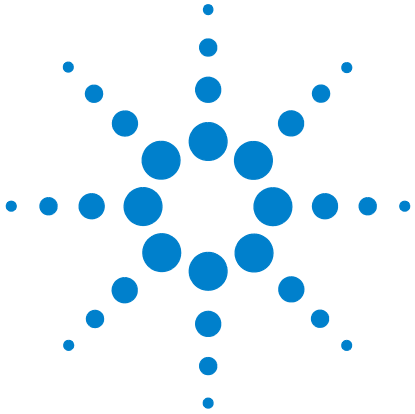


Infiniium

90000A Programmer's Reference

Agilent Technologies

Programmer's Reference

Publication Number 54913-97009
July 2010

This reference applies directly to software revision code 2.50 and later.

© Copyright Agilent Technologies 2010
All Rights Reserved.

90000A Series Infiniium Oscilloscopes

In This Book

This book is your guide to programming the Infiniium 90000A Series Oscilloscopes.

Chapters 1 through 5 give you an introduction to programming the oscilloscopes, along with necessary conceptual information. These chapters describe basic program communications, interface, syntax, data types, and status reporting.

Chapter 6 shows example BASIC and C programs, and describes chunks of one program to show you some typical applications. The BASIC and C example programs are also shipped on a disk with the oscilloscope.

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

The last chapter describes error messages.

Contents

1 Introduction to Programming
Communicating with the Oscilloscope 1-3
Output Command 1-4
Device Address 1-4
Instructions 1-4
Instruction Header 1-4
White Space (Separator) 1-5
Braces 1-5
Ellipsis 1-5
Square Brackets 1-5
Command and Query Sources 1-5
Program Data 1-6
Header Types 1-7
Duplicate Mnemonics 1-9
Query Headers 1-10
Program Header Options 1-11
Character Program Data 1-11
Numeric Program Data 1-12
Embedded Strings 1-13
Program Message Terminator 1-13
Common Commands within a Subsystem 1-14
Selecting Multiple Subsystems 1-14
Programming Getting Started 1-14
Initialization 1-15
Example Program using HP Basic 1-16
Using the DIGITIZE Command 1-17
Receiving Information from the Oscilloscope 1-19
String Variable Example 1-20
Numeric Variable Example 1-20
Definite-Length Block Response Data 1-21
Multiple Queries 1-22
Oscilloscope Status 1-22

2 Connectivity
LAN Interface Connector 2-3
GPIB Interface Connector 2-3
Default Startup Conditions 2-4
Interface Capabilities 2-5
GPIB Command and Data Concepts 2-6
Communicating Over the GPIB Interface 2-7
Communicating Over the LAN Interface 2-8
Communicating via Telnet and Sockets 2-10
Bus Commands 2-12

3 Message Communication and System Functions

Protocols 3-3

4 Status Reporting
Status Reporting Data Structures 4-5
Status Byte Register 4-8
Service Request Enable Register 4-10
Message Event Register 4-10
Trigger Event Register 4-10
Standard Event Status Register 4-11
Standard Event Status Enable Register 4-12
Operation Status Register 4-13
Operation Status Enable Register 4-14
Mask Test Event Register 4-15
Mask Test Event Enable Register 4-16
Acquisition Done Event Register 4-17
Process Done Event Register 4-17
Trigger Armed Event Register 4-17
Auto Trigger Event Register 4-17
Error Queue 4-18
Output Queue 4-18
Message Queue 4-19
Clearing Registers and Queues 4-19

5 Remote Acquisition Synchronization
Introduction 5-2
Programming Flow 5-2
Setting Up the Oscilloscope 5-2
Acquiring a Waveform 5-2
Retrieving Results 5-3
Acquisition Synchronization 5-3
Single Shot Device Under Test (DUT) 5-5
Averaging Acquisition Synchronization 5-6

6 Programming Conventions
Truncation Rule 6-3
The Command Tree 6-4
Infinity Representation 6-12
Sequential and Overlapped Commands 6-12
Response Generation 6-12
EOI 6-12

7 Sample Programs
Sample Program Structure 7-3
Sample C Programs 7-4
Listings of the Sample Programs 7-14
gpibdecl.h Sample Header 7-15

learnstr.c Sample Program 7-17
sicl_IO.c Sample Program 7-21
natl_IO.c Sample Program 7-26
init.bas Sample Program 7-30
lrn_str.bas Sample Program 7-38

8 Acquire Commands
AVERage 8-3
AVERage:COUNt 8-4
COMPlete 8-5
COMPlete:STATe 8-7
BANDwidth 8-8
INTerpolate 8-9
MODE 8-10
POINts 8-12
POINts:AUTO 8-16
RESPonse 8-17
SEGMented:COUNt 8-18
SEGMented:INDex 8-19
SEGMented:TTAGs 8-20
SRATe (Sample RATe) 8-21
SRATe Sample Rate Tables 8-23
SRATe:AUTO 8-24

9 Bus Commands
B1:TYPE 9-3

10 Calibration Commands
Oscilloscope Calibration 10-3
Probe Calibration 10-4

Calibration Commands 10-5
OUTPut 10-6
SKEW 10-7
STATus? 10-8

11 Channel Commands
BWLimit 11-4
COMMonmode 11-5
DIFFerential 11-6
DIFFerential:SKEW 11-7
DISPlay 11-8
DISPlay:AUTO 11-9
DISPlay:OFFSet 11-11
DISPlay:RANGe 11-12
DISPlay:SCALe 11-14

INPut 11-16
ISIM:APPLy 11-17
ISIM:BANDwidth 11-19
ISIM:BWLimit 11-21
ISIM:CONVolve 11-23
ISIM:DEConvolve 11-24
ISIM:DELay 11-25
ISIM:SPAN 11-27
ISIM:STATe 11-29
OFFSet 11-31
LABel 11-32
OFFSet 11-33
PROBe 11-34
PROBe:ATTenuation 11-36
PROBe:COUPling 11-37
PROBe:EADapter 11-38
PROBe:ECOupling 11-40
PROBe:EXTernal 11-42
PROBe:EXTernal:GAIN 11-43
PROBe:EXTernal:OFFSet 11-45
PROBe:EXTernal:UNITs 11-47
PROBe:HEAD:ADD 11-49
PROBe:HEAD:DELete ALL 11-50
PROBe:HEAD:SELect 11-51
PROBe:GAIN 11-52
PROBe:ID? 11-53
PROBe:SKEW 11-54
PROBe:STYPe 11-55
RANGe 11-56
SCALe 11-57
UNITs 11-58

12 Common Commands

*CLS (Clear Status) 12-4
*ESE (Event Status Enable) 12-5
*ESR? (Event Status Register) 12-7
*IDN? (Identification Number) 12-9
*LRN? (Learn) 12-10
*OPC (Operation Complete) 12-12
*OPT? (Option) 12-13
*PSC (Power-on Status Clear) 12-14
*RCL (Recall) 12-15
*RST (Reset) 12-16
*SAV (Save) 12-17
*SRE (Service Request Enable) 12-18

*STB? (Status Byte) 12-20
*TRG (Trigger) 12-22
*TST? (Test) 12-23
*WAI (Wait) 12-24

13 Disk Commands
CDIRectory 13-3
COPY 13-4
DELete 13-5
DIRectory? 13-6
LOAD 13-7
MDIRectory 13-8
PWD? 13-9
SAVE:IMAGe 13-10
SAVE:JITTer 13-11
SAVE:MEASurements 13-12
SAVE:SETup 13-13
SAVE:WAVeform 13-14
CSV and TSV Header Format 13-16
BIN Header Format 13-19
SEGMented 13-35

14 Display Commands
CGRade 14-3
CGRade:LEVels? 14-5
COLumn 14-7
CONNect 14-8
DATA? 14-9
GRATicule 14-10
LABel 14-12
LINE 14-13
PERSistence 14-14
ROW 14-15
SCOLor 14-16
STRing 14-19
TAB 14-20
TEXT 14-21

15 Function Commands
FUNCtion<N>? 15-4
ABSolute 15-5
ADD 15-6
AVERage 15-7
COMMonmode 15-8
DIFF (Differentiate) 15-9
DISPlay 15-10

DIVide 15-11
FFT:FREQuency 15-12
FFT:REFerence 15-13
FFT:RESolution? 15-14
FFT:WINDow 15-15
FFTMagnitude 15-17
FFTPhase 15-18
HIGHpass 15-19
HORizontal 15-20
HORizontal:POSition 15-21
HORizontal:RANGe 15-22
INTegrate 15-23
INVert 15-24
LOWPass 15-25
MAGNify 15-26
MAXimum 15-27
MINimum 15-28
MULTiply 15-29
OFFSet 15-30
RANGe 15-31
SMOoth 15-32
SQRT 15-33
SQUare 15-34
SUBTract 15-35
VERSus 15-36
VERTical 15-37
VERTical:OFFSet 15-38
VERTical:RANGe 15-39

16 Hardcopy Commands
AREA 16-3
DPRinter 16-4
FACTors 16-6
IMAGe 16-7
PRINters? 16-8

17 Histogram Commands
AXIS 17-4
MODE 17-5
SCALe:SIZE 17-6
WINDow:DEFault 17-7
WINDow:SOURce 17-8
WINDow:LLIMit 17-9
WINDow:RLIMit 17-10
WINDow:BLIMit 17-11
WINDow:TLIMit 17-12

18 InfiniiScan (ISCan) Commands
DELay 18-3
MEASurement:FAIL 18-4
MEASurement:LLIMit 18-5
MEASurement 18-6
MEASurement:ULIMit 18-7
MODE 18-8
NONMonotonic:EDGE 18-9
NONMonotonic:HYSTeresis 18-10
NONMonotonic:SOURce 18-11
RUNT:HYSTeresis 18-12
RUNT:LLEVel 18-13
RUNT:SOURce 18-14
RUNT:ULEVel 18-15
SERial:PATTern 18-16
SERial:SOURce 18-17
ZONE<N>:MODE 18-18
ZONE<N>:PLACement 18-19
ZONE:SOURce 18-20
ZONE<N>:STATe 18-21

19 Limit Test Commands
FAIL 19-3
LLIMit 19-4
MEASurement 19-5
RESults? 19-6
TEST 19-7
ULIMit 19-8

20 Marker Commands
CURSor? 20-3
MODE 20-4
X1Position 20-5
X2Position 20-6
X1Y1source 20-7
X2Y2source 20-9
XDELta? 20-11
Y1Position 20-12
Y2Position 20-13
YDELta? 20-14

21 Mask Test Commands
ALIGn 21-4
AlignFIT 21-5
AMASk:CREate 21-7
AMASk:SOURce 21-8

AMASk:SAVE | STORe 21-9
AMASk:UNITs 21-10
AMASk:XDELta 21-11
AMASk:YDELta 21-13
AUTO 21-15
AVERage 21-16
AVERage:COUNt 21-17
COUNt:FAILures? 21-18
COUNt:FUI? 21-19
COUNt:FWAVeforms? 21-20
COUNt:UI? 21-21
COUNt:WAVeforms? 21-22
DELete 21-23
ENABle 21-24
FOLDing 21-25
FOLDing:BITS 21-26
HAMPlitude 21-27
IMPedance 21-28
INVert 21-30
LAMPlitude 21-31
LOAD 21-32
NREGions? 21-33
PROBe:IMPedance? 21-34
RUMode 21-35
RUMode:SOFailure 21-37
SCALe:BIND 21-38
SCALe:X1 21-39
SCALe:XDELta 21-40
SCALe:Y1 21-41
SCALe:Y2 21-42
SOURce 21-43
STARt | STOP 21-44
STIMe 21-45
TITLe? 21-46
TRIGger:SOURce 21-47

22 Measure Commands
AREA 22-9
BWIDth 22-10
CDRRATE 22-11
CGRade:CROSsing 22-13
CGRade:DCDistortion 22-14
CGRade:EHEight 22-15
CGRade:EWIDth 22-16
CGRade:EWINdow 22-17

CGRade:JITTer 22-19
CGRade:QFACtor 22-20
CLEar 22-21
CLOCk 22-22
CLOCk:METHod 22-23
CLOCk:VERTical 22-26
CLOCk:VERTical:OFFSet 22-27
CLOCk:VERTical:RANGe 22-28
CROSsing 22-29
CTCDutycycle 22-30
CTCJitter 22-32
CTCNwidth 22-34
CTCPwidth 22-36
DATarate 22-38
DELTatime 22-41
DELTatime:DEFine 22-43
DUTYcycle 22-45
FALLtime 22-47
FFT:DFRequency 22-49
FFT:DMAGnitude 22-51
FFT:FREQuency 22-53
FFT:MAGNitude 22-55
FFT:PEAK1 22-57
FFT:PEAK2 22-58
FFT:THReshold 22-59
FREQuency 22-60
HISTogram:HITS 22-63
HISTogram:M1S 22-65
HISTogram:M2S 22-67
HISTogram:M3S 22-69
HISTogram:MAX? 22-71
HISTogram:MEAN? 22-72
HISTogram:MEDian? 22-73
HISTogram:MIN? 22-74
HISTogram:PEAK? 22-75
HISTogram:PP? 22-76
HISTogram:STDDev? 22-77
HOLDtime 22-78
JITTer:HISTogram 22-81
JITTer:MEASurement 22-82
JITTer:SPECtrum 22-83
JITTer:SPECtrum:HORizontal 22-84
JITTer:SPECtrum:HORizontal:POSition 22-85
JITTer:SPECtrum:HORizontal:RANGe 22-87
JITTer:SPECtrum:VERTical 22-88

JITTer:SPECtrum:VERTical:OFFSet 22-89
JITTer:SPECtrum:VERTical:RANGe 22-90
JITTer:SPECtrum:WINDow 22-91
JITTer:STATistics 22-92
JITTer:TRENd 22-93
JITTer:TRENd:SMOoth 22-94
JITTer:TRENd:SMOoth:POINts 22-95
JITTer:TRENd:VERTical 22-96
JITTer:TRENd:VERTical:OFFSet 22-97
JITTer:TRENd:VERTical:RANGe 22-98
NAME 22-99
NCJitter 22-100
NPERiod 22-102
NPULses 22-103
NUI 22-104
NWIDth 22-105
OVERshoot 22-107
PERiod 22-109
PHASe 22-111
PPULses 22-113
PREShoot 22-114
PWIDth 22-116
QUALifier<M>:CONDition 22-118
QUALifier<M>:SOURce 22-119
QUALifier<M>:STATe 22-120
RESults? 22-121
RISetime 22-124
RJDJ:ALL? 22-126
RJDJ:BANDwidth 22-128
RJDJ:BER 22-129
RJDJ:EDGE 22-131
RJDJ:INTerpolate 22-132
RJDJ:PLENgth 22-133
RJDJ:SOURce 22-135
RJDJ:STATe 22-137
RJDJ:TJRJDJ? 22-138
RJDJ:UNITs 22-139
SCRatch 22-140
SENDvalid 22-141
SETuptime 22-142
SLEWrate 22-145
SOURce 22-147
STATistics 22-149
TEDGe 22-150
THResholds:ABSolute 22-152

THResholds:HYSTeresis 22-154
THResholds:METHod 22-156
THResholds:PERCent 22-158
THResholds:TOPBase:METHod 22-160
THResholds:TOPBase:ABSolute 22-162
TIEClock2 22-164
TIEData 22-167
TIEFilter:STARt 22-169
TIEFilter:STATe 22-170
TIEFilter:STOP 22-171
TIEFilter:TYPE 22-172
TMAX 22-173
TMIN 22-175
TVOLt 22-177
UITouijitter 22-179
UNITinterval 22-180
VAMPlitude 22-182
VAVerage 22-184
VBASe 22-186
VLOWer 22-188
VMAX 22-190
VMIDdle 22-192
VMIN 22-194
VPP 22-196
VRMS 22-198
VTIMe 22-200
VTOP 22-202
VUPPer 22-204
WINdow 22-206

23 Root Level Commands
ADER? (Acquisition Done Event Register) 23-4
AER? (Arm Event Register) 23-5
ATER? (Auto Trigger Event Register) 23-6
AUToscale 23-7
AUToscale:CHANnels {ALL | DISPlayed} 23-8
AUToscale:PLACement {STACk | SEParate | OVERlay} 23-9
AUToscale:VERTical 23-10
BEEP 23-11
BLANk 23-12
CDISplay 23-13
DIGitize 23-14
MTEE 23-15
MTER? 23-16
MODel? 23-17

OPEE 23-18
OPER? 23-19
OVLRegister? 23-20
PDER? (Processing Done Event Register) 23-21
PRINt 23-22
RECall:SETup 23-23
RUN 23-24
SERial (Serial Number) 23-25
SINGle 23-26
STATus? 23-27
STOP 23-28
STORe:JITTer 23-29
STORe:SETup 23-30
STORe:WAVeform 23-31
TER? (Trigger Event Register) 23-32
VIEW 23-33

24 Self-Test Commands
CANCel 24-3
SCOPETEST 24-4

25 System Commands
DATE 25-3
DEBug 25-4
DSP 25-6
ERRor? 25-7
HEADer 25-8
LOCK 25-10
LONGform 25-11
PRESet 25-13
SETup 25-14
TIME 25-16

26 Time Base Commands
POSition 26-3
RANGe 26-4
REFClock 26-5
REFerence 26-6
SCALe 26-7
VIEW 26-8
WINDow:DELay 26-9
WINDow:POSition 26-11
WINDow:RANGe 26-12
WINDow:SCALe 26-13

27 Trigger Commands

Organization of Trigger Modes and Commands 27-4

Summary of Trigger Modes and Commands 27-5
AND:ENABle 27-6
AND:SOURce 27-7
COMM:BWIDth 27-8
COMM:ENCode 27-9
COMM:PATTern 27-10
COMM:POLarity 27-11
COMM:SOURce 27-12
DELay:ARM:SOURce 27-13
DELay:ARM:SLOPe 27-14
DELay:EDELay:COUNt 27-15
DELay:EDELay:SOURce 27-16
DELay:EDELay:SLOPe 27-17
DELay:MODE 27-18
DELay:TDELay:TIME 27-19
DELay:TRIGger:SOURce 27-20
DELay:TRIGger:SLOPe 27-21
EDGE:SLOPe 27-22
EDGE:SOURce 27-23
GLITch:POLarity 27-24
GLITch:SOURce 27-25
GLITch:WIDTh 27-26
HOLDoff 27-27
HTHReshold 27-28
HYSTeresis 27-29
LEVel 27-30
LTHReshold 27-31
MODE 27-32
PATTern:CONDition 27-35
PATTern:LOGic 27-36
PWIDth:DIRection 27-37
PWIDth:POLarity 27-38
PWIDth:SOURce 27-39
PWIDth:TPOint 27-40
PWIDth:WIDTh 27-41
RUNT:POLarity 27-42
RUNT:QUALified 27-43
RUNT:SOURce 27-44
RUNT:TIME 27-45
SEQuence:TERM1 27-46
SEQuence:TERM2 27-46
SEQuence:RESet:ENABle 27-47
SEQuence:RESet:TYPE 27-48

SEQuence:RESet:EVENt 27-49
SEQuence:RESet:TIME 27-50
SEQuence:RESet:WAIT:ENABle 27-51
SEQuence:RESet:WAIT:TIME 27-52
SHOLd:CSOurce 27-53
SHOLd:CSOurce:EDGE 27-55
SHOLd:DSOurce 27-56
SHOLd:HoldTIMe (HTIMe) 27-57
SHOLd:MODE 27-58
SHOLd:SetupTIMe 27-59
STATe:CLOCk 27-60
STATe:LOGic 27-61
STATe:LTYPe 27-62
STATe:SLOPe 27-63
SWEep 27-64
TIMeout:CONDition 27-65
TIMeout:SOURce 27-66
TIMeout:TIME 27-67
TRANsition:DIRection 27-68
TRANsition:SOURce 27-69
TRANsition:TIME 27-70
TRANsition:TYPE 27-71
TV:LINE 27-72
TV:MODE 27-73
TV:POLarity 27-74
TV:SOURce 27-75
TV:STANdard 27-76
TV:UDTV:ENUMber 27-77
TV:UDTV:HSYNc 27-78
TV:UDTV:HTIMe 27-79
TV:UDTV:PGTHan 27-80
TV:UDTV:POLarity 27-81
WINDow:CONDition 27-82
WINDow:SOURce 27-83
WINDow:TIME 27-84
WINDow:TPOint 27-85

28 Waveform Commands
BANDpass? 28-6
BYTeorder 28-7
COMPlete? 28-8
COUNt? 28-9
COUPling? 28-10
DATA? 28-11
FORMat 28-32

POINts? 28-35
PREamble 28-36
SEGMented:ALL 28-42
SEGMented:COUNt? 28-43
SEGMented:TTAG? 28-44
SEGMented:XLISt? 28-45
SOURce 28-46
STReaming 28-47
TYPE? 28-48
VIEW 28-49
XDISplay? 28-51
XINCrement? 28-52
XORigin? 28-53
XRANge? 28-54
XREFerence? 28-55
XUNits? 28-56
YDISplay? 28-57
YINCrement? 28-58
YORigin? 28-59
YRANge? 28-60
YREFerence? 28-61
YUNits? 28-62

29 Waveform Memory Commands
CLEar 29-3
DISPlay 29-4
LOAD 29-5
SAVE 29-6
XOFFset 29-7
XRANge 29-8
YOFFset 29-9
YRANge 29-10

30 Serial Data Equalization Commands
CTLequalizer:DISPlay 30-4
CTLequalizer:SOURce 30-5
CTLequalizer:DCGain 30-6
CTLequalizer:P1 30-7
CTLequalizer:P2 30-8
CTLequalizer:RATe 30-9
CTLequalizer:VERTical 30-10
CTLequalizer:VERTical:OFFSet 30-11
CTLequalizer:VERTical:RANGe 30-12
CTLequalizer:ZERo 30-13
FFEQualizer:DISPlay 30-14
FFEQualizer:SOURce 30-15

FFEQualizer:NPRecursor 30-16
FFEQualizer:NTAPs 30-17
FFEequalizer:RATe 30-18
FFEQualizer:TAP 30-19
FFEQualizer:TAP:PLENgth 30-20
FFEQualizer:TAP:WIDTh 30-21
FFEQualizer:TAP:DELay 30-22
FFEQualizer:TAP:AUTomatic 30-23
FFEQualizer:TAP :BANDwidth 30-24
FFEQualizer:TAP :BWMode 30-25
FFEQualizer:TAP :TDELay 30-26
FFEQualizer:TAP :TDMode 30-27
FFEQualizer:VERTical 30-28
FFEQualizer:VERTical:OFFSet 30-29
FFEQualizer:VERTical:RANGe 30-30
DFEQualizer:STATe 30-31
DFEQualizer:SOURce 30-32
DFEQualizer:NTAPs 30-33
DFEQualizer:TAP 30-34
DFEQualizer:TAP:WIDTh 30-35
DFEQualizer:TAP:DELay 30-36
DFEQualizer:TAP:MAX 30-37
DFEQualizer:TAP:MIN 30-38
DFEQualizer:TAP:GAIN 30-39
DFEQualizer:TAP:UTARget 30-40
DFEQualizer:TAP:LTARget 30-41
DFEQualizer:TAP:AUTomatic 30-42

31 Error Messages
Error Queue 31-3
Error Numbers 31-4
Command Error 31-5
Execution Error 31-6
Device- or Oscilloscope-Specific Error 31-7
Query Error 31-8
List of Error Messages 31-9

1

Introduction to Programming

Introduction to Programming

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

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

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

You can accomplish other tasks by combining these functions.

