
syntax rules, program data, 792
syntax, optional terms, 108
syntax, program message, 789
SYSTEM commands, 446
system commands, 447, 448, 449, 450, 453, 455
system commands introduction, 446
T
tdelta, 717
tedge, 342
telnet ports 5024 and 5025, 629
telnet sockets, 57
temporary message, 448
terminations, 448
termination conditions, mask test, 378
test sigma, mask test run, 383
test, self, 137
text, writing to display, 448
threshold, 243, 396, 697, 742
threshold commands, 680
threshold voltage (lower) for measurement, 715
timebase vernier
TIme commands
TImeAtMax measurement, 719
time-at-min measurement, 720
TImebase commands, 456
timebase vernier, 464
TImebase.MODE, 52
time-ordered label list, 250
timeout, 586
timer, 582
timing measurement, 312
title channels, 225
title, mask test, 392
tolerance, automask, 367, 368
top of waveform value measured, 355
TOTal commands, 680
total frame count (CAN), 426
total frame count (FlexRay), 432
total waveforms in mask test, 372
trace memories, how autoscale affects, 144
trace memory, 177, 180
trace memory data, 247
track measurements, 336
trademarks, 2
transfer instrument state, 123, 453
transmit, 247
transmit word size, I2S, 545
tree traversal rules, 808
tree, command, 793
TRG (Trigger), 133, 135, 136
TRIG OUT BNC, 211
trigger (external) input impedance, 257, 704
trigger armed event register, 165, 167
trigger burst, UART, 604
TRIGger CAN commands, 480
toggle channel source, 528, 597
TRIGger commands, 468, 680
TRIGger commands, general, 471
toggle trigger data, UART, 605
toggle duration, 493, 494
TRIGger DURATION commands, 492
TRIGger EBURSt commands, 498
trigger edge, 578
TRIGger EDGE commands, 502
trigger edge coupling, 503
trigger edge slope, 506
trigger event bit, 179
Trigger Event Register (TER), 762
TRIGger FLEXray commands, 508
TRIGger GLITch commands, 520
trigger holdoff, 473
TRIGger IIC commands, 529
trigger idle, UART, 606
TRIGger IIC commands, 547
trigger level constants, 227
trigger level voltage, 504
trigger level, 50%, 474
TRIGger LIN commands, 556
TRIGger M1553 commands, 569
trigger occurred, 135
trigger pattern, 477, 495
trigger qualifier, 496
trigger qualifier, UART, 609
trigger reset conditions, 128
Z

zero values in waveform data, 629
zoomed time base, 458
zoomed time base mode, how autoscale affects, 144
zoomed time base, measurement window, 356
zoomed window horizontal scale, 467
Index