Example Programs are Written in HP BASIC and C

The programming examples for individual commands in this manual are written in HP BASIC and C.

Communicating with the Oscilloscope

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

For example, HP BASIC uses the OUTPUT statement for sending commands and queries. After a query is sent, the response is usually read using the HP BASIC ENTER statement. The ENTER statement passes the value across the bus to the computer and places it in the designated variable.

For the GPIB interface, messages are placed on the bus using an output command and passing the device address, program message, and a terminator. Passing the device address ensures that the program message is sent to the correct GPIB interface and GPIB device.

The following HP BASIC OUTPUT statement sends a command that sets the channel 1 scale value to 500 mV:

```
OUTPUT <device address> ;":CHANNEL1:SCALE 500E-
3"<terminator>
```

The device address represents the address of the device being programmed. Each of the other parts of the above statement are explained on the following pages.

Use the Suffix Multiplier Instead

Using "mV" or "V" following the numeric voltage value in some commands will cause Error 138 - Suffix not allowed. Instead, use the convention for the suffix multiplier as described in chapter 3, "Message Communication and System Functions."

Output Command

The output command depends entirely on the programming language. Throughout this book, HP BASIC and ANSI C are used in the examples of individual commands. If you are using other languages, you will need to find the equivalents of HP BASIC commands like OUTPUT, ENTER, and CLEAR, to convert the examples.

Device Address

The location where the device address must be specified depends on the programming language you are using. In some languages, it may be specified outside the OUTPUT command. In HP BASIC, it is always specified after the keyword, OUTPUT. The examples in this manual assume that the oscilloscope and interface card are at GPIB device address 707. When writing programs, the device address varies according to how the bus is configured.

Instructions

Instructions, both commands and queries, normally appear as strings embedded in a statement of your host language, such as BASIC, Pascal, or 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 HP BASIC's "learnstring" command. There are only a few instructions that use block data.

Instructions are composed of two main parts:

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

Instruction Header

The instruction header is one or more command mnemonics separated by colons (:). They represent the operation to be performed by the oscilloscope. See the "Programming Conventions" chapter for more information.

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

White Space (Separator)

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32 in decimal.

Braces

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

Ellipsis

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

Square Brackets

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

Command and Query Sources

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

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

Program Data

Program data is used to clarify the meaning of the command or query. It provides necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data and the values they accept.

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

Header Types

There are three types of headers:

- Simple Command headers
- Compound Command headers
- Common Command headers

Simple Command Header

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

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

or

```
OUTPUT 707;":AUTOSCALE"
```

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

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

or

```
OUTPUT 707;":DIGITIZE CHANNEL1,FUNCTION2"
```

Compound Command Header

Compound command headers are a combination of two program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example:

```
OUTPUT 707;":CHANNEL1:BWLIMIT ON"
```

Combining Commands in the Same Subsystem

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

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

For example:

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

Common Command Header

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

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Duplicate Mnemonics

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

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

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

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

```
:TIMEBASE:RANGE  1
```

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

Query Headers

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

```
:TIMEBASE:RANGE?
```

places the current time base setting in the output queue.

In HP BASIC, the computer input statement:

```
ENTER < device address > ;Range
```

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

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

```
:MEASURE:RISETIME?
```

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

The output queue must be read before the next program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow it with an input statement. In HP BASIC, this is usually done with an ENTER statement immediately followed by a variable name. This statement reads the result of the query and places the result in a specified variable.

Handle Queries Properly

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

Program Header Options

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

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

:TIMEBASE:DELAY 1E-6 is the long form.

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

Using Long Form or Short Form

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

The rules for the short form syntax are described in the chapter, "Programming Conventions."

Character Program Data

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

The available mnemonics for character program data are always included with the instruction's syntax definition. You may send either the long form of commands, or the short form (if one exists). You may mix uppercase and lowercase letters freely. When receiving responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEBASE:RANGE requires the desired full-scale range to be expressed numerically.

For numeric program data, you can use exponential notation or suffix multipliers to indicate the numeric value. The following numbers are all equal:

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

When a syntax definition specifies that a number is an integer, it means that the number should be whole. Any fractional part is ignored and truncated. Numeric data parameters that accept fractional values are called real numbers. For more information see the chapter, “Interface Functions.”

All numbers are expected to be strings of ASCII characters.

- When sending the number 9, you would send a byte representing the ASCII code for the character “9” (which is 57).
- A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). The number of bytes is figured automatically when you include the entire instruction in a string.

Embedded Strings

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

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

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

Program Message Terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the GPIB interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

New Line Terminator Functions Like EOS and EOT

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

Common Commands within a Subsystem

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

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

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

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

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

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

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon lets you enter a new subsystem. For example:

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

:CHANNEL1:RANGE 0.4;:TIMEBASE:RANGE 1

You can Combine Compound and Simple Commands

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

Programming Getting Started

The remainder of this chapter explains how to set up the oscilloscope, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the computer. The chapter, “Measure Commands” describes sending measurement data to the oscilloscope.

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. For example, HP BASIC provides a CLEAR command which clears the interface buffer:

```
CLEAR 707  ! initializes the interface of the oscilloscope
```

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

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

```
OUTPUT 707;"*RST"  ! initializes the oscilloscope to a preset
state
```

Initializing the Oscilloscope

The commands and syntax for initializing the oscilloscope are discussed in the chapter, "Common Commands." Refer to your GPIB manual and programming language reference manual for information on initializing the interface.

Autoscale

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

The syntax for the autoscale function is:

```
:AUTOSCALE<terminator>
```

Setting Up the Oscilloscope

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

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

```
:CHANNEL1:PROBE 10; RANGE 16;OFFSET 1.00<terminator>
:SYSTEM:HEADER OFF<terminator>
:TIMEBASE:RANGE 1E-3;DELAY 100E-6<terminator>
```

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

Example Program using HP Basic

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

```
10    CLEAR 707! Initialize oscilloscope interface
20    OUTPUT 707;"*RST"!Initialize oscilloscope to preset state
30    OUTPUT 707;":TIMEBASE:RANGE 5E-4"! Time base to 500 us full scale
40    OUTPUT 707;":TIMEBASE:DELAY 0"! Delay to zero
50    OUTPUT 707;":TIMEBASE:REFERENCE CENTER"! Display reference at center
60    OUTPUT 707;":CHANNEL1:PROBE 10"! Probe attenuation to 10:1
70    OUTPUT 707;":CHANNEL1:RANGE 1.6"! Vertical range to 1.6 V full scale
80    OUTPUT 707;":CHANNEL1:OFFSET -.4"! Offset to -0.4
90    OUTPUT 707;":CHANNEL1:INPUT DC"! Coupling to DC
100   OUTPUT 707;":TRIGGER:MODE EDGE"! Edge triggering
110   OUTPUT 707;":TRIGGER:LEVEL CHAN1,-.4"! Trigger level to -0.4
120   OUTPUT 707;":TRIGGER:SLOPE POSITIVE"! Trigger on positive slope
125   OUTPUT 707;":SYSTEM:HEADER OFF<terminator>
130   OUTPUT 707;":ACQUIRE:MODE RTIME"! Normal acquisition
140   OUTPUT 707;":DISPLAY:GRATICULE FRAME"! Grid off
150   END
```

Overview of the Program

- Line 10 initializes the oscilloscope interface to a known state.
- Line 20 initializes the oscilloscope to a preset state.
- Lines 30 through 50 set the time base, the horizontal time at 500 µs full scale, and 0 s of delay referenced at the center of the graticule.
- Lines 60 through 90 set 10:1 probe attenuation, set the vertical range to 1.6 volts full scale, center screen at –0.4 volts, and select DC 1 Mohm impedance coupling.
- Lines 100 through 120 configure the oscilloscope to trigger at –0.4 volts with positive edge triggering.
- Line 125 turns system headers off.
- Line 130 configures the oscilloscope for real time acquisition.
- Line 140 turns the grid off.

Using the DIGITIZE Command

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

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

You can send the DIGITIZE command with no parameters for a higher throughput. Refer to the DIGITIZE command in the chapter, “Root Level Commands” for details.

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

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

```
OUTPUT 707;":SYSTEM:HEADER OFF<terminator>
OUTPUT 707;":ACQUIRE:MODE RTIME"<terminator>
OUTPUT 707;":ACQUIRE:COMPLETE 100"<terminator>
OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:FORMAT BYTE"<terminator>
OUTPUT 707;":ACQUIRE:COUNT 8"<terminator>
OUTPUT 707;":ACQUIRE:POINTS 500"<terminator>
OUTPUT 707;":DIGITIZE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:DATA?"<terminator>
```

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

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

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

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

When using GPIB, you may abort a digitize operation by sending a Device Clear over the bus (for example, CLEAR 707).

Receiving Information from the Oscilloscope

After receiving a query (a command header followed by a question mark), the oscilloscope places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the interface to the computer. The input statement for receiving a response message from an oscilloscope's output queue typically has two parameters; the device address and a format specification for handling the response message. For example, to read the result of the query command :CHANNEL1:INPUT? you would execute the HP BASIC statement:

```
ENTER <device address> ;Setting$
```

This would enter the current setting for the channel 1 coupling in the string variable Setting$. The device address parameter represents the address of the oscilloscope.

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

Handle Queries Properly

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

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

String Variable Example

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

For the example programs, assume that the device being programmed is at device address 707. The actual address depends on how you have configured the bus for your own application.

In HP BASIC 5.0, string variables are case-sensitive, and must be expressed exactly the same each time they are used. This example shows the data being returned to a string variable:

```
10 DIM Rang$[30]
20 OUTPUT 707;":CHANNEL1:RANGE?"
30 ENTER 707;Rang$
40 PRINT Rang$
50 END
```

After running this program, the computer displays:

+8.00000E-01

Numeric Variable Example

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

```
10 OUTPUT 707;":CHANNEL1:RANGE?"
20 ENTER 707;Rang
30 PRINT Rang
40 END
```

After running this program, the computer displays:

.8

Definite-Length Block Response Data

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

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

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

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

Multiple Queries

You can send multiple queries to the oscilloscope within a single program message, but you must also read them back within a single program message. This can be accomplished by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :TIMEBASE:RANGE?;DELAY? into the string variable Results$ with the command:

```
ENTER 707;Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :TIMEBASE:RANGE?;DELAY? would be:

```
<range_value>;<delay_value>
```

Use the following program message to read the query :TIMEBASE:RANGE?;DELAY? into multiple numeric variables:

```
ENTER 707;Result1,Result2
```

Oscilloscope Status

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

2

Connectivity

LAN, USB, and GPIB Interfaces

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

LAN Interface Connector

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

GPIB Interface Connector

The oscilloscope is equipped with a GPIB interface connector on the rear panel. This allows direct connection to a GPIB equipped computer. You can connect an external GPIB compatible device to the oscilloscope by installing a GPIB cable between the two units. Finger tighten the captive screws on both ends of the GPIB cable to avoid accidentally disconnecting the cable during operation.

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

C A U T I O N Avoid stacking more than three or four cables on any one connector. Multiple connectors produce leverage that can damage a connector mounting.

Default Startup Conditions

The following default conditions are established during power-up:

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

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

Interface Capabilities

The interface capabilities of this oscilloscope, as defined by IEEE 488.1 and IEEE 488.2, are listed in Table 2-1.

Table 2-1 Interface Capabilities

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

GPIB Command and Data Concepts

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

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

Communicating Over the GPIB Interface

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

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

The Oscilloscope is at Address 707 for Programming Examples

The programming examples in this manual assume that the oscilloscope is at device address 707.

Interface Select Code

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

Oscilloscope Address

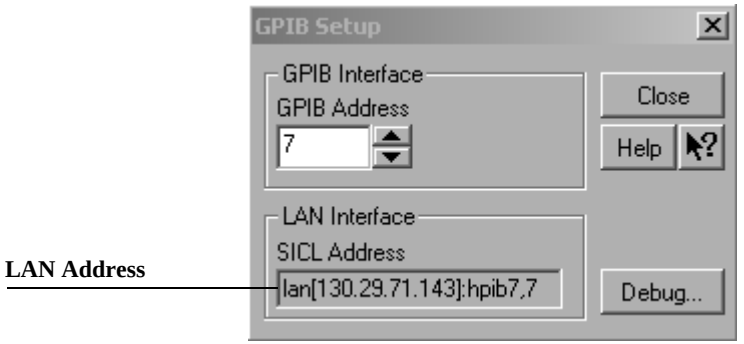
Each oscilloscope on the GPIB must have a unique oscilloscope address between decimal 0 and 30. This oscilloscope address is used by the computer to direct commands and communications to the proper oscilloscope on an interface. The default is typically "7" for this oscilloscope. You can change the oscilloscope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

Do Not Use Address 21 for an Oscilloscope Address

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

Communicating Over the LAN Interface

The device address used to send commands and receive data using the LAN interface is located in the GPIB Setup dialog box as shown below.



GPIB Setup Dialog Box

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

```
#include <sicl.h>

#define BUFFER_SIZE 1024

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

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

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

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

Communicating via Telnet and Sockets

Telnet

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

```
telnet Oscilloscope_IP_Address 5024
```

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

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

Sockets

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

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

Some basic commands used in communicating to your oscilloscope include:

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

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

```
#include <winsock2.h>

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

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

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

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

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

   //Do some work here

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

Bus Commands

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

Device Clear

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

Group Execute Trigger

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

Interface Clear

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

3

Message Communication and System Functions

Message Communication and System Functions

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

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

Protocols

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

Functional Elements

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

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

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

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

Protocol Overview

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

A program message is sent by the computer to the oscilloscope, and a response message is sent from the oscilloscope to the computer in response to a query message. A query message is defined as being a program message that contains one or more queries. The oscilloscope will only talk when it has received a valid query message, and therefore has something to say. The computer should only attempt to read a response after sending a complete query message, but before sending another program message.

Remember this Rule of Oscilloscope Communication

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

Protocol Operation

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

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

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

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

Commands are executed in the order they are received.

Protocol Exceptions

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

Suffix Multiplier

The suffix multipliers that the oscilloscope will accept are shown in Table 3-1.

Table 3-1 **<suffix mult>**

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

Suffix Unit

The suffix units that the oscilloscope will accept are shown in Table 3-2.

Table 3-2 **<suffix unit>**

Suffix	Referenced Unit
V	Volt
S	Second

4

Status Reporting

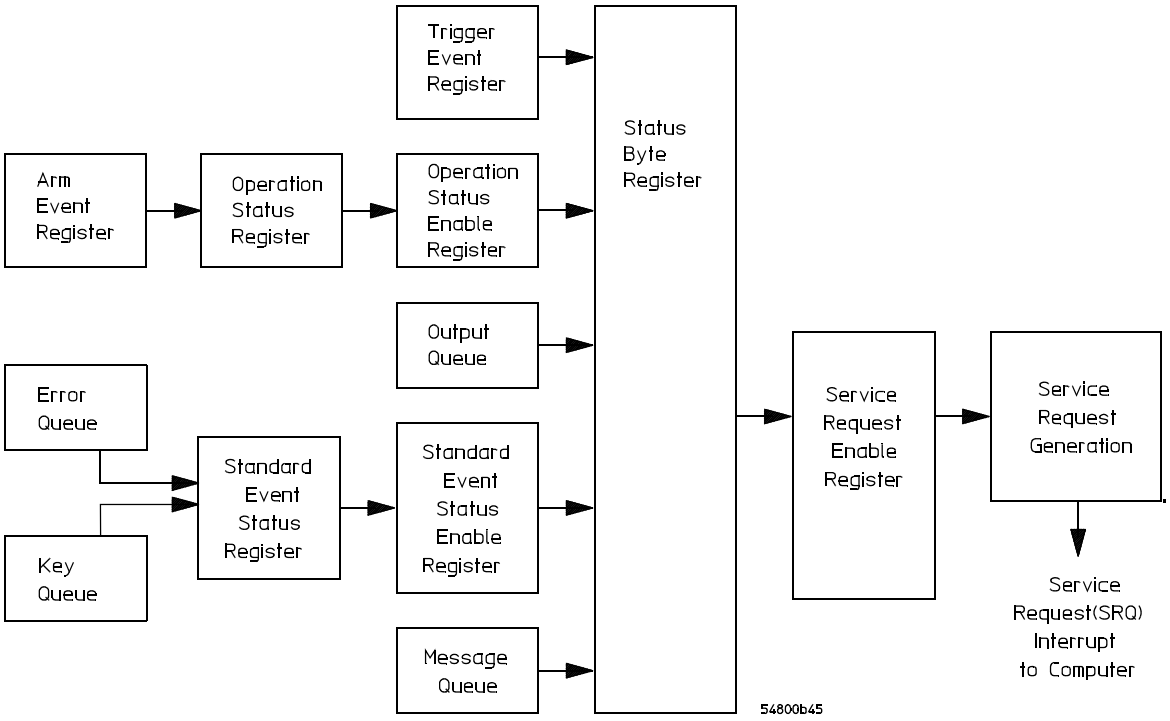
Status Reporting

An overview of the oscilloscope's status reporting structure is shown in Figure 4-1. The status reporting structure shows you how to monitor specific events in the oscilloscope. Monitoring these events lets you determine the status of an operation, the availability and reliability of the measured data, and more.

- To monitor an event, first clear the event, then enable the event. All of the events are cleared when you initialize the oscilloscope.
- To generate a service request (SRQ) interrupt to an external computer, enable at least one bit in the Status Byte Register.

The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987. IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting. There are also oscilloscope-defined structures and bits.

Figure 4-1



Status Reporting Overview Block Diagram

The status reporting structure consists of the registers shown here.

Table 4-1 lists the bit definitions for each bit in the status reporting data structure.

Table 4-1 **Status Reporting Bit Definition**

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

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

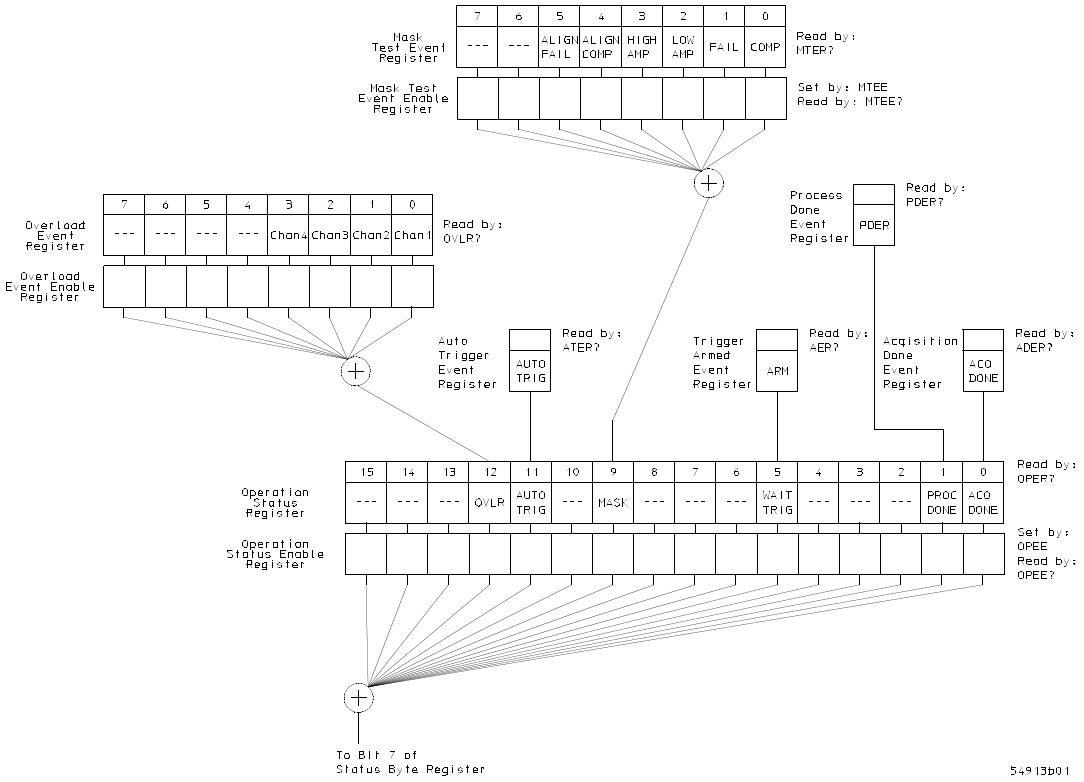
Status Reporting Data Structures

The different status reporting data structures, descriptions, and interactions are shown in Figure 4-2. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, you must enable the corresponding bits. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to the computer, you must enable at least one bit in the Status Byte Register. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

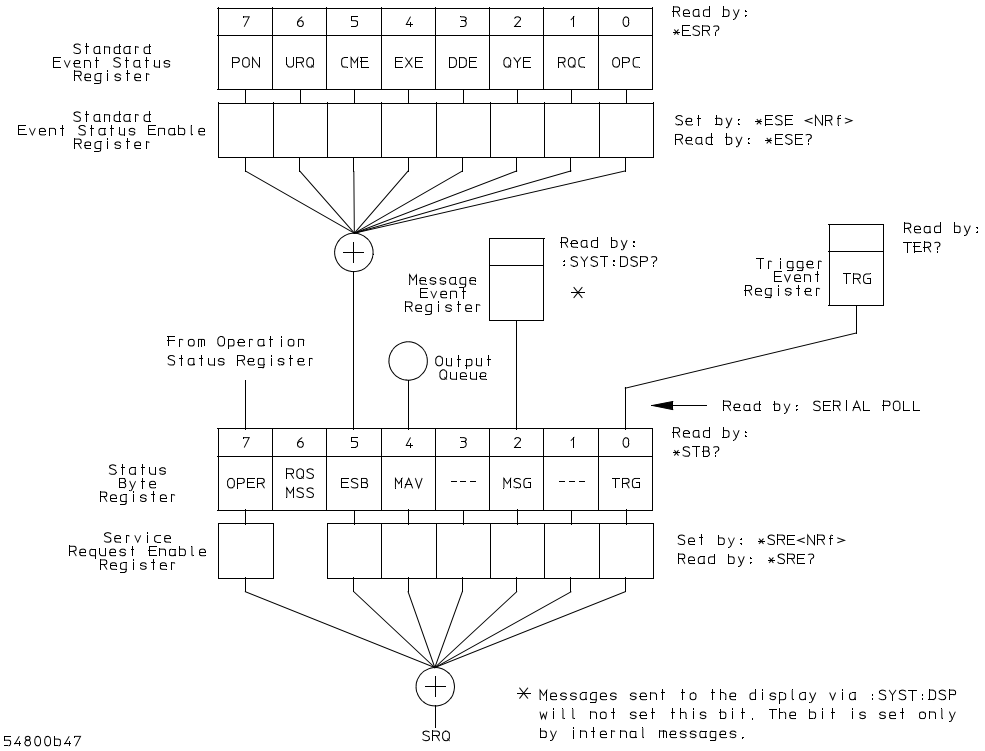
For more information about common commands, see the “Common Commands” chapter.

Figure 4-2



Status Reporting Data Structures

Figure 4-2 (Continued)



Status Reporting Data Structures (Continued)

Status Byte Register

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

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

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

The only other bit in the Status Byte Register affected by the *STB? query is the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? query.

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

Example 1

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF;*STB?"     !Turn headers off
20  ENTER 707;Result      !Place result in a numeric variable
30  PRINT Result      !Print the result
40  End
```

The next program prints 132 and clears bit 6 (RQS) of the Status Byte Register. The difference in the decimal value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set, and is cleared when the Status Byte Register is read by the serial poll command.

Example 2

This example uses the HP BASIC serial poll (SPOLL) command to read the contents of the oscilloscope's Status Byte Register.

```
10  Result = SPOLL(707)
20  PRINT Result
30  END
```

Use Serial Polling to Read the Status Byte Register

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

Service Request Enable Register

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

Bits are set in the Service Request Enable Register using the *SRE command, and the bits that are set are read with the *SRE? query. Bit 6 always returns 0. Refer to the Status Reporting Data Structures shown in Figure 4-2.

Example

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

```
OUTPUT 707;"*SRE 48"
```

This example uses the parameter “48” to allow the oscilloscope to generate an SRQ interrupt under the following conditions:

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

Message Event Register

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

Trigger Event Register

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

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

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

Standard Event Status Register

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

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

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

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

Example

This example uses the *ESR? query to read the contents of the Standard Event Status Register.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"      !Turn headers off
20  OUTPUT 707;"*ESR?"
30  ENTER 707;Result      !Place result in a numeric variable
40  PRINT Result      !Print the result
50  End
```

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

Standard Event Status Enable Register

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

Example

Suppose your application requires an interrupt whenever any type of error occurs. The error status bits in the Standard Event Status Register are bits 2 through 5. The sum of the decimal weights of these bits is 60. Therefore, you can enable any of these bits to generate the summary bit by sending:

```
OUTPUT 707;"*ESE 60"
```

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

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

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

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

Operation Status Register

This register hosts the following bits:

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

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

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

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

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

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

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

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

Operation Status Enable Register

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

Example

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

```
OUTPUT 707;"OPEE 512" ( hex 200 )
```

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

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

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

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

Mask Test Event Register

This register hosts the following bits:

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

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

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

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

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

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

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

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

Mask Test Event Enable Register

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

Example

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

```
OUTPUT 707;"MTEE 2"
```

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

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

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

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

Acquisition Done Event Register

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

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

Process Done Event Register

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

Trigger Armed Event Register

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

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

Auto Trigger Event Register

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

Error Queue

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

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

The error queue is cleared when any of these events occur:

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

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

Output Queue

The output queue stores the oscilloscope-to-computer responses that are generated by certain oscilloscope commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register. You may read the output queue with the HP Basic ENTER statement.

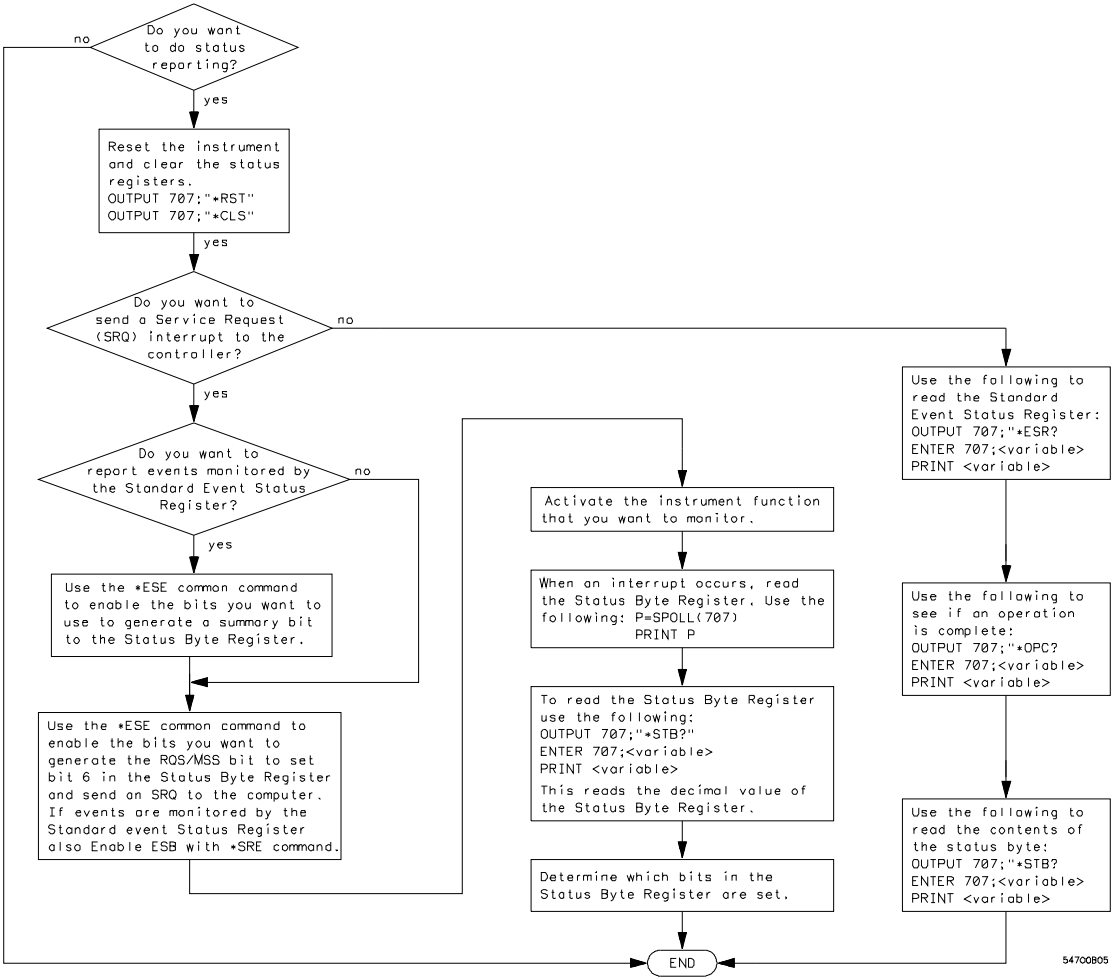
Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The queue is read with the :SYSTEM:DSP? query. Note that messages sent with the :SYSTEM:DSP command do not set the MSG status bit in the Status Byte Register.

Clearing Registers and Queues

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

Figure 4-3



Status Reporting Decision Chart

5

Remote Acquisition Synchronization

Introduction

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

Programming Flow

Most remote programming follows these three general steps:

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

Setting Up the Oscilloscope

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

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

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

Acquiring a Waveform

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

Table 0-1

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

Retrieving Results

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

Acquisition Synchronization

Blocking Synchronization

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

Example

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

//Acquire
:DIG

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

Polling Synchronization With Timeout

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

Example

```
TIMEOUT = 1000ms
currentTime = 0ms

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

// Acquire
:SINGLE

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

```
}

//Get results
if (currentTime < TIMEOUT)
{
   :MEASURE:RISETIME?
}
```

Single Shot Device Under Test (DUT)

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

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

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

Example

```
TIMEOUT = 1000ms
currentTime = 0ms
```

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

// Acquire
:SINGLE

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

//oscilloscope is armed and ready, enable DUT here

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

//Get results
if (currentTime < TIMEOUT)
{
   :MEASURE:RISETIME?
}
```

Averaging Acquisition Synchronization

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

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

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

Example

```
AVERAGE_COUNT = 256

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

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

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

:STOP;*OPC?

// Get results
```

6

Programming Conventions

Programming Conventions

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

Truncation Rule

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

Command Truncation Rule

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

Table 5-1 shows how the truncation rule is applied to commands.

Table 6-1 Mnemonic Truncation

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

The Command Tree

The command tree in Figure 5-1 shows all of the commands in the Infiniium-Series Oscilloscopes and the relationship of the commands to each other. The IEEE 488.2 common commands are not 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 oscilloscope, the parser is set to the "root" of the command tree.

Command Types

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

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

Tree Traversal Rules

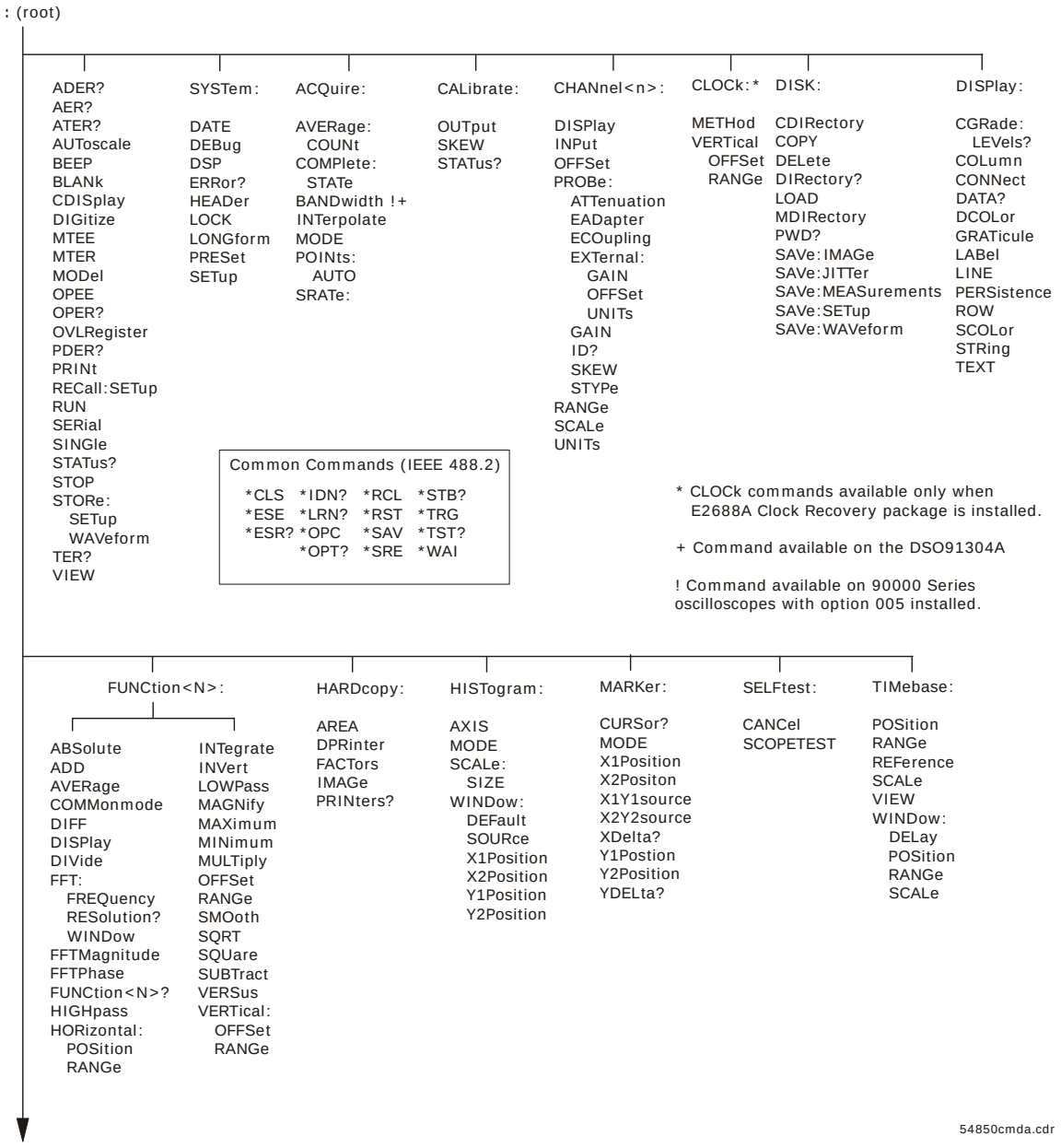
Command headers are created by traversing down the command tree. A legal command header from the command tree would be :TIMEBASE:RANGE. This is referred to as a compound header. A compound header is a header made up of two or more mnemonics separated by colons. The compound header contains no spaces.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

Tree Traversal Rules

A leading colon or a program message terminator (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command places the oscilloscope in that subsystem until a leading colon or a program message terminator is found.

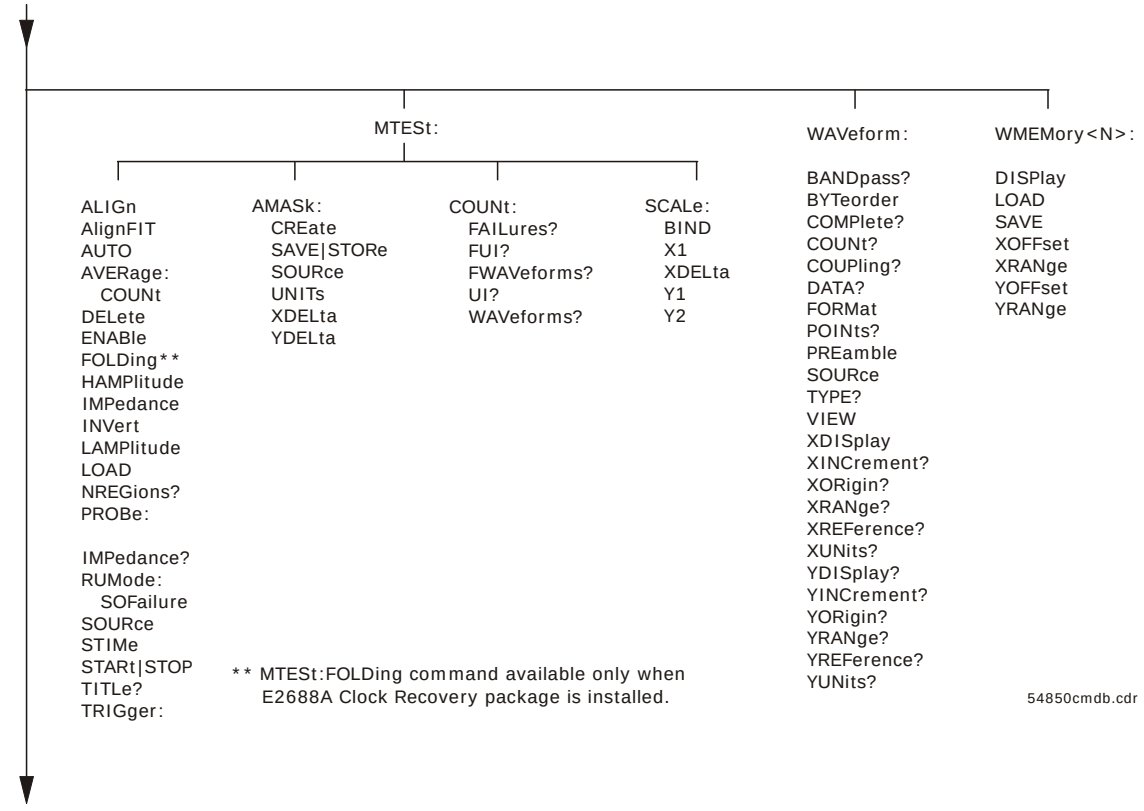
In the command tree, use the last mnemonic in the compound header as a reference point (for example, RANGE). Then find the last colon above that mnemonic (TIMEBASE:). That is the point where the parser resides. You can send any command below this point within the current program message without sending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

Figure 5-1



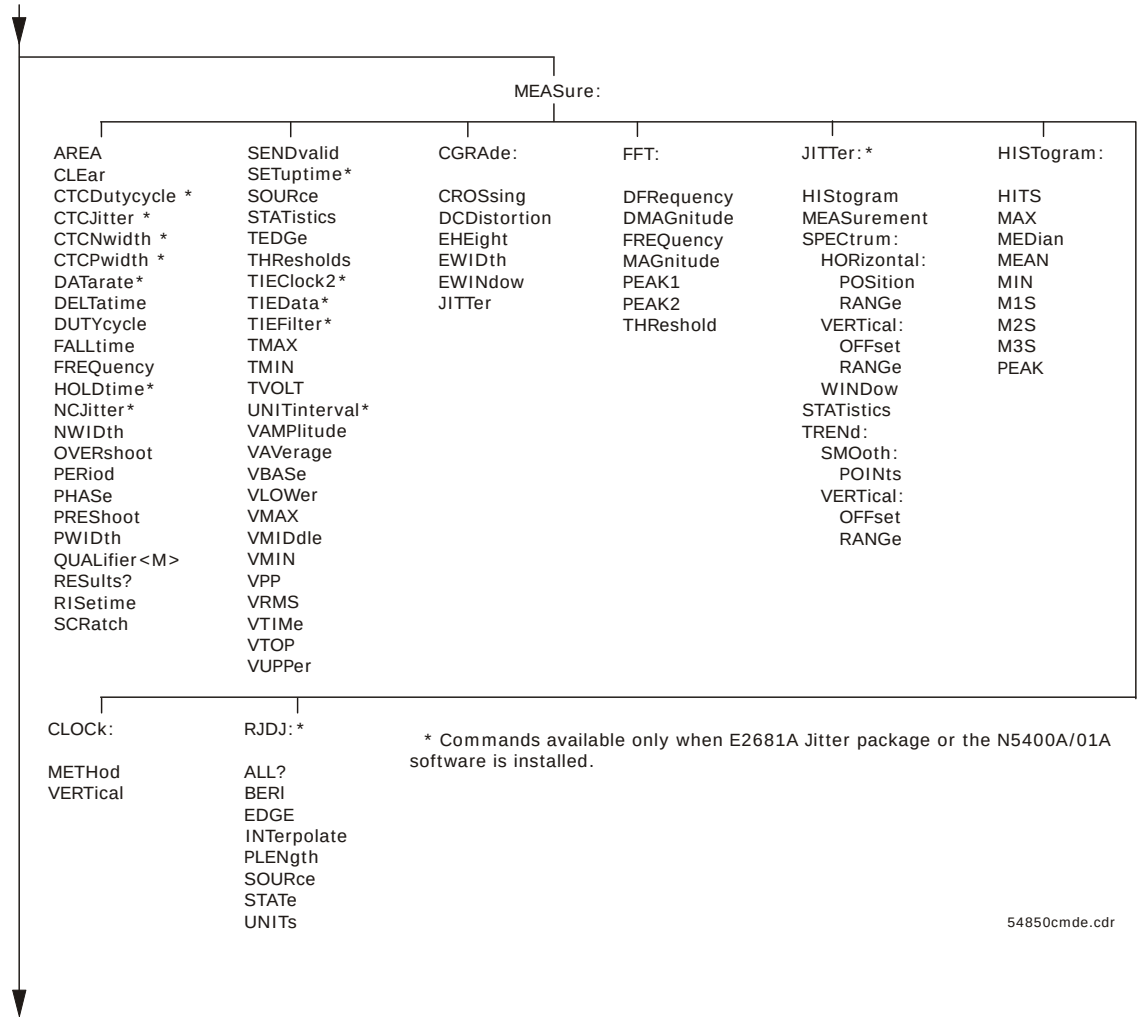
Command Tree

Figure 5-1 (continued)



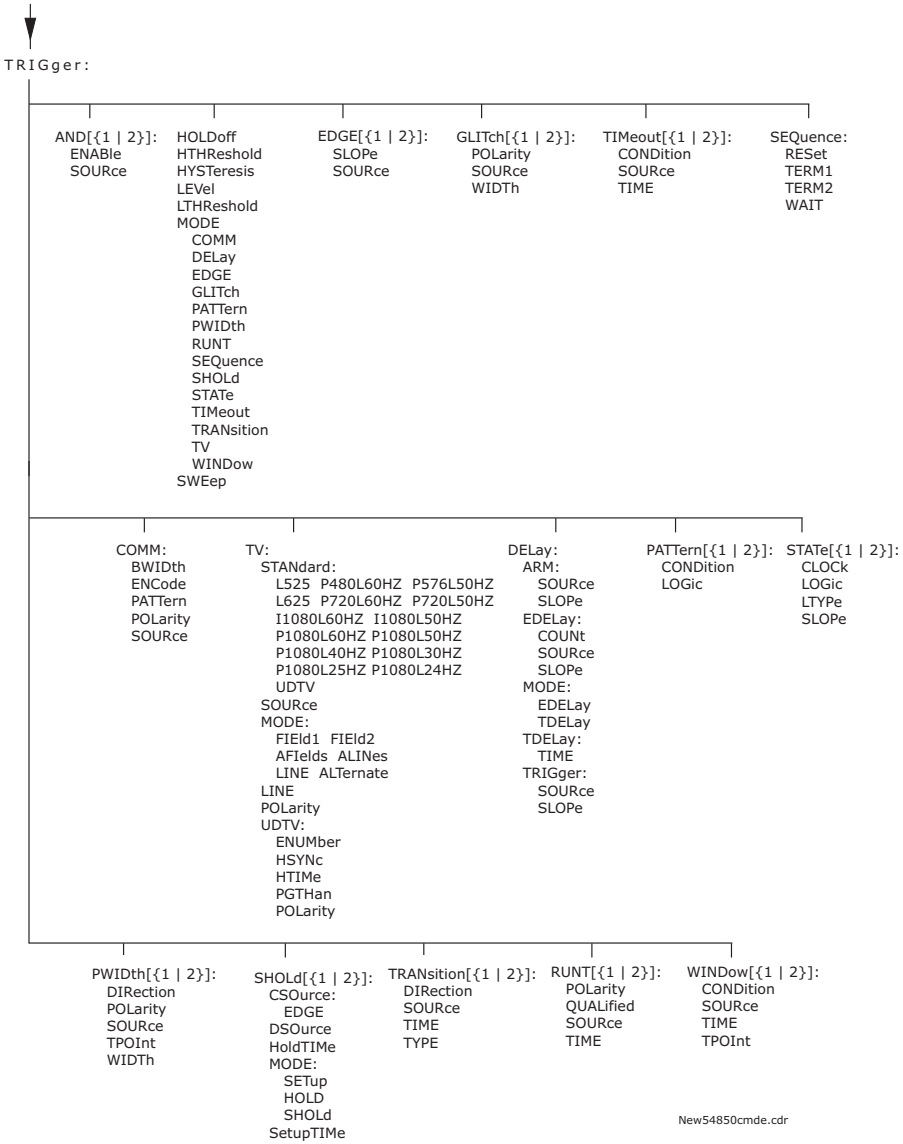
Command Tree

Figure 5-1 (continued)



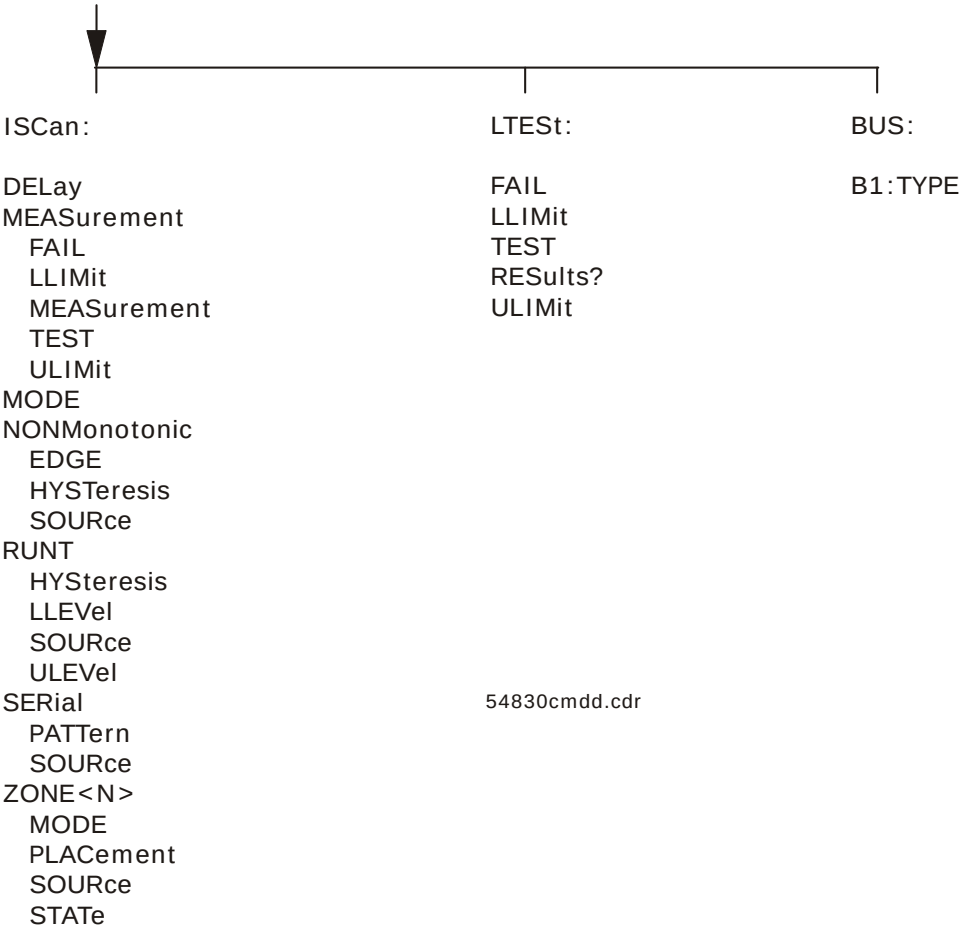
Command Tree

Figure 5-1 (continued)



Command Tree

Figure 5-1 (continued)



Command Tree

Tree Traversal Examples

The OUTPUT statements in the following examples are written using HP BASIC 5.0. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

Example 1

Consider the following command:

```
OUTPUT 707;":CHANNEL1:RANGE 0.5;OFFSET 0"
```

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

Example 2

Consider the following commands:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;POSITION 0.00001"
```

or

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
OUTPUT 707;":TIMEBASE:POSITION 0.00001"
```

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

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

Example 3

Consider the following command:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;:CHANNEL1:OFFSET 0"
```

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

Infinity Representation

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

Sequential and Overlapped Commands

IEEE 488.2 makes a distinction between sequential and overlapped commands. Sequential commands finish their task before the execution of the next command starts. Overlapped commands run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

Response Generation

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

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

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

EOI

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

7

Sample Programs

Sample Programs

Sample programs for the Infiniium-Series Oscilloscopes are shipped on a CD ROM with the instrument. Each program demonstrates specific sets of instructions.

This chapter shows you some of those functions, and describes the commands being executed. Both C and BASIC examples are included.

The header file is:

- gpibdecl.h

The C examples include:

- init.c
- learnstr.c
- sicl_IO.c
- natl_IO.c

The BASIC examples include:

- init.bas
- lrn_str.bas

The sample program listings are included at the end of this chapter.

Sample Program Structure

This chapter includes segments of both the C and BASIC sample programs. Each program includes the basic functions of initializing the interface and oscilloscope, capturing the data, and analyzing the data.

In general, both the C and BASIC sample programs typically contain the following fundamental segments:

Segment	Description
main program	**Defines global variables and constants, specifies include files, and calls various functions.**
initialize	**Initializes the GPIB or LAN interface and oscilloscope, and sets up the oscilloscope and the ACQuire subsystem.**
acquire_data	**Digitizes the waveform to capture data.**
auto_measurements	**Performs simple parametric measurements.**
transfer_data	**Brings waveform data and voltage/timing information (the preamble) into the computer.**

The BASIC programming language can be used to set up and transfer data to your PC. However, because of the limitations of BASIC, it is not the best language to use when transferring large amounts of data to your PC.

Sample C Programs

Segments of the sample programs “init.c” and “gen_srq.c” are shown and described in this chapter.

init.c - Initialization

```
/* init. c */

/* Command Order Example.  This program demonstrates the order of commands
   suggested for operation of the oscilloscope via GPIB.
   This program initializes the oscilloscope, acquires data, performs
   automatic measurements, and transfers and stores the data on the
   PC as time/voltage pairs in a comma-separated file format useful
   for spreadsheet applications. It assumes a SICL INTERFACE exists
   as 'hpib7' and an oscilloscope at address 7.
   It also requires a waveform connected to Channel 1.

   See the README file on the demo disk for development and linking information.
*/

#include <stdio.h>          /* location of: printf() */
#include <stdlib.h>         /* location of: atof(), atoi() */
#include "gpibdecl.h"       /* prototypes, global declarations, constants */

void initialize( void );         /* initialize the oscilloscope */
void acquire_data( void );       /* digitize waveform */
void auto_measurements( void ); /* perform built-in automatic measurements */
void transfer_data( void );  /* transfers waveform data from oscilloscope to PC */
int convert_data( int, int );       /* converts data to time/voltage values */
void store_csv( FILE *, int ); /* stores time/voltage pairs to */
                               /* comma-separated variable file format */
```

The include statements start the program. The file “gpibdecl.h” includes prototypes and declarations that are necessary for the Infiniium Oscilloscope sample programs.

This segment of the sample program defines the functions, in order, that are used to initialize the oscilloscope, digitize the data, perform measurements, transfer data from the oscilloscope to the PC, convert the digitized data to time and voltage pairs, and store the converted data in comma-separated variable file format.

See the following descriptions of the program segments.

init.c - Global Definitions and Main Program

```
/* GLOBALS */
int count;
double xorg,xinc;         /* values necessary for conversion of data */
double yorg,yinc;
int Acquired_length;
char data[MAX_LENGTH];          /* data buffer */
double time_value[MAX_LENGTH]; /* time value of data */
double volts[MAX_LENGTH];       /* voltage value of data */


void main( void )
{
/* initialize interface and device sessions */
/* note: routine found in sicl_IO.c or natl_IO.c  */

   if( init_IO( ) )
   {

      /* initialize the oscilloscope and interface and set up SRQ */
      initialize( );
      acquire_data( );      /* capture the data */

      /* perform automated measurements on acquired data */
      auto_measurements( );

      /* transfer waveform data to the PC from oscilloscope */
      transfer_data( );
      close_IO( );       /* close interface and device sessions */
   }
} /* end main( ) */
```

The init_IO routine initializes the oscilloscope and interface so that the oscilloscope can capture data and perform measurements on the data. At the start of the program, global symbols are defined which will be used to store and convert the digitized data to time and voltage values.

init.c - Initializing the Oscilloscope

```
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the oscilloscope for proper
*  acquisition of data. The instrument is reset to a known state and the
*  interface is cleared. System headers are turned off to allow faster
*  throughput and immediate access to the data values requested by queries.
*  The oscilloscope time base, channel, and trigger subsystems are then
*  configured. Finally, the acquisition subsystem is initialized.
*/
void initialize( void )
{
   write_IO("*RST");      /* reset oscilloscope - initialize to known state */
   write_IO("*CLS");       /* clear status registers and output queue */

   write_IO(":SYSTem:HEADer OFF"); /* turn off system headers */

   /* initialize time base parameters to center reference, */
   /* 2 ms full-scale (200 us/div), and 20 us delay */
   write_IO(":TIMebase:REFerence CENTer;RANGe 2e-3;POSition 20e-6");


   /* initialize Channel1 1.6V full-scale (200 mv/div); offset-400mv */
   write_IO(":CHANnel1:RANGe 1.6;OFFSet-400e-3");


   /* initialize trigger info: channel1 waveform on positive slope at 300mv */
   write_IO(":TRIGger:EDGE:SOURce CHANnel1;SLOPe POSitive");
   write_IO(":TRIGger:LEVel CHANnel1,-0.40");


   /* initialize acquisition subsystem */
   /* Real time acquisition - no averaging; memory depth 1,000,000 */
   write_IO(":ACQuire:MODE RTIMe;AVERage OFF;POINts 1000000");

}  /* end initialize() */
```

init.c - Acquiring Data

```
/*
*  Function name:  acquire_data
*  Parameters: none
*  Return value:  none
*  Description:  This routine acquires data according to the current
*  instrument settings.
*/
void acquire_data( void )
{
/*
*  The root level :DIGitize command is recommended for acquisition of new
*  data. It will initialize data buffers, acquire new data, and ensure that
*  acquisition criteria are met before acquisition of data is stopped.  The
*  captured data is then available for measurements, storage, or transfer
*  to a PC.  Note that the display is automatically turned off by the
*  :DIGitize command and must be turned on to view the captured data.
*/

   write_IO(":DIGitize CHANnel1");
   write_IO(":CHANnel1:DISPlay ON"); /* turn on channel 1 display which is */
                                     /* turned off by the :DIGitize command */

} /*  end acquire_data() */
```

init.c - Making Automatic Measurements

```
/*
*  Function name:  auto_measurements
*  Parameters:  none
*  Return value:  none
*  Description:  This routine performs automatic measurements of volts
*  peak-to-peak and frequency on the acquired data. It also demonstrates
*  two methods of error detection when using automatic measurements.
*/

void auto_measurements( void )
{
   float frequency, vpp;
   unsigned char vpp_str[16];
   unsigned char freq_str[16];
   int bytes_read;

/*
*  Error checking on automatic measurements can be done using one of two methods.
*  The first method requires that you turn on results in the Measurements
*  subsystem using the command :MEASure:SEND ON.  When this is on, the oscilloscope
*  will return the measurement and a result indicator.  The result flag is zero
*  if the measurement was successfully completed, otherwise a non-zero value is
*  returned which indicates why the measurement failed.
*
*  The second method simply requires that you check the return value of the
*  measurement.  Any measurement not made successfully will return with the value
*  +9.999E37. This could indicate that either the measurement was unable to be
*  performed, or that insufficient waveform data was available to make the
*  measurement.
*/
/*
*  METHOD ONE - turn on results to indicate whether the measurement completed
*  successfully.  Note that this requires transmission of extra data from the
*  oscilloscope.
*/
   write_IO(":MEASure:SENDvalid ON");    /* turn results on */

   /* query volts peak-to-peak channel 1 */
   write_IO(":MEASure:VPP? CHANnel1");

   bytes_read = read_IO(vpp_str,16L);    /* read in value and result flag */

   if (vpp_str[bytes_read-2] != '0')
      printf("Automated vpp measurement error with result %c\n",
              vpp_str[bytes_read-2]);
   else
      printf("VPP is %f\n",(float)atof(vpp_str));
```

```
   write_IO(":MEASure:FREQuency? CHANnel1");  /* frequency channel 1 */

   bytes_read = read_IO(freq_str,16L);   /* read in value and result flag */

   if (freq_str[bytes_read-2] != '0')
      printf("Automated frequency measurement error with result %c\n",
              freq_str[bytes_read-2]);
   else
      printf("Frequency is %f\n",(float)atof(freq_str));

/*
*  METHOD TWO - perform automated measurements and error checking with
*  :MEAS:RESULTS OFF
*/
   frequency =(float)0;
   vpp = (float)0;

/* turn off results */
   write_IO(":MEASure:SENDvalid OFF");

   write_IO(":MEASure:FREQuency? CHANnel1");  /* frequency channel 1 */
   bytes_read = read_IO(freq_str,16L);   /* read in value and result flag */

   frequency = (float) atof(freq_str);

   if ( frequency > 9.99e37 )
      printf("\nFrequency could not be measured.\n");
   else
      printf("\nThe frequency of channel 1 is %f Hz.\n", frequency );

   write_IO(":MEASure:VPP? CHANnel1");
   bytes_read = read_IO( vpp_str,16L );

   vpp = (float) atof(vpp_str);

   if ( vpp > 9.99e37 )
      printf("Peak-to-peak voltage could not be measured.\n");
   else
      printf("The voltage peak-to-peak is %f volts.\n", vpp );

} /* end auto_measurements() */
```

init.c - Transferring Data to the PC

```
/*
*  Function name:  transfer_data
*  Parameters:  none
*  Return value:  none
*  Description:  This routine transfers the waveform conversion factors and
*  waveform data to the PC.
*/

void transfer_data( void )
{
   int header_length;
   char header_str[8];
   FILE *fp;
   int time_division=0;

   char xinc_str[32],xorg_str[32];
   char yinc_str[32],yorg_str[32];

   int bytes_read;

   write_IO(":WAVeform:SOURce CHANnel1"); /* waveform data source channel 1 */
   write_IO(":WAVeform:FORMat BYTE");       /* setup transfer format */


   write_IO(":WAVeform:XINCrement?");      /* request values to allow
                                              interpretation of raw data */
   bytes_read = read_IO(xinc_str,32L);
   xinc = atof(xinc_str);


   write_IO(":WAVeform:XORigin?");
   bytes_read = read_IO(xorg_str,32L);
   xorg = atof(xorg_str);

   write_IO(":WAVeform:YINCrement?");
   bytes_read = read_IO(yinc_str,32L);
   yinc = atof(yinc_str);

   write_IO(":WAVeform:YORigin?");
   bytes_read = read_IO(yorg_str,32L);
   yorg = atof(yorg_str);

   write_IO(":WAVeform:DATA?");            /* request waveform data */
   bytes_read = read_IO(data,1L);            /* fine the # character */
   while(data[0] != '#')
      bytes_read = read_IO(data,1L);            /* fine the # character */
```

```
    bytes_read = read_IO(header_str,1L);      /* input byte counter */
    header_length = atoi(header_str);

    /* read number of points to download */
    bytes_read = read_IO(header_str,(long)header_length);
    Acquired_length = atoi(header_str);                /* number of bytes */

    bytes_read = 0;

    fp = fopen("pairs.csv","wb");    /* open file in binary mode - clear file
                                        if already exists */

    while((bytes_read + MAX_LENGTH) < Acquired_length)
    {
       bytes_read += read_IO(data,MAX_LENGTH); /* input waveform data */
       /* Convert data to voltage and time */
       time_division = convert_data(time_division,MAX_LENGTH);
       store_csv(fp,MAX_LENGTH);               /* Store data to disk */
    }

    /* input last of waveform data */
    bytes_read = read_IO(data,(Acquired_length-bytes_read+1));
    /* Convert data to voltage and time */
    time_division = convert_data(time_division,(bytes_read-1));
    store_csv(fp,(bytes_read-1));               /* Store data to disk */

    fclose( fp );                                      /* close file */

} /* end transfer_data() */
```

An example header resembles the following when the information is stripped off:

```
#510225
```

The left most “5” defines the number of digits that follow (10225). The number “10225” is the number of points in the waveform. The information is stripped off of the header to get the number of data bytes that need to be read from the oscilloscope.

init.c - Converting Waveform Data

```
/*
*  Function name: convert_data
*  Parameters:    int time_division which is the index value of the next time
*                     value calculated.
*                 int length number of voltage and time values to calculate.
*  Return value:  int time_division which contains the next time index.
*  Description:  This routine converts the waveform data to time/voltage
*  information using the values that describe the waveform.  These values are
*  stored in global arrays for use by other routines.
*/

int convert_data( int time_division, int length )
{
   int i;

   for (i = 0; i < Acquired_length; i++)
   {
      /* calculate time info */
      time_value[i] =(time_division * xinc) + xorg;
      /* calculate volt info */
      volts[i] = (data[i] * yinc) + yorg;
      time_division++;
   }

   return time_division;
} /* end convert_data() */
```

The data values are returned as digitized samples (sometimes called quantization levels or q-levels). These data values must be converted into voltage and time values.

init.c - Storing Waveform Time and Voltage Information

```
/*
*  Function name:  store_csv
*  Parameters:  none
*  Return value:  none
*  Description:  This routine stores the time and voltage information about
*  the waveform as time/voltage pairs in a comma-separated variable file
*  format.
*/

void store_csv( FILE *fp, int length )
{
   int i;

   if (fp != NULL)
   {
      for (i = 0; i < length; i++)
      {
         /* write time,volt pairs to file */
         fprintf( fp,"%e,%lf\n",time_value[i],volts[i]);
      }
   }
   else
      printf("Unable to open file 'pairs.csv'\n");

}  /* end store_csv() */
```

The time and voltage information of the waveform is stored with the time stored first, followed by a comma, and the voltage stored second.

Listings of the Sample Programs

Listings of the C sample programs in this section include:

- gpibdecl.h
- learnstr.c
- sicl_IO.c
- natl_IO.c

Listings of the BASIC sample programs in this section include:

- init.bas
- lrn_str.bas

gpibdecl.h Sample Header

```
/* gpibdecl.h */

/* This file includes necessary prototypes and declarations for the
   example programs for the Agilent oscilloscope */

/* User must indicate which GPIB card (Agilent or National) is being used or
   if the LAN interface is being used.
   Also, if using a National card, indicate which version of windows
   (WIN31 or WIN95) is being used */


#define LAN    /* Uncomment if using LAN interface */
#define AGILENT   /* Uncomment if using LAN or Agilent interface card */
// #define NATL   /* Uncomment if using National interface card */

/* #define WIN31 */        /* For National card ONLY - select windows version */
#define WIN95


#ifdef WIN95
   #include <windows.h>         /* include file for Windows 95 */
#else
   #include <windecl.h>         /* include file for Windows 3.1 */
#endif

#ifdef AGILENT
   #include "d:\siclnt\c\sicl.h" /* Change the path for the sicl.h location */
#else
   #include "decl-32.h"
#endif

#define CME 32
#define EXE 16
#define DDE  8
#define QYE  4

#define SRQ_BIT 64
#define MAX_LRNSTR 40000
#define MAX_LENGTH 262144
#define MAX_INT 4192

#ifdef AGILENT
   #ifdef LAN
      #define INTERFACE "lan[130.29.71.82]:hpib7,7"
   #else
```

```
      #define DEVICE_ADDR "hpib7,7"
      #define INTERFACE "hpib7"
   #endif
#else
   #define INTERFACE "gpib0"

   #define board_index 0
   #define prim_addr 7
   #define second_addr 0
   #define timeout 13
   #define eoi_mode  1
   #define eos_mode 0
#endif

/*  GLOBALS */
#ifdef AGILENT
   INST bus;
   INST scope;
#else
   int bus;
   int scope;
#endif

#define TRUE 1
#define FALSE 0

extern int srq_asserted;

/* GPIB prototypes */
void init_IO( void );
void write_IO( char* );
void write_lrnstr( char*, long );
int read_IO( char*, unsigned long );
unsigned char read_status( );
void close_IO( void );
void gpiberr( void );

#ifdef AGILENT
   extern void SICLCALLBACK srq_agilent( INST );
#else
   extern int __stdcall srq_national( int, int, int, long, void* );
#endif
```

learnstr.c Sample Program

```
/* learnstr.c */

/*
*  This example program initializes the oscilloscope, runs autoscale to
*  acquire a waveform, queries for the learnstring, and stores the learnstring
*  to disk.  It then allows the user to change the setup, then restores the
*  original learnstring. It assumes that a waveform is attached to the
*  oscilloscope.
*/

#include <stdio.h>         /* location of: printf(), fopen(), fclose(),
                              fwrite(),getchar */
#include "gpibdecl.h"

void initialize( void );
void store_learnstring( void );
void change_setup( void );
void get_learnstring( void );

void main( void )
{
   if( init_IO( ) )    /* initialize device and interface  */
   {                   /* Note: routine found in sicl_IO.c or natl_IO.c */
      /* initialize the oscilloscope and interface, and set up SRQ */
      initialize( );
      store_learnstring( ); /* request learnstring and store */
      change_setup( );      /* request user to change setup */
      get_learnstring( );   /* restore learnstring */
      close_IO( );          /* close device and interface sessions */
                         /* Note: routine found in sicl_IO.c or natl_IO.c */
   }
} /* end main */
```

```
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the oscilloscope for proper
*  acquisition of data. The instrument is reset to a known state and the
*  interface is cleared. System headers are turned off to allow faster
*  throughput and immediate access to the data values requested by queries.
*  Autoscale is performed to acquire a waveform. The waveform is then
*  digitized, and the channel display is turned on following the acquisition.
*/

void initialize( void )
{
   write_IO("*RST");  /* reset oscilloscope - initialize to known state */
   write_IO("*CLS");  /* clear status registers and output queue */

   write_IO(":SYSTem:HEADer ON");               /* turn on system headers */

   /* initialize Timebase parameters to center reference, 2 ms
      full-scale (200 us/div), and 20 us delay */
   write_IO(":TIMebase:REFerence CENTer;RANGe 5e-3;POSition 20e-6");

   /* initialize Channel1 1.6v full-scale (200 mv/div);
      offset-400mv */
   write_IO(":CHANnel1:RANGe 1.6;OFFSet-400e-3");

   /* initialize trigger info: channel1 waveform on positive slope
      at 300mv */
   write_IO(":TRIGger:EDGE:SOURce CHANnel1;SLOPe POSitive");
   write_IO(":TRIGger:LEVel CHANnel1,-0.40");

   /* initialize acquisition subsystem */
   /* Real time acquisition - no averaging; record length 4096 */
   write_IO(":ACQuire:MODE RTIMe;AVERage OFF;POINts 4096");

} /* end initialize() */
```

```
/*
*  Function name:  store_learnstring
*  Parameters:  none
*  Return value:  none
*  Description:  This routine requests the system setup known as a
*  learnstring. The learnstring is read from the oscilloscope and stored in a file
*  called Learn2.
*/

void store_learnstring( void )
{
   FILE *fp;
   unsigned char setup[MAX_LRNSTR]={0};
   int actualcnt = 0;

   write_IO(":SYSTem:SETup?");              /* request learnstring */
   actualcnt = read_IO(setup, MAX_LRNSTR);

   fp = fopen( "learn2","wb");


   if ( fp != NULL )
   {
      fwrite( setup,sizeof(unsigned char),(int)actualcnt,fp);
      printf("Learn string stored in file Learn2\n");

      fclose( fp );
   }
   else
      printf("Error in file open\n");

}/* end store_learnstring */

/*
*  Function name:  change_setup
*  Parameters:  none
*  Return value:  none
*  Description:  This routine places the oscilloscope into local mode to allow the
*  customer to change the system setup.
*/

void change_setup( void )
{
   printf("Please adjust setup and press ENTER to continue.\n");
   getchar();

} /* end change_setup */
```

```
/*
*  Function name:  get_learnstring
*  Parameters:  none
*  Return value:  none
*  Description:  This routine retrieves the system setup known as a
*  learnstring from a disk file called Learn2. It then restores
*  the system setup to the oscilloscope.
*/

void get_learnstring( void )
{
   FILE *fp;
   unsigned char setup[MAX_LRNSTR];
   unsigned long count = 0;

   fp = fopen( "learn2","rb");

   if ( fp != NULL )
   {
      count = fread( setup,sizeof(unsigned char),MAX_LRNSTR,fp);

      fclose( fp );
   }
   write_lrnstr(setup,count);           /* send learnstring */
   write_IO(":RUN");

}/* end get_learnstring */
```

sicl_IO.c Sample Program

```
/* sicl_IO.c */

#include <stdio.h>              /* location of: printf() */
#include <string.h>             /* location of: strlen() */
#include "gpibdecl.h"

/*  This file contains IO and initialization routines for the SICL libraries. */
/*
*  Function name:  init_IO
*  Parameters:  none
*  Return value:  int indicating success or failure of initialization.
*  Description:  This routine initializes the SICL environment. It sets up
*  error handling, opens both an interface and device session, sets timeout
*  values, clears the interface by pulsing IFC, and clears the instrument
*  by performing a Selected Device Clear.
*/

int init_IO( )
{
   ionerror(I_ERROR_EXIT);       /* set-up interface error handling */

   /* open interface session for verifying SRQ line */
   bus = iopen( INTERFACE );
   if ( bus == 0 )
   {
      printf("Bus session invalid\n");
      return FALSE;
   }

   itimeout( bus, 20000 );       /* set bus timeout to 20 sec */
   iclear( bus );                /* clear the interface - pulse IFC */
```

```
#ifdef LAN
   scope = bus;
#else
   scope = iopen( DEVICE_ADDR );            /* open the scope device session */
   if ( scope == 0)
   {
      printf( "Scope session invalid\n" );
      return FALSE;
   }

   itimeout( scope, 20000 );     /* set device timeout to 20 sec */
   iclear( scope );       /* perform Selected Device Clear on oscilloscope */
#endif

   return TRUE;
} /* end init_IO */
```

```
/*
*  Function name:  write_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  output; unsigned long length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs strings to the oscilloscope device session
*  using the unformatted I/O SICL commands.
*/

void write_IO( void *buffer )
{
   unsigned long actualcnt;
   unsigned long length;
   int send_end = 1;
   length = strlen( buffer );
   iwrite( scope, buffer, length, send_end, &actualcnt );

} /* end write_IO */

/*
*  Function name:  write_lrnstr
*  Parameters:  char *buffer which is a pointer to the character string to be
*  output; long length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs a learnstring to the oscilloscope device
*  session using the unformatted I/O SICL commands.
*/

void write_lrnstr( void *buffer, long length )
{
   unsigned long actualcnt;
   int send_end = 1;

   iwrite( scope, buffer,(unsigned long) length,
      send_end, &actualcnt );

} /* end write_lrnstr() */
```

```
/*
*  Function name:  read_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  input; unsigned long length which indicates the max length of the string to
*  be input
*  Return value:  integer which indicates the actual number of bytes read
*  Description:  This routine inputs strings from the oscilloscope device session
*  using SICL commands.
*/

int read_IO(void *buffer,unsigned long length)
{
   int reason;
   unsigned long actualcnt;

   iread(scope,buffer,length,&reason,&actualcnt);

   return( (int) actualcnt );
}

/*
*  Function name: check_SRQ
*  Parameters:  none
*  Return value:  integer indicating if bus SRQ line was asserted
*  Description:  This routine checks for the status of SRQ on the bus and
*  returns a value to indicate the status.
*/

int check_SRQ( void )
{
   int srq_asserted;

      /* check for SRQ line status */
      igpibbusstatus(bus, I_GPIB_BUS_SRQ, &srq_asserted);

   return( srq_asserted );

} /* end check_SRQ() */
```

```
/*
*  Function name:  read_status
*  Parameters:  none
*  Return value:  unsigned char indicating the value of status byte
*  Description:  This routine reads the oscilloscope status byte and returns
*  the status.
*/

unsigned char read_status( void )
{
   unsigned char statusbyte;

   /* Always read the status byte from instrument */
   /* NOTE: ireadstb uses serial poll to read status byte - this
            should clear bit 6 to allow another SRQ. */

      ireadstb( scope, &statusbyte );
   return( statusbyte );

} /* end read_status() */

/*
*  Function name:  close_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine closes device and interface sessions for the
*  SICL environment and calls the routine _siclcleanup which de-allocates
*  resources used by the SICL environment.
*/

void close_IO( void )
{
   iclose( scope );   /* close device session */
   iclose( bus );     /* close interface session */

   _siclcleanup();    /* required for 16-bit applications */

} /* end close_SICL() */
```

natl_IO.c Sample Program

```
/* natl_IO.c */

#include <stdio.h>      /* location of: printf() */
#include <string.h>     /* location of: strlen() */
#include "gpibdecl.h"

/*  This file contains IO and initialization routines for the NI488.2 commands. */
/*
*  Function name:  gpiberr
*  Parameters:  char* - string describing error
*  Return value:  none
*  Description:  This routine outputs error descriptions to an error file.
*/

void gpiberr( char *buffer )
{
   printf("Error string: %s\n",buffer );

} /* end gpiberr() */

/*
*  Function name:  init_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the NI environment. It sets up error
*  handling, opens both an interface and device session, sets timeout values
*  clears the interface by pulsing IFC, and clears the instrument by performing
*  a Selected Device Clear.
*/

void init_IO( void )
{
   bus = ibfind( INTERFACE );          /* open and initialize GPIB board */
   if( ibsta & ERR )
      gpiberr("ibfind error");

   ibconfig( bus, IbcAUTOPOLL, 0);  /* turn off autopolling */

   ibsic( bus );                    /* clear interface - pulse IFC */
   if( ibsta & ERR )
   {
      gpiberr( "ibsic error" );
   }
```

```
   /* open device session */
   scope = ibdev( board_index, prim_addr, second_addr, timeout,
                  eoi_mode, eos_mode );
   if( ibsta & ERR )
   {
      gpiberr( "ibdev error" );
   }

   ibclr( scope );                     /* clear the device( scope ) */

   if( ibsta & ERR)
   {
      gpiberr("ibclr error" );
   }

} /* end init_IO */

/*
*  Function name:  write_IO
*  Parameters:  void *buffer which is a pointer to the character string
*               to be output
*  Return value:  none
* Description: This routine outputs strings to the oscilloscope device session.
*/
void write_IO( void *buffer )
{
   long length;

   length = strlen( buffer );

   ibwrt( scope, buffer, (long) length );
   if ( ibsta & ERR )
   {
      gpiberr( "ibwrt error" );
   }

} /* end write_IO() */
```

```
/*
*  Function name:  write_lrnstr
*  Parameters:  void *buffer which is a pointer to the character string to
*  be output; length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs a learnstring to the oscilloscope device
*  session.
*/
void write_lrnstr( void *buffer, long length )
{

   ibwrt( scope, buffer, (long) length );
   if ( ibsta & ERR )
   {
      gpiberr( "ibwrt error" );
   }

} /* end write_lrnstr() */

/*
*  Function name:  read_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  input; unsigned long length which indicates the max length of the string
*  to be input
*  Return value:  integer which indicates the actual number of bytes read
* Description: This routine inputs strings from the oscilloscope device session.
*/

int read_IO(void *buffer,unsigned long length)
{
   ibrd(scope, buffer,( long )length );

   return( ibcntl );

} /* end read_IO() */
```

```
/*
*  Function name: read_status
*  Parameters:    none
*  Return value:  unsigned char indicating the value of status byte
*  Description:  This routine reads the oscilloscope status byte and returns
*  the status.
*/
unsigned char read_status( void )
{
   unsigned char statusbyte;

   /* Always read the status byte from instrument */

   ibrsp( scope,  &statusbyte );

   return( statusbyte );

} /* end read_status() */
```

```
/*
*  Function name: close_IO
*  Parameters:    none
*  Return value:  none
*  Description:  This routine closes device session.
*/

void close_IO( void )
{
   ibonl( scope,0 );  /* close device session */

} /* end close_IO() */
```

init.bas Sample Program

The BASIC programming language can be used to set up and transfer data to your PC. However, because of the limitations of BASIC, it is not the best language to use when transferring large amounts of data to your PC.

```
10    !file: init
20    !
30    !
40     !    This program demonstrates the order of commands suggested for
operation of
50    !   the oscilloscope via GPIB.  This program initializes the oscilloscope,
acquires
60    !    data, performs automatic measurements, and transfers and stores the
data on the
70    !    PC as time/voltage pairs in a comma-separated file format useful for
spreadsheet
80    !   applications.  It assumes an interface card at interface select code 7, an
90    !    oscilloscope at address 7, and the cal waveform connected to Channel 1.
100    !
110    !
120    !
130    COM /Io/@Scope,@Path,Interface
140    COM /Raw_data/ INTEGER Data(4095)
150    COM /Converted_data/ REAL Time(4095),Volts(4095)
160    COM /Variables/ REAL Xinc,Xorg,Yinc,Yorg
170    COM /Variables/ INTEGER Record_length
180    !
190    !
200    CALL Initialize
210    CALL Acquire_data
220    CALL Auto_msmts
230    CALL Transfer_data
240    CALL Convert_data
250    CALL Store_csv
260    CALL Close
270    END
280    !
```

```
290
!!!
```

```
700     !
710     !
720
!!!
```

```
1040    !
1050    !
1060    SUB Auto_msmts
1070    COM /Io/@Scope,@Path,Interface
1080    REAL Freq,Vpp
1090    DIM Vpp_str$[64]
1100    DIM Freq_str$[64]
1110    Bytes_read=0
1120    !
1130    !     Error checking on automatic measurements can be done using one of
two methods.
1140    !    The first method requires that you turn on results in the Measurement
subsystem
1150    !     using the command ":MEASure:SEND ON".  When this is on, the
oscilloscope will return the
1160    !     measurement and a result indicator.  The result flag is zero if
the measurement
1170    !    was successfully completed, otherwise a non-zero value is returned
which indicates
1180    !     why the measurement failed.  See the Programmer's Manual for
descriptions of result
1190    !     indicators.  The second method simply requires that you check the
return value of
1200    !    the measurement.  Any measurement not made successfully will return
with the value
1210    !     +9.999e37.  This could indicate that either the measurement was
unable to be
1220    !    performed or that insufficient waveform data was available to make
the measurement.
1230    !
1240    !     METHOD ONE
1250    !
1260          OUTPUT @Scope;":MEASure:SENDvalid ON"       !turn on results
1270        OUTPUT @Scope;":MEASure:VPP? CHANnel1"    !Query volts peak-to-peak
1280          ENTER @Scope;Vpp_str$
1290         Bytes_read=LEN(Vpp_str$)                   !Find length of string
1300          CLEAR SCREEN
1310          IF Vpp_str$[Bytes_read;1]="0" THEN         !Check result value
1320             PRINT
1330             PRINT "VPP is ";VAL(Vpp_str$[1,Bytes_read-1])
1340             PRINT
1350          ELSE
1360             PRINT
1370             PRINT "Automated vpp measurement error with result
";Vpp_str$[Bytes_read;1]
1380             PRINT
1390          END IF
1400    !
```

```
1410    !
1420          OUTPUT @Scope;":MEASure:FREQuency? CHANnel1"  !Query frequency
1430          ENTER @Scope;Freq_str$
1440         Bytes_read=LEN(Freq_str$)                     !Find string length
1450        IF Freq_str$[Bytes_read;1]="0" THEN         !Determine result value
1460             PRINT
1470             PRINT "Frequency is ";VAL(Freq_str$[1,Bytes_read-1])
1480             PRINT
1490          ELSE
1500             PRINT
1510             PRINT "Automated frequency measurement error with result
";Freq_str$[Bytes_read;1]
1520             PRINT
1530          END IF
1540    !
1550    !
1560    !     METHOD TWO
1570    !
1580          OUTPUT @Scope;":MEASure:SENDvalid OFF"     !turn off results
1590       OUTPUT @Scope;":MEASure:VPP? CHANnel1"     !Query volts peak-to-peak
1600          ENTER @Scope;Vpp
1610          IF Vpp<9.99E+37 THEN
1620             PRINT
1630             PRINT "VPP is ";Vpp
1640             PRINT
1650          ELSE
1660             PRINT
1670             PRINT "Automated vpp measurement error ";Vpp
1680             PRINT
1690          END IF
1700          OUTPUT @Scope;":MEASure:FREQuency? CHANnel1"
1710          ENTER @Scope;Freq
1720          IF Freq<9.99E+37 THEN
1730             PRINT
1740             PRINT "Frequency is ";Freq
1750             PRINT
1760          ELSE
1770             PRINT
1780             PRINT "Automated frequency measurement error";Freq
1790             PRINT
1800          END IF
1810   SUBEND
1820   !
1830   !
1840
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!
1850   !
```

```
1860   !
1870   !     Subprogram name:  Transfer_data
1880   !     Parameters: none
1890   !     Return value: none
1900  !    Description:  This routine transfers the waveform data and conversion
factors to
1910   !                   to PC.
1920   !
1930   !
1940   SUB Transfer_data
1950   COM /Io/@Scope,@Path,Interface
1960   COM /Raw_data/ INTEGER Data(4095)
1970   COM /Converted_data/ REAL Time(4095),Volts(4095)
1980   COM /Variables/ REAL Xinc,Xorg,Yinc,Yorg
1990   COM /Variables/ INTEGER Record_length
2000   !                                  define waveform data source and format
2010   OUTPUT @Scope;":WAVeform:SOURce CHANnel1"
2020   OUTPUT @Scope;":WAVeform:FORMat WORD"
2030  !                           request values needed to convert raw data to real
2040   OUTPUT @Scope;":WAVeform:XINCrement?"
2050   ENTER @Scope;Xinc
2060   OUTPUT @Scope;":WAVeform:XORigin?"
2070   ENTER @Scope;Xorg
2100   OUTPUT @Scope;":WAVeform:YINCrement?"
2110   ENTER @Scope;Yinc
2120   OUTPUT @Scope;":WAVeform:YORigin?"
2130   ENTER @Scope;Yorg
2160   !
2170   !                                          request data
2180   OUTPUT @Scope;":WAVeform:DATA?"
2190   ENTER @Scope USING "#,1A";First_chr$       !ignore leading #
2200   ENTER @Scope USING "#,1D";Header_length    !input number of bytes in
header value
2210  ENTER @Scope USING "#,"&VAL$(Header_length)&"D";Record_length    !Record
length in bytes
2220   Record_length=Record_length/2              !Record length in words
2230   ENTER @Scope USING "#,W";Data(*)
2240   ENTER @Scope USING "#,A";Term$             !Enter terminating character
2250   !
2260   SUBEND
2270   !
2280   !
2290
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!
2300   !
2310   !
2320   !     Subprogram name:  Convert_data
```

```
2330   !     Parameters: none
2340   !     Return value: none
2350   !     Description:  This routine converts the waveform data to time/
voltage information
2360   !              using the values Xinc, Xorg, Yinc, and Yorg used to describe
2370   !                    the raw waveform data.
2380   !
2390   !
2400   SUB Convert_data
2410   COM /Io/@Scope,@Path,Interface
2420   COM /Raw_data/ INTEGER Data(4095)
2430   COM /Converted_data/ REAL Time(4095),Volts(4095)
2440   COM /Variables/ REAL Xinc,Xorg,Yinc,Yorg
2450   COM /Variables/ INTEGER Record_length
2460   !
2470   FOR I=0 TO Record_length-1
2480        Time(I)=(I-*Xinc)+Xorg
2490        Volts(I)=(Data(I)*Yinc)+Yorg
2500   NEXT I
2510   SUBEND
2520   !
2530   !
2540
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!
2550   !
2560   !
2570   !     Subprogram name: Store_csv
2580   !     Parameters: none
2590   !     Return value: none
2600   !    Description:  This routine stores the time and voltage information
about the waveform
2610   !                  as time/voltage pairs in a comma-separated variable
file format.
2620   !
2630   !
2640   SUB Store_csv
2650   COM /Io/@Scope,@Path,Interface
2660   COM /Converted_data/ REAL Time(4095),Volts(4095)
2670   COM /Variables/ REAL Xinc,Xorg,Yinc,Yorg
2680   COM /Variables/ INTEGER Record_length
2690         !Create a file to store pairs in
2700   ON ERROR GOTO Cont
2710   PURGE "Pairs.csv"
2720 Cont: OFF ERROR
2730   CREATE "Pairs.csv",Max_length
2740   ASSIGN @Path TO "Pairs.csv";FORMAT ON
2750                                  !Output data to file
```

```
2760   FOR I=0 TO Record_length-1
2770      OUTPUT @Path;Time(I),Volts(I)
2780   NEXT I
2790   SUBEND
2800   !
2810   !
2820
!!!
```

lrn_str.bas Sample Program

The BASIC programming language can be used to set up and transfer data to your PC. However, because of the limitations of BASIC, it is not the best language to use when transferring large amounts of data to your PC.

```
10    !FILE: lrn_str.bas
20    !
30    !THIS PROGRAM WILL INITIALIZE THE OSCILLOSCOPE, AUTOSCALE, AND DIGITIZE
THE WAVEFORM
40   !INFORMATION.  IT WILL THEN QUERY THE INSTRUMENT FOR THE LEARNSTRING AND WILL
50   !SAVE THE INFORMATION TO A FILE.  THE PROGRAM WILL THEN PROMPT YOU TO CHANGE
60   !THE SETUP THEN RESTORE THE ORIGINAL LEARNSTRING CONFIGURATION. IT ASSUMES
70   !AN OSCILLOSCOPE at ADDRESS 7, GPIB INTERFACE at 7, AND THE CAL waveform
ATTACHED TO
80    !CHANNEL 1.
90    !
100   !
110   COM /Io/@Scope,@Path,Interface
120   COM /Variables/Max_length
130   CALL Initialize
140   CALL Store_lrnstr
150   CALL Change_setup
160   CALL Get_lrnstr
170   CALL Close
180   END
190   !
200   !
210
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
220   !
230   !                    BEGIN SUBROUTINES
240   !
250
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
260   !    Subprogram name:  Initialize
270   !    Parameters: none
280   !    Return value: none
290   !    Description:  This routine initializes the path descriptions and
resets the
300   !               interface and the oscilloscope.  It performs an autoscale
```

```
on the waveform,
310   !                acquires the data on channel 1, and turns on the display.
320   !           NOTE:  This routine also turns on system headers.  This allows the
330   !                    string ":SYSTEM:SETUP " to be returned with the
learnstring so the
340   !                    return string is in the proper format.
350   !
360   SUB Initialize
370         COM /Io/@Scope,@Path,Interface
380         COM /Variables/Max_length
390         Max_length=40000
400         ASSIGN @Scope TO 707
410         Interface=7
420         RESET Interface
430         CLEAR @Scope
440         OUTPUT @Scope;"*RST"
450         OUTPUT @Scope;"*CLS"
460         OUTPUT @Scope;":SYSTem:HEADer ON"
470         OUTPUT @Scope;":AUToscale"
480   SUBEND
490   !
500   !
510
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
520   !
530   !
540   !     Subprogram name: Store_lrnstr
550   !     Parameters: none
560   !     Return value: none
570   !     Description:  This routine creates a file in which to store the
learnstring
580   !            configuration (Filename:Lrn_strg).  It requests the learnstring
590   !             and inputs the configuration to the PC.  Finally, it stores the
600   !                 configuration to the file.
610   !
620   SUB Store_lrnstr
630         COM /Io/@Scope,@Path,Interface
640         COM /Variables/Max_length
650         ON ERROR GOTO Cont
660         PURGE "Lrn_strg"
670 Cont:   OFF ERROR
680         CREATE BDAT "Lrn_strg",1,40000
690         DIM Setup$[40000]
700         ASSIGN @Path TO "Lrn_strg"
710         OUTPUT @Scope;":SYSTem:SETup?"
720         ENTER @Scope USING "-K";Setup$
730         OUTPUT @Path,1;Setup$
```

```
740         CLEAR SCREEN
750         PRINT "Learn string stored in file: Lrn_strg"
760    SUBEND
770    !
780    !
790
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!
800    !
810    !      Subprogram name: Change_setup
820    !      Parameters: none
830    !      Return value: none
840    !      Description:  This subprogram requests that the user change the
850    !                    oscilloscope setup, then press a key to continue.
860    !
870    !
880    SUB Change_setup
890         COM /Io/@Scope,@Path,Interface
900
910         PRINT
920         PRINT "Please adjust setup and press Continue to resume."
930         PAUSE
940    SUBEND
950    !
960    !
970
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!
980    !
990    !      Subprogram name: Get_lrnstr
1000   !      Parameters: none
1010   !      Return value: none
1020   !      Description:  This subprogram loads a learnstring from the
1030   !                    file "Lrn_strg" to the oscilloscope.
1040   !
1050   !
1060   SUB Get_lrnstr
1070        COM /Io/@Scope,@Path,Interface
1080        COM /Variables/Max_length
1090        DIM Setup$[40000]
1100        ENTER @Path,1;Setup$
1110        OUTPUT @Scope USING "#,-K";Setup$
1120        OUTPUT @Scope;":RUN"
1130   SUBEND
1140   !
1150   !
```

```
1160
!!!
```

8

Acquire Commands

Acquire Commands

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

These ACQuire commands and queries are implemented in the Infiniium Oscilloscopes:

- AVERage
- AVERage:COUNt
- COMPlete
- COMPlete:STATe
- BANDwidth (Enhanced bandwidth or noise reduction option only)
- INTerpolate
- MODE
- POINts (memory depth)
- POINts:AUTO
- RESPonse
- SEGMented:COUNt
- SEGMented:INDex
- SEGMented:TTAGs
- SRATe (sampling rate)
- SRATe:AUTO

AVERage

Command

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

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

Averaging is not available in PDETect mode.

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

Example

This example turns averaging on.

```
10  OUTPUT 707;":ACQUIRE:AVERAGE ON"
20  END
```

Query

```
:ACQuire:AVERage?
```

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

Returned Format

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

Example

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

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:AVERAGE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

AVERage:COUNt

Command

```
:ACQuire:AVERage:COUNt <count_value>
```

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

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

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

Example

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

```
10  OUTPUT 707;":ACQUIRE:COUNT 16"
20  END
```

Query

```
:ACQuire:COUNt?
```

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

Returned Format

```
[:ACQuire:COUNt] <value><NL>
```

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

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:AVERAGE:COUNT?"
30  ENTER 707;Result
40  PRINT Result
50  END
```

COMPlete

Command

```
:ACQuire:COMPlete <percent>
```

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

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

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

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

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

Example

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

```
10  OUTPUT 707;":ACQUIRE:COMPLETE 90"
20  END
```

Query `:ACQuire:COMPlete?`

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

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

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

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF”
20  OUTPUT 707;":ACQUIRE:COMPLETE?"
30  ENTER 707;Percent
40  PRINT Percent
50  END
```

COMPlete:STATe

Command

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

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

Use :ACQuire:COMPlete:STATe when DIGitize is Not Performing

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

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

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

Query

```
:ACQuire:COMPlete:STATe?
```

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

BANDwidth

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

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

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

`AUTO` Sets the oscilloscope to the hardware bandwidth limit and disables the bandwidth filter.

`<bandwidth>` a real number representing the bandwidth of the bandwidth filter whose range of values depends on the model number of your oscilloscope.

`91304A` 13E09, 12E09, 10E09, 8E09, 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09

`91204A` 12E09, 10E09, 8E09, 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09

`90804A` 8E09, 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09

`90604A` 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09

`90404A` 4E09, 3E09, 2.5E09, 2E09, 1E09

`90254A` 2.5E09, 2E09, 1E09

Query `:ACQuire:BANDwidth?`

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

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

INTerpolate

Command

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

The :ACQuire:INTerpolate command turns the sin(x)/x interpolation filter on or off when the oscilloscope is in one of the real time sampling modes.

Query

```
:ACQuire:INTerpolate?
```

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

Returned Format

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

MODE

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

The :ACQuire:MODE command sets the acquisition mode of the oscilloscope. Sampling mode can be Real Time Normal, Real Time Peak Detect, or Real Time High Resolution.

`ETIMe` In Equivalent Time mode, the data record is acquired over multiple trigger events.

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

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

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

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

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

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

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

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

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

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

SEGPdetect Enables Peak Detect Segmented mode.

SEGHres Enables High Resolution Segmented mode.

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

```
10  OUTPUT 707;":ACQUIRE:MODE RTIME"
20  END
```

Query `:ACQuire:MODE?`

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

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

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

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:MODE?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

POINts

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

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

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

`<points_value>` An integer representing the memory depth.

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

Table 8-1

1 G Memory Option Installed

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

Table 8-2

500 M Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	Auto or Triggered
Normal, Peak Detect (full sample rate), and High Resolution modes	16 to 512,500,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
High Resolutiont Mode at < full sample rate	16 to 512,500,000	16 to 256,250,000
Peak Detect Mode at < full sample rate	16 to 256,250,000	16 to 128,125,000
Equivalent Time Mode	16 to 262,144	

Table 8-3

200 M Memory Option Installed

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

Table 8-4

100 M Memory Option Installed

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

Table 8-5

50 M Memory Option Installed

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

Table 8-6

20 M Memory Option Installed

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

Table 8-7

10 M Standard Memory

Sampling mode and sample rate	Trigger Sweep	
	Single	Auto or Triggered
Normal, Peak Detect (full sample rate), and High Resolution modes	16 to 10,250,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
Peak Detect Mode at < full sample rate	16 to 10,250,000	
Equivalent Time Mode	16 to 262,144	

Interaction between :ACQuire:SRATe and :ACQuire:POINts

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

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

Example

This example sets the memory depth to 500 points.

```
10  OUTPUT 707;":ACQUIRE:POINTS 500"
20  END
```

Query

```
:ACQuire:POINts?
```

The :ACQuire:POINts? query returns the value of the memory depth control.

Returned Format

```
[:ACQuire:POINts] <points_value><NL>
```

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF”
20  OUTPUT 707;":ACQUIRE:POINTS?"
30  ENTER 707;Length
40  PRINT Length
50  END
```

See Also

:WAVeform:DATA?

POINts:AUTO

Command

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

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

Example

This example sets the automatic memory depth control to off.

```
10  OUTPUT 707;":ACQUIRE:POINTS:AUTO OFF"
20  END
```

Query

```
:ACQuire:POINts:AUTO?
```

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

Returned Format

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

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:POINTS:AUTO?"
30  ENTER 707;State
40  PRINT State
50  END
```

See Also

:WAVeform:DATA?

RESPonse

Command

```
:ACQuire:RESPonse {FLATmagnitude | NORinging}
```

This command is available on the 8 channel 90000A Series oscilloscopes (91308A, 91208A, and 90808A). The Flat Magnitude filter is the default one and is the filter typically used on Infiniium oscilloscopes. The No Ringing filter eliminates all ringing (preshoot or overshoot) caused by the oscilloscope's response. Therefore, any ringing you see in the displayed signal is actually in your signal and is not caused by the oscilloscope. The main drawback to using the No Ringing Filter is the decrease in bandwidth. Please consult the Flat Magnitude / No Ringing Filters topic in the help system for specific information regarding the decrease in bandwidth.

Example

This example turns on the No Ringing filter.

```
10  OUTPUT 707;":ACQUIRE:RESPonse NORinging"
20  END
```

Query

```
:ACQuire:RESPonse?
```

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

Returned Format

```
[:ACQuire:POINts:AUTO] {FLATmagnitude | NORinging}<NL>
```

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:RESPonse?"
30  ENTER 707;State
40  PRINT State
50  END
```

SEGMented:COUNt

Command

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

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

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

Example

This example sets the segmented memory count control to 1000.

```
10  OUTPUT 707;":ACQUIRE:SEGMented:COUNt 1000"
20  END
```

Query

```
:ACQuire:SEGMented:COUNt?
```

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

Returned Format

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

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:SEGMents:COUNt?"
30  ENTER 707;Segments
40  PRINT Segments
50  END
```

SEGMented:INDex

Command

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

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

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

Example

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

```
10  OUTPUT 707;":ACQUIRE:SEGMented:INDex 1000"
20  END
```

Query

```
:ACQuire:SEGMented:INDex?
```

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

Returned Format

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

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF”
20  OUTPUT 707;":ACQUIRE:SEGMents:INDex?"
30  ENTER 707;Index
40  PRINT Index
50  END
```

SEGMented:TTAGs

Command

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

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

Example

This example turns the time tags on for segmented memory.

```
10  OUTPUT 707;":ACQUIRE:SEGMented:TTAGs ON"
20  END
```

Query

```
:ACQuire:SEGMented:TTAGs?
```

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

Returned Format

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

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:SEGMents:TTAGs?"
30  ENTER 707;timetags
40  PRINT timetags
50  END
```

SRATe (Sample RATe)

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

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

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

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

`<rate>` A real number representing the sample rate. You can send any value, but the value is rounded to the next fastest sample rate. For a list of available sample rate values see see “SRATe Sample Rate Tables” on page 8-23.

Interaction between :ACQuire:SRATe and :ACQuire:POINts

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

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

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

```
10  OUTPUT 707;":ACQUIRE:SRATE 250E+6"
20  END
```

Query `:ACQuire:SRATe?`

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

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

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

```
10  DIM Sample$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:SRATE?"
30  ENTER 707;Sample$
40  PRINT Sample$
50  END
```

SRATe Sample Rate Tables

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

Table 8-8

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

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

Table 8-9

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

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

Table 8-10

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

Table 8-11

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

SRATe:AUTO

Command

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

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

Example

This example changes the sampling rate to manual.

```
10  OUTPUT 707;":ACQUIRE:SRATE:AUTO OFF"
20  END
```

Query

```
:ACQuire:SRATe:AUTO?
```

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

Returned Format

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

Example

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF”
20  OUTPUT 707;":ACQUIRE:SRATE:AUTO?"
30  ENTER 707;Sample
40  PRINT Sample
50  END
```

9

Bus Commands

Bus Commands

The BUS commands only apply to the Oscilloscopes with serial data analyse option installed.

The :BUS modes and commands described in this chapter include:

- B1:TYPE

B1:TYPE

Command `:BUS:B1:TYPE <protocol>`

The BUS commands only apply to the Oscilloscopes with serial data analysis option installed.

The :BUS:B1:TYPE command sets the type of protocol being analyzed.

`<protocol>` {CAN | DVI | FIBRechannel | FLEXray | GEN8B10B | GENeric | IIC | INFiniband | MOST | PCIexpress | SAS | SATA | SPI | XAUI | MIPI}

Example This example sets the protocol type to FLEXray.

```
10 Output 707;"BUS:B1:TYPE FLEXRAY"
20 END
```

Query `:BUS:B1:TYPE?`

The :BUS:B1:TYPE? query returns the name of the protocol being used.

Return format `[:BUS:B1:TYPE] <protocol><NL>`

10

Calibration Commands

Calibration Commands

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

Oscilloscope Calibration

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

- Initiate the calibration from the “Utilities Calibration” menu.

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

See Also

The Oscilloscope’s Service Guide has more details about the calibration.

Probe Calibration

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

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

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

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

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

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

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

Calibration Commands

The commands in the CALibration subsystem allow you to change the output of the front-panel Aux Out connector, adjust the skew of channels, and check the status of calibration. These CALibration commands and queries are implemented in the Infiniium Oscilloscopes:

- OUTPut
- SKEW
- STATus?

OUTPut

Command

```
:CALibrate:OUTPut {{AC | TRIGOUT} | {DC,<dc_value>}}
```

The :CALibrate:OUTPut command sets the coupling frequency, trigger output pulse, and dc level of the calibrator waveform output through the front-panel Aux Out connector. To trigger other instruments, use the TRIGOUT setting to cause the oscilloscope to send a pulse when the trigger event occurs. The AC sets the Aux Out to be the probe compensation square wave (approximately 750 Hz).

`<dc_value>` A real number for the DC level value in volts, adjustable from -2.4 V to +2.4 V dc.

Example

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

```
10  OUTPUT 707;":CALIBRATE:OUTPUT DC,2.0"
20  END
```

Query

```
:CALibrate:OUTPut?
```

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

Returned Format

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

Example

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

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":CALIBRATE:OUTPUT?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

SKEW

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

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

`<N>` An integer, 1 - 4.

`<skew_value>` A real number, in seconds.

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

```
10  OUTPUT 707;":CALIBRATE:SKEW CHANNEL1,1E-6"
20  END
```

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

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

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

STATus?

Query `:CALibrate:STATus?`

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

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

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

11

Channel Commands

Channel Commands

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

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

These CHANnel commands and queries are implemented:

- BWLimit
- COMMonmode
- DIFFerential
- DIFFerential:SKEW
- DISPlay
- DISPlay:AUTO
- DISPlay:OFFSet
- DISPlay:RANge
- DISPlay:SCALe
- INPut
- ISIM:APPLy
- ISIM:BANDwidth
- ISIM:BWLimit
- ISIM:CONVolve
- ISIM:DEConvolve
- ISIM:DELay
- ISIM:SPAN

- ISIM:STATe
- LABel
- OFFSet
- PROBe
- PROBe:ATTenuation (only for the 1154A probe)
- PROBe:COUPling
- PROBe:EADapter (only for the 1153A, 1154A, and 1159A probes)
- PROBe:ECOupling (only for the 1153A, 1154A, and 1159A probes)
- PROBe:GAIN (only for the 1154A probe)
- PROBe:EXTernal
- PROBe:EXTernal:GAIN
- PROBe:EXTernal:OFFSet
- PROBe:EXTernal:UNITs
- PROBe:HEAD:ADD
- PROBe:HEAD:DELete
- PROBe:HEAD:SELect
- PROBe:ID?
- PROBe:SKEW
- PROBe:STYPe (only for 113xA series, 1168A, and 1169A probes)
- RANGe
- SCALe
- UNITs

BWLimit

Command

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

The :CHANnel<N>:BWLimit command controls the low-pass filter. When ON, the bandwidth of the specified channel is limited. The bandwidth filter can be used with either AC or DC coupling.

<N> An integer, 1 - 4

Example

This example sets the internal low-pass filter t “ON” for channel 1.

```
10 OUTPUT 707;"CHANNEL1:BWLimit ON"
20 END
```

Query

```
:CHANnel<N>:BWLimit?
```

The :CHANnel<N>:BWLimit? query returns the current state of the low-pass filter for the specified channel.

Returned Format

```
[:CHANnel<N>:BWLimit] {1 | 0}<NL>
```

Example

This example places the current setting of the low-pass filter in the variable Limit, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:BWLimit?"
30  ENTER 707;Limit
40  PRINT Limit
50  END
```

COMMonmode

Command

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

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

<N> An integer, 1 - 4

Example

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

```
10 OUTPUT 707;"CHANNEL1:DIFFerential ON"
20 END
```

Query

```
:CHANnel<N>:COMMonmode?
```

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

Returned Format

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

Example

This example places the current common mode setting of the channel 1 display in the variable Comm, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:COMMonmode?"
30  ENTER 707;Comm
40  PRINT Comm
50  END
```

DIFFerential

Command

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

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

<N> An integer, 1 - 4

Example

This example turns channel 1 differential on (channel 1 - channel 3).

```
10 OUTPUT 707;"CHANNEL1:DIFFerential ON"
20 END
```

Query

```
:CHANnel<N>:DIFFerential?
```

The :CHANnel<N>:DIFFerential? query returns whether the channel is in differential mode or not.

Returned Format

```
[:CHANnel<N>:DIFFerential] {1 | 0}<NL>
```

Example

This example places the current differential setting of the channel 1 display in the variable Diff, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DIFFerential?"
30  ENTER 707;Diff
40  PRINT Diff
50  END
```

DIFFerential:SKEW

Command

```
:CHANnel<N>:DIFFerential:SKEW <skew>
```

The :CHANnel<N>:DIFFerential:SKEW <skew> command sets the skew that is applied to the differential or common mode pair of channels.

`<skew>` A real number for the skew value

Example

This example sets the skew applied to the channel 1 - channel 3 differential channel to 10 μs.

```
10  OUTPUT 707;":CHANNEL1:DIFFerential:SKEW 10E-6"
20  END
```

Query

```
:CHANnel<N>:DIFFerential:SKEW?
```

The :CHANnel<N>:DIFFerential:SKEW? query returns the skew that is applied to the differential or common mode pair of channels.

Returned Format

```
[:CHANnel<N>:DIFFerential:SKEW] <skew_value><NL>
```

Example

This example places the current skew setting of the channel 1 - channel 3 differential channel in the variable skew, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DIFFerential:SKEW?"
30  ENTER 707;skew
40  PRINT skew
50  END
```

DISPlay

Command

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

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

<N> An integer, 1 - 4

Example

This example sets channel 1 display to on.

```
10 OUTPUT 707;"CHANNEL1:DISPLAY ON"
20 END
```

Query

```
:CHANnel<N>:DISPlay?
```

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

Returned Format

```
[:CHANnel<N>:DISPlay] {1 | 0}<NL>
```

Example

This example places the current setting of the channel 1 display in the variable Display, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DISPLAY?"
30  ENTER 707;Display
40  PRINT Display
50  END
```

DISPlay:AUTO

Command

```
:CHANnel<N>:DISPlay:AUTO {{ON | 1} | {OFF | 0}}
```

The :CHANnel<N>:DISPlay:AUTO command sets the differential and common mode display scale and offset to track the acquisition scale and offset.

<N> An integer, 1 - 4

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

Example

This example sets the channel 1 - channel 3 differential channel display scale and offset to track the acquisition scale and offset.

```
10 OUTPUT 707;"CHANNEL1:DISPLAY:AUTO ON"
20 END
```

Query

```
:CHANnel<N>:DISPlay:AUTO?
```

The :CHANnel<N>:DISPlay:AUTO? query returns whether or not the differential or common mode display scale and offset are tracking the acquisition scale and offset.

Returned Format

```
[:CHANnel<N>:DISPlay:AUTO] {1 | 0}<NL>
```

Example

This example places whether or not the channel 1 - channel 3 differential channel display scale and offset is tracking the acquisition scale and offset in the variable auto, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DISPLAY:AUTO?"
30  ENTER 707;auto
40  PRINT auto
50  END
```

DISPlay:OFFSet

Command `:CHANnel<N>:DISPlay:OFFSet <value>`

The :CHANnel<N>:DISPlay:OFFSet command sets the displayed offset of the selected channel. Setting the display range turns off display auto.

`<value>` A real number for the value variable

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

Example This example sets the displayed offset of channel 1 to

```
10 OUTPUT 707;"CHANNEL1:DISPLAY:RANGe 10e-6"
20 END
```

Query `:CHANnel<N>:DISPlay:OFFSet?`

The :CHANnel<N>:DISPlay:OFFSet? query returns the displayed offset for the selected channel.

Returned Format `[:CHANnel<N>:DISPlay:OFFSet] <value><NL>`

Example This example places the displayed offset of channel 1 in the variable offset, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DISPLAY:OFFSet?"
30  ENTER 707;offset
40  PRINT offset
50  END
```

DISPlay:RANGe

Command

```
:CHANnel<N>:DISPlay:RANGe <range>
```

The :CHANnel<N>:DISPlay:RANGe command sets the full scale vertical range of the display of the selected channel. Setting the display range turns off display auto.

`<range>` A real number for the range value

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

Example

This example sets the display range of the display of channel 1 to

```
10 OUTPUT 707;"CHANNEL1:DISPLAY:RANGe 10e-6"
20 END
```

Query

```
:CHANnel<N>:DISPlay:RANGe?
```

The :CHANnel<N>:DISPlay:RANGe? query returns the full scale vertical range of the display for the selected channel.

Returned Format

```
[:CHANnel<N>:DISPlay:RANGe] <range><NL>
```

Example

This example places the range of channel 1 in the variable range, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DISPLAY:RANGe?"
30  ENTER 707;range
40  PRINT range
50  END
```

DISPlay:SCALe

Command

```
:CHANnel<N>:DISPlay:SCALe <scale>
```

The :CHANnel<N>:DISPlay:SCALe command sets the displayed scale of the selected channel per division. Setting the display range turns off display auto.

`<scale>` A real number for the scale value

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

Example

This example sets the display scale of channel 1 per division to

```
10 OUTPUT 707;"CHANNEL1:DISPLAY:SCALe 10e-6"
20 END
```

Query

```
:CHANnel<N>:DISPlay:SCALe?
```

The :CHANnel<N>:DISPlay:SCALe? query returns the displayed scale of the selected channel per division.

Returned Format

```
[:CHANnel<N>:DISPlay:SCALe] <scale><NL>
```

Example

This example places the display scale of channel 1 in the variable scale, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DISPLAY:SCALe?"
30  ENTER 707;scale
40  PRINT scale
```

```
50  END
```

INPut

Command

```
:CHANnel<N>:INPut {DC50 | DCFifty}
```

The :CHANnel<N>:INPut command selects the input coupling and impedance for the specified channel and is always DC coupling, 50Ω input impedance. This command is provided for compatibility with other Infiniium oscilloscopes.

<N> An integer, 1 - 4.

Example

This example sets the channel 1 input to DC50.

```
10 OUTPUT 707;":CHANNEL1:INPut DC50"
20 END
```

Query

```
:CHANnel<N>:INPut?
```

The :CHANnel<N>:INPut? query returns the selected channel input parameter and is always DC coupling, 50Ω input impedance. .

Returned Format

```
[CHANnel<N>:INPut] DC50<NL>
```

Example

This example puts the current input for channel 1 in the string variable, Input$. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:INPUT?
30  ENTER 707;Input$
40  PRINT Input$
50  END
```

ISIM:APPLy

Command

```
:CHANnel<N>:ISIM:APPLy <transfer_funct_file>
```

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:APPLy command applies a pre-computed transfer function to the waveform. If InfiniiSim is in 2 port mode, the file must be a .tf2 file. If in 4 port mode, the file must be a .tf4 file. Use the ISIM:STATe command to enable InfiniiSim before issuing the APPLy command.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

`<N>` An integer, 1 - 4.

`<transfer_funct_file>` The .tf2 file name (if in 2 port mode) or .tf4 file (if in 4 port mode).

Example

This example applies the example.tf4 file to the waveform on channel 1.

```
10 OUTPUT 707;":CHANNEL1:ISIM:APPLy example.tf4"
20 END
```

Query

```
:CHANnel<N>:ISIM:APPLy?
```

The :CHANnel<N>:ISIM:APPLy? query returns the currently selected function file name when 2 port or 4 port mode is enabled.

Returned Format

```
[CHANnel<N>:ISIM:APPLy] <file_name><NL>
```

Example

This example puts the current transfer function file name in the variable file. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:ISIM:APPLy?
30  ENTER 707;file
40  PRINT file
50  END
```

ISIM:BANDwidth

Command `:CHANnel<N>:ISIM:BANDwidth <bw_value>`

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:BANDwidth command sets the Bandwidth Limit field in the InfiniiSim GUI to a desired value (sets the bandwidth limit cutoff frequency). The CHANnel<N>:ISIM:BWLimit command turns this feature on or off. Please refer to the InfiniiSim User's Guide on your oscilloscope or on Agilent.com for more explanation regarding this field.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

`<N>` An integer, 1 - 4.

`<bw_value>` The maximum value is the sample rate / 2. The minimum value is 1000 Hz.

Example This example sets the channel 1 input bandwidth limit cutoff frequency to 2 GHz.

```
10 OUTPUT 707;":CHANNEL1:ISIM:BANDwidth 2e9"
20 END
```

Query `:CHANnel<N>:ISIM:BANDwidth?`

The :CHANnel<N>:ISIM:BANDwidth? query returns the selected channel input's bandwidth limit cutoff frequency.

Returned Format `[CHANnel<N>:ISIM:BANDwidth] <parameter><NL>`

Example

This example puts the current input for channel 1 in the string variable, bwlimit. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:ISIM:BANDwidth?
30  ENTER 707;bwlimit
40  PRINT bwlimit
50  END
```

ISIM:BWLimit

Command

```
:CHANnel<N>:ISIM:BWLimit {{ON | 1} | {OFF | 0}}
```

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:BWLimit command activates or deactivates the Bandwidth Limit field in the InfiniiSim GUI. This field sets the bandwidth limit cutoff frequency. The CHANnel<N>:ISIM:BANDwidth command sets the value to be used when this field is activated. Please refer to the InfiniiSim User's Guide on your oscilloscope or on Agilent.com for more explanation regarding this field.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1 - 4.

Example

This example turns on the InfiniiSim bandwidth limit feature for channel 1.

```
10 OUTPUT 707;":CHANNEL1:ISIM:BWLimit ON"
20 END
```

Query

```
:CHANnel<N>:ISIM:BWLimit?
```

The :CHANnel<N>:ISIM:BWLimit? query returns the current state of the corresponding channel's InfiniiSim bandwidth limiting feature.

Returned Format

```
[CHANnel<N>:ISIM:BWLimit] {1 | 0}<NL>
```

Example

This example puts the current InfiniiSim bandwidth limit state for channel 1 in the string variable, limit. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:ISIM:BWLimit?
30  ENTER 707;limit
40  PRINT limit
50  END
```

ISIM:CONVolve

Command

```
:CHANnel<N>:ISIM:CONVolve <s_parameter_file>, {OFF | ON}
```

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:CONVolve command convolves the indicated S-parameter file with the waveform. This command only uses a single S21 component block. If a .s4p file is indicated, ports 1 and 2 are used assuming a 1-2, 3-4 port numbering for 4 port files. Optionally, include ON to flip the port numbering when reading the s-parameter file.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

`<N>` An integer, 1 - 4.

`<s_parameter_file>` The name of the s-parameter file.

Example

This example convolves the s-parameter file example.s2p with the waveform on channel 1.

```
10 OUTPUT 707;":CHANNEL1:ISIM:CONVolve example.s2p"
20 END
```

ISIM:DEConvolve

Command

```
:CHANnel<N>:ISIM:DEConvolve <s_parameter_file>,
{OFF | ON}
```

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:DEConvolve command deconvolves the indicated S-parameter file with the waveform. This command only uses a single S21 component block. If a .s4p file is indicated, ports 1 and 2 are used assuming a 1-2, 3-4 port numbering for 4 port files. Optionally, include ON to flip the port numbering when reading the s-parameter file.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

`<N>` An integer, 1 - 4.

`<s_parameter_file>` The name of the s-parameter file.

Example

This example deconvolves the s-parameter file example.s2p with the waveform on channel 1.

```
10 OUTPUT 707;":CHANNEL1:ISIM:DEConvolve example.s2p"
20 END
```

ISIM:DELay

Command

```
:CHANnel<N>:ISIM:DElay {OFF | ON}
```

This CHANnel command only applies if you have purchased the InfiniiSim software application.

If the :CHANnel<N>:ISIM:DELay command is turned on, the transfer function delay is included in the resultant waveform. Consult the InfiniiSim User's Guide in the Manuals section of the GUI help system for more information.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1 - 4.

Example

This example applies the transfer function delay in the resultant waveform.

```
10 OUTPUT 707;":CHANNEL1:ISIM:DELay ON"
20 END
```

Query

```
:CHANnel<N>:ISIM:DELay?
```

The :CHANnel<N>:ISIM:DELay? query returns the current state of the transfer function delay feature on the corresponding input channel.

Returned Format

```
[CHANnel<N>:ISIM:DELay] {OFF | ON}<NL>
```

Example

This example puts whether or not the transfer function delay is included in the resultant waveform for channel 1 in the string variable, delay. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:ISIM:DELay?
30  ENTER 707;delay
40  PRINT delay
50  END
```

ISIM:SPAN

Command `:CHANnel<N>:ISIM:SPAN <max_time_span>`

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:SPAN command sets the maximum time span control in the InfiniiSim Setup dialog box.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

`<N>` An integer, 1 - 4.

`<max_time_span>` A real number.

Example This example sets the maximum time span control to 100e-9.

```
10 OUTPUT 707;":CHANNEL1:ISIM:SPAN 100e-9"
20 END
```

Query `:CHANnel<N>:ISIM:SPAN?`

The :CHANnel<N>:ISIM:SPAN? query returns the current InfiniiSim filter maximum time span on the corresponding input channel.

Returned Format `[CHANnel<N>:ISIM:SPAN] <max_time_span><NL>`

Example This example puts the InfiniiSim filter's maximum time span value in the variable tspan. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
```

```
20  OUTPUT 707;":CHANNEL1:ISIM:SPAN?
30  ENTER 707;tspan
40  PRINT tspan
50  END
```

ISIM:STATe

Command

```
:CHANnel<N>:ISIM:STATe {OFF | PORT2 | PORT4}
```

This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:STATe command turns InfiniiSim on or off and sets whether 2 port or 4 port mode is being used (if it is turned on).

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1 - 4.

Example

This example turns on InfiniiSim for channel 1 and puts it in 2 port mode.

```
10 OUTPUT 707;":CHANNEL1:ISIM:STATe PORT2"
20 END
```

Query

```
:CHANnel<N>:ISIM:STATe?
```

The :CHANnel<N>:ISIM:STATe? query returns the current state of InfiniiSim on the corresponding input channel.

Returned Format

```
[CHANnel<N>:ISIM:STATe] {OFF | PORT2 | PORT4}<NL>
```

Example

This example puts the current InfiniiSim state for channel 1 in the string variable, mode. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:ISIM:STATe?
30  ENTER 707;mode
```

```
40  PRINT mode
50  END
```

OFFSet

Command

```
:CHANnel<N>:OFFSet <offset_value>
```

The :CHANnel<N>:OFFSet command sets the vertical value that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

`<N>` An integer, 1 - 4

`<offset_value>` A real number for the offset value at center screen. Usually expressed in volts, but it can also be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:UNITs command or the CHANnel<N>:PROBe:EXTernal:UNITs command.

Example

This example sets the offset for channel 1 to 0.125 in the current measurement units:

```
10  OUTPUT 707;":CHANNEL1:OFFSET 125E-3"
20  END
```

Query

```
:CHANnel<N>:OFFSet?
```

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

Returned Format

```
[CHANnel<N>:OFFSet] <offset_value><NL>
```

Example

This example places the offset value of the specified channel in the string variable, Offset$, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;"CHANNEL1:OFFSET?"
30 ENTER 707;Offset
40 PRINT Offset
50 END
```

LABel

Command

```
:CHANnel<N>:LABel <string>
```

The :CHANnel<N>:LABel command sets the channel label to the quoted string.

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

:CHANnel1 would refer to the Channel 1 - Channel 3 differential channel

:CHANnel2 would refer to the Channel 2 - Channel 4 differential channel

:CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel

:CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

`<N>` An integer, 1 - 4.

`<string>` A series of 16 or less characters as a quoted ASCII string

Example

This example sets the channel 1 label to Data.

```
10 OUTPUT 707;":CHANNEL1:LABel ""Data"""
20 END
```

Query

```
:CHANnel<N>:LABel?
```

The :CHANnel<N>:LABel? query returns the label of the specified channel.

Returned Format

```
[CHANnel<N>:LABel] <string><NL>
```

OFFSet

Command

```
:CHANnel<N>:OFFSet <offset_value>
```

The :CHANnel<N>:OFFSet command sets the vertical value that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

<N> An integer, 1 - 4

<offset_value> A real number for the offset value at center screen. Usually expressed in volts, but it can also be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:UNITs command or the CHANnel<N>:PROBe:EXTernal:UNITs command.

Example

This example sets the offset for channel 1 to 0.125 in the current measurement units:

```
10  OUTPUT 707;":CHANNEL1:OFFSET 125E-3"
20  END
```

Query

```
:CHANnel<N>:OFFSet?
```

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

Returned Format

```
[CHANnel<N>:OFFSet] <offset_value><NL>
```

Example

This example places the offset value of the specified channel in the string variable, Offset$, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;"CHANNEL1:OFFSET?"
30 ENTER 707;Offset
40 PRINT Offset
50 END
```

PROBe

Command

```
:CHANnel<N>:PROBe <attenuation_factor>[,{RATio | DECibel}]
```

The :CHANnel<N>:PROBe command sets the probe attenuation factor and the units (ratio or decibels) for the probe attenuation factor for a user-defined probe.

The DECibel and RATio parameters also set the "mode" for the probe attenuation. These parameters, along with attenuation factor, determine the scaling of the display and affect automatic measurements and trigger levels.

This mode also determines the units (ratio or decibels) that may be used for a subsequent command.

`<N>` An integer, 1-4

`<attenuation_factor>` A real number from 0.0001 to 1000 for the RATio attenuation units or from –80 dB to 60 dB for the DECibel attenuation units.

Example

This example sets the probe attenuation factor for a 10:1 probe on channel 1 in ratio units.

```
10  OUTPUT 707;":CHANNEL1:PROBE 10,RAT"
20  END
```

Query

```
:CHANnel<N>:PROBe?
```

The :CHANnel<N>:PROBe? query returns the current probe attenuation setting and units for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe] <attenuation>,{RATio | DECibel}<NL>
```

Example

This example places the current attenuation setting for channel 1 in the string variable, Atten$, then the program prints the contents.

```
10  DIM Atten$[50]!Dimension variable
20  OUTPUT 707;":CHANNEL1:PROBE?
30  ENTER 707;Atten$
40  PRINT Atten$
50  END
```

If you use a string variable, the query returns the attenuation value and the factor (decibel or ratio). If you use an integer variable, the query returns the attenuation value. You must then read the attenuation units into a string variable.

PROBe:ATTenuation

Command `:CHANnel<N>:PROBe:ATTenuation {DIV1 | DIV10}`

This command is only valid for the 1154A probe.

The :CHANnel<N>:PROBe:ATTenuation command sets the 1154A probe's input amplifier attenuation. If the 1154A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 - 4

Example This example sets the probe attenuation for channel 1 to divide by 10.

```
10  OUTPUT 707;":CHANNEL1:PROBE:ATTENUATION DIV10"
20  END
```

Query `:CHANnel<N>:PROBe:ATTenuation?`

The :CHANnel<N>:PROBe:ATTenuation? query returns the current 1154A probe input amplifier attenuation setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:ATTenuation] {DIV1 | DIV10}<NL>`

PROBe:COUPling

Command

```
:CHANnel<N>:PROBe:COUPling {DC | AC}
```

The :CHANnel<N>:PROBe:COUPling command sets the coupling to either AC or DC.

<N> An integer, 1 - 4

Example

This example sets the probe coupling for channel 1 to AC.

```
10  OUTPUT 707;":CHANNEL1:PROBE:COUPling AC"
20  END
```

Query

```
:CHANnel<N>:PROBe:COUPling?
```

The :CHANnel<N>:PROBe:COUPling? query returns the current probe coupling setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:COUPling] {DC | AC}<NL>
```

PROBe:EADapter

Command

```
:CHANnel<N>:PROBe:EADapter {NONE | DIV10 |
DIV20 | DIV100}
```

This command is valid only for the 1153A, 1154A, and 1159A probes.

The :CHANnel<N>:PROBe:EADapter command sets the probe external adapter control. The 1153A, 1154A, and 1159A probes have external adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the EADapter command to set the external adapter control to match the adapter connected to your probe as follows.

Parameter	Description
NONE	**Use this setting when there is no adapter connected to the end of your probe.**
DIV10	**Use this setting when you have a divide by 10 adapter connected to the end of your probe.**
DIV20	**Use this setting when you have a divide by 20 adapter connected to the end of your probe. (1159A)**
DIV100	**Use this setting when you have a divide by 100 adapter connected to the end of your probe.(1153A only)**

If an 1153A, 1154A, or 1159A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 - 4

Example

This example sets the external adapter for channel 1 to divide by 10:

```
10  OUTPUT 707;":CHANNEL1:PROBE:EADAPTER DIV10"
20  END
```

Query

```
:CHANnel<N>:PROBe:EADapter?
```

The :CHANnel<N>:PROBe:EADapter? query returns the current external adapter value for the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:EDApter] {NONE | DIV10 | DIV20 | DIV100}<NL>
```

Example

This example places the external adapter value of the specified channel in the string variable, Adapter$, then prints the contents of the variable to the computer's screen.

```
10  DIM Adapter$[50]!Dimension variable
20  OUTPUT 707;":CHANNEL1:PROBE:EADAPTER?
30  ENTER 707;Adapter$
40  PRINT Adapter$
50  END
```

PROBe:ECOupling

Command

```
:CHANnel<N>:PROBe:ECOupling {NONE | AC}
```

This command is valid only for the 1153A, 1154A, and 1159A probes.

The :CHANnel<N>:PROBe:ECOupling command sets the probe external coupling adapter control. The 1154A and 1159A probes have external coupling adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the ECOupling command to set the external coupling adapter control to match the adapter connected to your probe as follows.

Parameter	Description
NONE	**Use this setting when there is no adapter connected to the end of your probe.**
AC	**Use this setting when you have an ac coupling adapter connected to the end of your probe.**

If an 1153A, 1154A, or 1159A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 - 4

Example

This example sets the external coupling adapter for channel 1 to ac:

```
10  OUTPUT 707;":CHANNEL1:PROBE:ECOUPLING AC"
20  END
```

Query

```
:CHANnel<N>:PROBe:ECOupling?
```

The :CHANnel<N>:PROBe:ECoupling? query returns the current external adapter coupling value for the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:ECOupling] {NONE | AC}<NL>
```

Example

This example places the external coupling adapter value of the specified channel in the string variable, Adapter$, then prints the contents of the variable to the computer's screen.

```
10  DIM Adapter$[50]!Dimension variable
20  OUTPUT 707;":CHANNEL1:PROBE:ECOUPLING?
30  ENTER 707;Adapter$
40  PRINT Adapter$
50  END
```

PROBe:EXTernal

Command

```
:CHANnel<N>:PROBe:EXTernal {{ON | 1} | {OFF | 0}}
```

The :CHANnel<N>:PROBe:EXTernal command sets the external probe mode to on or off.

<N> An integer, 1 - 4

Example

This example sets channel 1 external probe mode to on.

```
10 OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL ON"
20 END
```

Query

```
:CHANnel<N>:PROBe:EXTernal?
```

The :CHANnel<N>:PROBe:EXTernal? query returns the current external probe mode for the specified channel.

Returned Format

```
[:CHANnel<N>:PROBe:EXTernal] {1 | 0}<NL>
```

Example

This example places the current setting of the external probe mode on channel 1 in the variable Mode, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL?"
30  ENTER 707;Mode
40  PRINT Mode
50  END
```

PROBe:EXTernal:GAIN

Command

```
:CHANnel<N>:PROBe:EXTernal:GAIN
<gain_factor>[,{RATio | DECibel}]
```

CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect.

The :CHANnel<N>:PROBe:EXTernal:GAIN command sets the probe external scaling gain factor and, optionally, the units for the probe gain factor. The reference factors that are used for scaling the display are changed with this command, and affect automatic measurements and trigger levels.

The RATio or DECibel also sets the mode for the probe attenuation and also determines the units that may be used for a subsequent command. For example, if you select RATio mode, then the attenuation factor must be given in ratio gain units. In DECibel mode, you can specify the units for the argument as “dB”.

`<N>` An integer, 1 - 4

`<gain_factor>` A real number from 0.001 to 10000 for the RATio gain units, or from –60 dB to 80 dB for the DECibel gain units.

Example

This example sets the probe external scaling gain factor for channel 1 to 10.

```
10  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL ON"
20  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL:GAIN 10,RATIO"
30  END
```

Query

```
:CHANnel<N>:PROBe:EXTernal:GAIN?
```

The :CHANnel<N>:PROBe:EXTernal:GAIN? query returns the probe external gain setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:EXTernal:GAIN] <gain_factor><NL>
```

Example

This example places the external gain value of the probe on the specified channel in the variable, Gain, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL ON"
20  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL:GAIN?"
30  ENTER 707;Gain
40  PRINT Gain
50  END
```

PROBe:EXTernal:OFFSet

Command

```
:CHANnel<N>:PROBe:EXTernal:OFFSet <offset_value>
```

CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect.

The :CHANnel<N>:PROBe:EXTernal:OFFSet command sets the external vertical value for the probe that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

When using the 113xA series probes, the CHANnel<N>:PROBe:STYPe command determines how the offset is applied. When CHANnel<N>:PROBe:STYPe SINGle is selected, the :CHANnel<N>:PROBe:EXTernal:OFFset command changes the offset value of the probe amplifier. When CHANnel<N>:PROBe:STYPe DIFFerential is selected, the :CHANnel<N>:PROBe:EXTernal:OFFSet command changes the offset value of the channel amplifier.

`<N>` An integer, 1 - 4

`<offset_value>` A real number for the offset value at center screen. Usually expressed in volts, but can be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:PROBe:EXTernal:UNITs command.

Example

This example sets the external offset for the probe on channel 1 to 0.125 in the current measurement units:

```
10  OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL ON"
20  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL:OFFSET 125E-3"
30  END
```

Query

```
:CHANnel<N>:EXTernal:PROBe:OFFSet?
```

The :CHANnel<N>:PROBe:EXTernal:OFFSet? query returns the current external offset value for the probe on the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:EXTernal:OFFSet] <offset_value><NL>
```

Example

This example places the external offset value of the probe on the specified channel in the variable, Offset, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL ON"
30 OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL:OFFSET?"
40 ENTER 707;Offset
50 PRINT Offset
60 END
```

PROBe:EXTernal:UNITs

Command

```
:CHANnel<N>:PROBe:EXTernal:UNITs {VOLT | AMPere | WATT | UNKNown}
```

CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect. UNITs can also be set using the CHANnel<N>:UNITs command.

The :CHANnel<N>:PROBe:EXTernal:UNITs command sets the probe external vertical units on the specified channel. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel probe external commands and channel commands (such as :CHANnel<N>:PROBe:EXTernal:OFFSet and :CHANnel<N>:RANGe). See the Probe Setup dialog box for more information.

<N> An integer, 1 - 4

Example

This example sets the external units for the probe on channel 1 to amperes.

```
10  OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL ON"
20  OUTPUT 707;":CHANNEL1:PROBE:EXTERNAL:UNITS AMPERE"
30  END
```

Query

```
:CHANnel<N>:PROBe:EXTernal:UNITs?
```

The :CHANnel<N>:PROBe:EXTernal:UNITs? query returns the current external units setting for the probe on the specified channel.

Returned Format

```
[:CHANnel<N>:PROBe:EXTernal:UNITs] {VOLT | AMPere | WATT |
UNKNown}<NL>
```

Example

This example places the external vertical units for the probe on the specified channel in the string variable, Units$, then prints the contents of the variable to the computer's screen.

```
10 DIM Units$[50]
20 OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL ON"
30 OUTPUT 707;"CHANNEL1:PROBE:EXTERNAL:UNITS?"
40 ENTER 707;Units$
50 PRINT Units$
60 END
```

PROBe:HEAD:ADD

Command

```
:CHANnel<N>:PROBe:HEAD:ADD "head", ["label"]
```

The :CHANnel<N>:PROBe:HEAD:ADD command adds an entry to the list of probe heads.

`<N>` An integer, 1 - 4

`"head"` A quoted string matching the probe head model such as “N5381A”, “E2678A”, etc.

`"label"` An optional quoted string for the head label.

Example

This example adds the probe head N5381A to the list of probe heads for channel 1.

```
10  OUTPUT 707;":CHANNEL1:PROBE:HEAD:ADD "N5381A""
20  END
```

Query

There is no query available for this command.

PROBe:HEAD:DELete ALL

Command

```
:CHANnel<N>:PROBe:HEAD:DELete ALL
```

The :CHANnel<N>:PROBe:HEAD:DELete ALL command deletes all the nodes in the list of probe heads except for one default probe head which remains after this command is executed.

<N> An integer, 1 - 4

Example

This example deletes the entire list of probe heads for channel 1 except for the default head.

```
10  OUTPUT 707;":CHANNEL1:PROBE:HEAD:DELete ALL"
20  END
```

Query

There is no query available for this command.

PROBe:HEAD:SELect

Command

```
:CHANnel<N>:PROBe:HEAD:SELect <head_list_number>
```

The :CHANnel<N>:PROBe:HEAD:SELect command selects the position number of the probe head being used from a list of possible probe head choices. Note that the actual probe head model number of label cannot be used to specify the probe head. Instead, its position in the list is used to indicate which probe head is being used.

`<N>` An integer, 1 - 4

`<head_list_number>` Specifies the position in the configure list. The entry at the top of the list starts at 1. Note that this command does not reference the list by label or model number because there can be duplicate entries in the list.

Example

This example sets the probe head for channel 1 to the first selection in the configuration list.

```
10  OUTPUT 707;":CHANNEL1:PROBE:HEAD:SELect 1"
20  END
```

Query

```
:CHANnel<N>:PROBe:HEAD:SELect?
```

The :CHANnel<N>:PROBe:HEAD:SELect? query returns a SCPI formatted string of the selected probe head.

PROBe:GAIN

Command

```
:CHANnel<N>:PROBe:GAIN {X1 | X10}
```

This command is valid only for the 1154A probe.

The :CHANnel<N>:PROBe:GAIN command sets the 1154A probe input amplifier gain.

If an 1154A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 - 4

Example

This example sets the probe gain for channel 1 to times 10.

```
10  OUTPUT 707;":CHANNEL1:PROBE:GAIN X10"
20  END
```

Query

```
:CHANnel<N>:PROBe:GAIN?
```

The :CHANnel<N>:PROBe:GAIN? query returns the current probe gain setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:GAIN] {X1 | X10}<NL>
```

PROBe:ID?

Query `:CHANnel<N>:PROBe:ID?`

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

`<N>` An integer, 1 - 4

Returned Format [:CHANnel<N>:PROBe:ID] <probe_id>

`<probe_id>` A string of alphanumeric characters. Some of the possible returned values are:

1131A	**1132A**	**1134A**
1152A	**1154A**	**1156A**
1157A	**1158A**	**1159A**
1163A	**1168A**	**1169A**
AutoProbe	**E2621A**	**E2622A**
E2695A	**E2697A**	**N5380A**
N5381A	**N5382A**	**E2695A**
No Probe	**Unknown**	**User Defined Probe**

Example This example reports the probe type connected to channel 1, if one is connected.

```
10  OUTPUT 707;":CHANNEL1:PROBE:ID?"
20  END
```

PROBe:SKEW

Command

```
:CHANnel<N>:PROBe:SKEW <skew_value>
```

The :CHANnel<N>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. You can use the oscilloscope's probe skew control to remove timing differences between probes or cables on different channels.

`<N>` An integer, 1 - 4

`<skew_value>` A real number for the skew value, in the range -1 ms to +1 ms.

Example

This example sets the probe skew for channel 1 to 10 µs.

```
10  OUTPUT 707;":CHANNEL1:PROBE:SKEW 10E-6"
20  END
```

Query

```
:CHANnel<N>:PROBe:SKEW?
```

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

Returned Format

```
[:CHANnel<N>:PROBe:SKEW] <skew_value><NL>
```

PROBe:STYPe

Command

```
:CHANnel<N>:PROBe:STYPe {DIFFerential | SINGle}
```

This command is valid only for the 113xA series probes, 1168A probe, and 1169A probe.

The :CHANnel<N>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA series probes, 1168A probe, and 1169A probe. This setting determines how offset is applied.

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

<N> An integer, 1 - 4

Example

This example sets the probe mode to single-ended.

```
10  OUTPUT 707;":CHANNEL1:PROBE:STYPE SINGLE"
20  END
```

Query

```
:CHANnel<N>:PROBe:STYPe?
```

The :CHANnel<N>:PROBe:STYPe? query returns the current probe mode setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:STYPe] {DIFFerential | SINGle}<NL>
```

RANGe

Command

```
:CHANnel<N>:RANGe <range_value>
```

The :CHANnel<N>:RANGe command defines the full-scale vertical axis of the selected channel. It sets up acquisition and display hardware to display the waveform at a given range scale. The values represent the full-scale deflection factor of the vertical axis in volts. These values change as the probe attenuation factor is changed.

`<N>` An integer, 1 - 4

`<range_value>` A real number for the full-scale voltage of the specified channel number.

Example

This example sets the full-scale range for channel 1 to 500 mV.

```
10  OUTPUT 707;":CHANNEL1:RANGE 500E-3"
20  END
```

Query

```
:CHANnel<N>:RANGe?
```

The :CHANnel<N>:RANGe? query returns the current full-scale vertical axis setting for the selected channel.

Returned Format

```
[:CHANnel<N>:RANGe]<range_value><NL>
```

Example

This example places the current range value in the number variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":CHANNEL1:RANGE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

SCALe

Command `:CHANnel<N>:SCALe <scale_value>`

The :CHANnel<N>:SCALe command sets the vertical scale, or units per division, of the selected channel. This command is the same as the front-panel channel scale.

`<N>` An integer, 1 - 4

`<scale_value>` A real number for the vertical scale of the channel in units per division.

Example This example sets the scale value for channel 1 to 500 mV/div.

```
10  OUTPUT 707;":CHANNEL1:SCALE 500E-3"
20  END
```

Query `:CHANnel<N>:SCALe?`

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

Returned Format `[:CHANnel<N>:SCALe] <scale_value><NL>`

Example This example places the current scale value in the number variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":CHANNEL1:SCALE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

UNITs

Command

```
:CHANnel<N>:UNITs {VOLT | AMPere | WATT | UNKNown}
```

UNITs can also be set using the CHANnel<N>:PROBe:EXTernal:UNITs command when CHANnel<N>:PROBe:EXTernal command has been set to ON.

The :CHANnel<N>:UNITs command sets the vertical units. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel commands (such as :CHANnel<N>:RANGe and :CHANnel<N>:OFFSet). See the Probe Setup dialog box for more information.

<N> An integer, 1 - 4

Example

This example sets the units for channel 1 to amperes.

```
10  OUTPUT 707;":CHANNEL1:UNITS AMPERE"
20  END
```

Query

```
:CHANnel<N>:UNITs?
```

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

Returned Format

```
[:CHANnel<N>:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>
```

Example

This example places the vertical units for the specified channel in the string variable, Units$, then prints the contents of the variable to the computer's screen.

```
10 DIM Units$[50]
20 OUTPUT 707;"CHANNEL1:UNITS?"
30 ENTER 707;Units$
40 PRINT Units$
50 END
```

Common Commands

Common Commands

Common commands are defined by the IEEE 488.2 standard. They control generic device functions that are common to many different types of instruments. Common commands can be received and processed by the oscilloscope, whether they are sent over the GPIB as separate program messages or within other program messages.

These common commands and queries are implemented in the Infiniium Oscilloscopes:

- *CLS (Clear Status)
- *ESE (Event Status Enable)
- *ESR? (Event Status Register)
- *IDN? (Identification Number)
- *LRN? (Learn)
- *OPC (Operation Complete)
- *OPT? (Option)
- *PSC (Power-on Status Clear)
- *RCL (Recall)
- *RST (Reset)
- *SAV (Save)
- *SRE (Service Request Enable)
- *STB? (Status Byte)
- *TRG (Trigger)
- *TST? (Test)
- *WAI (Wait-to-Continue)

Receiving Common Commands

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

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

is received by the oscilloscope, the oscilloscope sets the acquire type, clears the status information, then sets the number of averages without leaving the selected subsystem.

Headers and Common Commands.

Headers are not prepended to common commands.

Status Registers

The following two status registers used by common commands have an enable (mask) register. By setting bits in the enable register, you can select the status information for use. Refer to the chapter, “Status Reporting,” for a complete discussion of status.

Table 12-1 Status and Enable Registers

Status Register	Enable Register
Event Status Register	Event Status Enable Register
Status Byte Register	Service Request Enable Register

*CLS (Clear Status)

Command

```
*CLS
```

The *CLS command clears all status and error registers.

Example

This example clears the status data structures of the oscilloscope.

```
10  OUTPUT 707;"*CLS"
20  END
```

See Also

Refer to the “Status Reporting” chapter for a complete discussion of status.

*ESE (Event Status Enable)

Command

```
*ESE <mask>
```

The *ESE command sets the Standard Event Status Enable Register bits.

`<mask>` An integer, 0 to 255, representing a mask value for the bits to be enabled in the Standard Event Status Register as shown in Table 12-2.

Example

This example enables the User Request (URQ) bit of the Standard Event Status Enable Register. When this bit is enabled and a front-panel key is pressed, the Event Summary bit (ESB) in the Status Byte Register is also set.

```
10  OUTPUT 707;"*ESE 64"
20  END
```

Query

```
*ESE?
```

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

Returned Format

```
<mask><NL>
```

`<mask>` An integer, +0 to +255 (the plus sign is also returned), representing a mask value for the bits enabled in the Standard Event Status Register as shown in Table 12-2.

Example

This example places the current contents of the Standard Event Status Enable Register in the numeric variable, Event. The value of the variable is printed on the computer's screen.

```
10  OUTPUT 707;"*ESE?"
20  ENTER 707;Event
30  PRINT Event
40  END
```

The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A "1" in the Standard Event Status Enable Register enables the corresponding bit in the Standard Event Status Register. A "0" in the enable register disables the corresponding bit.

Table 12-2 Standard Event Status Enable Register Bits

Bit	Weight	Enables	Definition
7	128	PON - Power On	Indicates power is turned on.
6	64		Not Used. Permanently set to zero.
5	32	CME - Command Error	Indicates whether the parser detected an error.
4	16	EXE - Execution Error	Indicates whether a parameter was out of range, or was inconsistent with the current settings.
3	8	DDE - Device Dependent Error	Indicates whether the device was unable to complete an operation for device-dependent reasons.
2	4	QYE - Query Error	Indicates if the protocol for queries has been violated.
1	2	RQC - Request Control	Indicates whether the device is requesting control.
0	1	OPC - Operation Complete	Indicates whether the device has completed all pending operations.

See Also Refer to the chapter, "Status Reporting," for a complete discussion of status.

*ESR? (Event Status Register)

Query

```
*ESR?
```

The *ESR? query returns the contents of the Standard Event Status Register. Reading this register clears the Standard Event Status Register, as does a *CLS.

Returned Format

```
<status><NL>
```

`<status>` An integer, 0 to 255, representing the total bit weights of all bits that are high at the time you read the register.

Example

This example places the current contents of the Standard Event Status Register in the numeric variable, Event, then prints the value of the variable to the computer's screen.

```
10  OUTPUT 707;"*ESR?"
20  ENTER 707;Event
30  PRINT Event
40  END
```

Table 12-3 lists each bit in the Event Status Register and the corresponding bit weights.

Table 12-3 Standard Event Status Register Bits

Bit	Bit Weight	Bit Name	Condition
7	128	PON	1 = OFF to ON transition has occurred.
6	64		Not Used. Permanently set to zero.
5	32	CME	0 = no command errors. 1 = a command error has been detected.
4	16	EXE	0 = no execution error. 1 = an execution error has been detected.
3	8	DDE	0 = no device-dependent errors. 1 = a device-dependent error has been detected.
2	4	QYE	0 = no query errors. 1 = a query error has been detected.
1	2	RQC	0 = request control - NOT used - always 0.
0	1	OPC	0 = operation is not complete. 1 = operation is complete.
	0 = False = Low		1 = True = High

*IDN? (Identification Number)

Query `*IDN?`

The *IDN? query returns the company name, oscilloscope model number, serial number, and software version by returning this string:

```
Agilent Technologies,<Model #>,<USXXXXXXXX>,<Rev #>
[,<Options>]
```

`<Model #>` Specifies the model number of the oscilloscope.

`<USXXXXXXXX>` Specifies the serial number of the oscilloscope. The first four digits and letter are the serial prefix, which is the same for all identical oscilloscopes. The last five digits are the serial suffix, which is assigned sequentially, and is different for each oscilloscope.

`<Rev #>` Specifies the software version of the oscilloscope, and is the revision number.

`<Options>` Comma separated list of the installed options.

Returned Format `Agilent Technologies,DSO80804B,USXXXXXXXX,A.XX.XX`

Example This example places the oscilloscope's identification information in the string variable, Identify$, then prints the identification information to the computer's screen.

```
10  DIM Identify$[50]!dimension variable
20  OUTPUT 707;"*IDN?"
30  ENTER 707;Identify$
40  PRINT Identify$
50  END
```

*LRN? (Learn)

Query

```
*LRN?
```

The *LRN? query returns a block of data that contains the oscilloscope's current setup. You can store the oscilloscope's setup and send it back to the oscilloscope at a later time. This block of setup data should be sent to the oscilloscope just as it is. It works because of its embedded ":SYST:SET" header.

Returned Format

```
:SYST:SET<setup><NL>
```

<setup> This is a definite-length, arbitrary block response specifying the current oscilloscope setup. The block size is subject to change with different firmware revisions.

Example

This example sets the oscilloscope's address and asks for the block of setup data, then determines the length according to the IEEE 488.2 block specification. It then reads the block of setup data and the last EOF character.

```
10  ! Set up the oscilloscope's address and
20  ! ask for the learn string...
30  ASSIGN @Scope TO 707
40  OUTPUT @Scope:"*LRN?"
50  !
60  ! Search for the # sign.
70  !
80  Find_pound_sign: !
90  ENTER @Scope USING "#,A";Thischar$
100 IF Thischar$<>"#" THEN Find_pound_sign
110 !
120 ! Determine the string length according
130 ! to the IEEE 488.2 # block spec.
140 ! Read the string then the last EOF char.
150 !
160 ENTER @Scope USING "#,D";Digit_count
170 ENTER @Scope USING
"#,"&VAL$(Digit_count)&"D";Stringlength
180 ALLOCATE Learn_string$[Stringlength+1]
190 ENTER @Scope USING "-K";Learn_string$
200 OUTPUT 707;":syst:err?"
210 ENTER 707;Errornum
220 PRINT "Error Status=";Errornum
```

See Also

:SYSTem:SETup command and query. When HEADers is ON and LONGform is OFF, the :SYSTem:SETup command performs the same function as the *LRN? query. However, *LRN and SETup block setup data are not interchangeable.

***LRN? Returns Prefix to Setup Block**

The *LRN? query always returns :SYST:SET as a prefix to the setup block. The :SYSTem:HEADer command has no effect on this response.

*OPC (Operation Complete)

Command

```
*OPC
```

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

Example

This example sets the operation complete bit in the Standard Event Status Register when the DIGitize operation is complete.

```
10  OUTPUT 707;":DIGITIZE CHANNEL1;*OPC"
20  END
```

Query

```
*OPC?
```

The *OPC? query places an ASCII character “1” in the oscilloscope's output queue when all pending selected device operations have finished.

Returned Format

```
1<NL>
```

Example

This example places an ASCII character “1” in the oscilloscope's output queue when the AUToscale operation is complete. Then the value in the output queue is placed in the numeric variable “Complete.”

```
10  OUTPUT 707;":AUTOSCALE;*OPC?"
20  ENTER 707;Complete
30  PRINT Complete
40  END
```

The *OPC? query allows synchronization between the computer and the oscilloscope by using the message available (MAV) bit in the Status Byte, or by reading the output queue. Unlike the *OPC command, the *OPC query does not affect the OPC Event bit in the Standard Event Status Register.

*OPT? (Option)

Query

```
*OPT?
```

The *OPT? query returns a string with a list of installed options. If no options are installed, the string will have a 0 as the first character.

The length of the returned string may increase as options become available in the future. Once implemented, an option name will be appended to the end of the returned string, delimited by a comma.

Returned Format

```
[002,EZP,EZJ,SDA,LSS,ABD,ABC,ABB,NRD,ERC,AIP,PCI1,ETH,DVI,
HDM,B30,CAN,SA1,DDR]<NL>
```

```
See on-line help system in the Help/About dialog box for the
installed options list.
```

Example

This example places all options into the string variable, Options$, then prints the option name to the computer's screen.

```
10  DIM Options$[100]
20  OUTPUT 707;"*OPT?"
30  ENTER 707;Options$
40  PRINT Options$
50  END
```

*PSC (Power-on Status Clear)

Command

```
*PSC {{ON|1} | {OFF|0}}
```

The *PSC command determines whether or not the SRQ line is set upon the completion of the oscilloscope's boot process. When the *PSC flag is set to 1, the Power On (PON) bit of the Standard Event Status Register is 0 during the boot process. When the *PSC flag is set to 0, the PON bit is set to a 1 during the boot process.

When the *PSC flag is set to 0, the Standard Event Status Enable Register must be set to 128 decimal and the Service Request Enable Register must be set to 32 decimal. This allows the Power On (PON) bit to set the SRQ line when the oscilloscope is ready to receive commands.

If you are using a LAN interface rather than a GPIB interface, it is not possible to receive the SRQ during the boot process.

Example

This example sets the *PSC flag to 0 which sets the SRQ line during the boot process.

```
10 OUTPUT 707;"*PSC 0;*SRE 32;*ESE 128"
20 END
```

Query

The *PSC? query returns the value of the *PSC flag.

Returned Format

```
1<NL>
```

Example

This example places the *PSC flag into the integer variable Pscflag.

```
10 OUTPUT 707;"*PSC?"
20 ENTER 707;Pscflag
30 PRINT Pscflag
40 END
```

*RCL (Recall)

Command

```
*RCL <register>
```

The *RCL command restores the state of the oscilloscope to a setup previously stored in the specified save/recall register. An oscilloscope setup must have been stored previously in the specified register. Registers 0 through 9 are general-purpose registers and can be used by the *RCL command.

`<register>` An integer, 0 through 9, specifying the save/recall register that contains the oscilloscope setup you want to recall.

Example

This example restores the oscilloscope to the oscilloscope setup stored in register 3.

```
10  OUTPUT 707;"*RCL 3"
20  END
```

See Also

*SAV (Save). An error message appears on the oscilloscope's display if nothing has been previously saved in the specified register.

*RST (Reset)

Command

```
*RST
```

The *RST command performs a default setup which is the same as pressing the oscilloscope front panel default key.

Example

This example resets the oscilloscope to a known state.

```
10  OUTPUT 707;"*RST"
20  END
```

The default values for all of the Infiniium controls is located in the Infiniium Help System under Default Setup.

*SAV (Save)

Command `*SAV <register>`

The *SAV command stores the current state of the oscilloscope in a save register.

`<register>` An integer, 0 through 9, specifying the register used to save the current oscilloscope setup.

Example This example stores the current oscilloscope setup to register 3.

```
10  OUTPUT 707;"*SAV 3"
20  END
```

See Also *RCL (Recall).

*SRE (Service Request Enable)

Command `*SRE <mask>`

The *SRE command sets the Service Request Enable Register bits. By setting the *SRE, when the event happens, you have enabled the oscilloscope's interrupt capability. The oscilloscope will then do an SRQ (service request), which is an interrupt.

`<mask>` An integer, 0 to 255, representing a mask value for the bits to be enabled in the Service Request Enable Register as shown in Table 12-4.

Example This example enables a service request to be generated when a message is available in the output queue. When a message is available, the MAV bit is high.

```
10  OUTPUT 707;"*SRE 16"
20  END
```

Query `*SRE?`

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

Returned Format `<mask><NL>`

`<mask>` An integer, 0 to 255, representing a mask value for the bits enabled in the Service Request Enable Register.

Example This example places the current contents of the Service Request Enable Register in the numeric variable, Value, then prints the value of the variable to the computer's screen.

```
10  OUTPUT 707;"*SRE?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A “1” in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A “0” disables the bit.

Table 12-4 Service Request Enable Register Bits

Bit	Weight	Enables
7	128	OPER - Operation Status Register
6	64	Not Used
5	32	ESB - Event Status Bit
4	16	MAV - Message Available
3	8	Not Used
2	4	MSG - Message
1	2	USR - User Event Register
0	1	TRG - Trigger

*STB? (Status Byte)

Query

```
*STB?
```

The *STB? query returns the current contents of the Status Byte, including the Master Summary Status (MSS) bit. See Table 12-5 for Status Byte Register bit definitions.

Returned Format

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

<value> An integer, 0 to 255, representing a mask value for the bits enabled in the Status Byte.

Example

This example reads the contents of the Status Byte into the numeric variable, Value, then prints the value of the variable to the computer's screen.

```
10  OUTPUT 707;"*STB?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

In response to a serial poll (SPOLL), Request Service (RQS) is reported on bit 6 of the status byte. Otherwise, the Master Summary Status bit (MSS) is reported on bit 6. MSS is the inclusive OR of the bitwise combination, excluding bit 6, of the Status Byte Register and the Service Request Enable Register. The MSS message indicates that the oscilloscope is requesting service (SRQ).

Table 12-5 Status Byte Register Bits

Bit	Bit Weight	Bit Name	Condition
7	128	OPER	0 = no enabled operation status conditions have occurred 1 = an enabled operation status condition has occurred
6	64	RQS/MSS	0 = oscilloscope has no reason for service 1 = oscilloscope is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition has occurred
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8	---	0 = not used
2	4	MSG	0 = no message has been displayed 1 = message has been displayed
1	2	USR	0 = no enabled user event conditions have occurred 1 = an enabled user event condition has occurred
0	1	TRG	0 = no trigger has occurred 1 = a trigger occurred
	0 = False = Low		1 = True = High

*TRG (Trigger)

Command

```
*TRG
```

The *TRG command has the same effect as the Group Execute Trigger message (GET) or RUN command. It acquires data for the active waveform display, if the trigger conditions are met, according to the current settings.

Example

This example starts the data acquisition for the active waveform display according to the current settings.

```
10  OUTPUT 707;"*TRG"
20  END
```

Trigger Conditions Must Be Met

When you send the *TRG command in Single trigger mode, the trigger conditions must be met before the oscilloscope will acquire data.

*TST? (Test)

Query

```
*TST?
```

The *TST? query causes the oscilloscope to perform a self-test, and places a response in the output queue indicating whether or not the self-test completed without any detected errors. Use the :SYSTem:ERRor command to check for errors. A zero indicates that the test passed and a non-zero indicates the self-test failed.

Disconnect Inputs First

You must disconnect all front-panel inputs before sending the *TST? command.

Returned Format

```
<result><NL>
```

<result> 0 for pass; non-zero for fail.

Example

This example performs a self-test on the oscilloscope and places the results in the numeric variable, Results. The program then prints the results to the computer's screen.

```
10  OUTPUT 707;"*TST?"
20  ENTER 707;Results
30  PRINT Results
40  END
```

If a test fails, refer to the troubleshooting section of the service guide.

Expanded Error Reporting

The :SELFtest:SCOPETEST command has expanded error reporting. Instead of using *TST?, Agilent recommends that you use the :SELFtest:SCOPETEST command. In either case, be sure you disconnect all front-panel inputs before sending the *TST? command.

*WAI (Wait)

Command

```
*WAI
```

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

Example

```
Output 707;"*WAI"
```

13

Disk Commands

Disk Commands

The DISK subsystem commands perform the disk operations as defined in the File menu. This allows saving and loading of waveforms and setups, as well as saving screen images to bitmap files.

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

Filenames are Not Case Sensitive.

The filename that you use is not case sensitive.

These DISK commands and queries are implemented in the Infiniium Oscilloscopes:

- CDIRectory
- COPY
- DELete
- DIRectory?
- LOAD
- MDIRectory
- PWD?
- SAVE:IMAGe
- SAVE:JITTer
- SAVE:MEASurements
- SAVE:SETup
- SAVE:WAVeform
- SEGMented

CDIRectory

Command

```
:DISK:CDIRectory "<directory>"
```

The :DISK:CDIRectory command changes the present working directory to the designated directory name. An error occurs when the requested directory does not exist. You can then view the error with the :SYSTem:ERRor? [{NUMBer | STRing}] query.

`<directory>` A character-quoted ASCII string, which can include the subdirectory designation. You must separate the directory name and any subdirectories with a backslash (\).

Example

This example sets the present working directory to C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

```
10 OUTPUT 707;":DISK:CDIRECTORY ""C:\Document and Settings\All
Users\Shared Documents\Infiniium\Data"""
20 END
```

COPY

Command

```
:DISK:COPY "<source_file>",”<dest_file>”
```

The :DISK:COPY command copies a source file from the disk to a destination file on the disk. An error is displayed on the oscilloscope screen if the requested file does not exist. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<source_file>`
`<dest_file>`

A character-quoted ASCII string which can include subdirectories with the name of the file.

Example

This example copies FILE1.SET to NEWFILE.SET on the disk.

```
10 OUTPUT 707;":DISK:COPY ""FILE1.SET"",””NEWFILE.SET”””
20 END
```

DELete

Command

```
:DISK:DELete "<file_name>"
```

The :DISK:DELete command deletes a file from the disk. An error is displayed on the oscilloscope screen if the requested file does not exist. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<file_name>` A character-quoted ASCII string which can include subdirectories with the name of the file.

Example

This example deletes FILE1.SET from the disk.

```
10 OUTPUT 707;":DISK:DELETE ""FILE1.SET"""
20 END
```

DIRectory?

Query `:DISK:DIRectory? ["<directory>"]`

The :DISK:DIRectory? query returns the requested directory listing. Each entry is 63 bytes long, including a carriage return and line feed. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<directory>` The list of filenames and directories.

Returned Format `[:DISK:DIRectory]<n><NL><directory>`

`<n>` The specifier that is returned before the directory listing, indicating the number of lines in the listing.

`<directory>` The list of filenames and directories. Each line is separated by a <NL>.

Example This example displays a number, then displays a list of files and directories in the current directory. The number indicates the number of lines in the listing.

```
10  DIM A$[80]
20  INTEGER Num_of_lines
30  OUTPUT 707;":DISK:DIR?"
40  ENTER 707;Num_of_lines
50  PRINT Num_of_lines
60  FOR I=1 TO Num_of_lines
70  ENTER 707;A$
80  PRINT A$
90  NEXT I
100 END
```

LOAD

Command

```
:DISK:LOAD "<file_name>"[,<destination>]
```

The :DISK:LOAD command restores a setup or a waveform from the disk. The type of file is determined by the filename suffix if one is present, or by the destination field if one is not present. You can load .WFM, .CSV, .TSV, .TXT, and .SET file types. The destination is only used when loading a waveform memory.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. You can use either .WFM, .CSV, .TSV, .TXT or .SET as a suffix after the filename. If no file suffix is specified, the default is .wfm.

The present working directory is assumed, or you can specify the entire path. For example, you can load the standard setup file "SETUP0.SET" using the command:

:DISK:LOAD "C:\Document and Settings\All Users\Shared Documents\Infiniium\Setups\SETUP0.SET"

Or, you can use :DISK:CDIRectory to change the present working directory to C:\Document and Settings\All Users\Shared Documents\Infiniium\Setups, then just use the file name ("SETUP0.SET", for example). The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<destination>` WMEMory<N>.

Where <N> is an integer from 1-4.

If a destination is not specified, waveform memory 1 is used.

Example

This example restores the waveform in FILE1.WFM to waveform memory 1.

```
10 OUTPUT 707;":DISK:LOAD ""FILE1.WFM"",WMEM1"
20 END
```

MDIRectory

Command

```
:DISK:MDIRectory "<directory>"
```

The :DISK:MDIRectory command creates a directory in the present working directory which has been set by the :DISK:CDIRectory command. If the present working directory has not been set by the :DISK:CDIRectory command, you must specify the full path in the <directory> parameter as shown in Example 1 below.

An error is displayed if the requested subdirectory does not exist.

`<directory>` A quoted ASCII string which can include subdirectories. You must separate the directory name and any subdirectories with a backslash (\).

Example 1

This example creates the directory CPROGRAMS in the C:\Document and Settings\All Users\Shared Documents\Infiniium\Data directory.

```
10 OUTPUT 707;":DISK:MDIRECTORY ""C:\Document and Settings\All
Users\Shared Documents\Infiniium\Data\CPROGRAMS"""
20 END
```

Example 2

This example creates the directory CPROGRAMS in the present working directory set by the :DISK:CDIRectory command.

```
10 OUTPUT 707;":DISK:MDIRECTORY ""CPROGRAMS"""
20 END
```

You can check your path with the :DISK:DIRectory? query.

PWD?

Query

```
:DISK:PWD?
```

The :DISK:PWD? query returns the name of the present working directory (including the full path). If the default path (C:\Document and Settings\All Users\Shared Documents\Infiniium\Data) has not been changed by the :DISK:CDIRectory command, the :DISK:PWD? query will return an empty string.

Returned Format

```
:DISK:PWD? <present_working_directory><NL>
```

Example

This example places the present working directory in the string variable Wdir?, then prints the contents of the variable to the computer's screen.

```
10 DIM Wdir$[200]
20 OUTPUT 707;":DISK:PWD?"
30 ENTER 707; Wdir$
40 PRINT Wdir$
50 END
```

SAVE:IMAGe

Command

```
:DISK:SAVE:IMAGe "<file_name>" [,<format>
[,{SCReen | GRATicule}
[,{ON | 1} | {OFF | 0}
[,{NORMal | INVert}]]]]
```

The DISK:SAVE:IMAGe command saves a screen image in BMP, GIF, TIF, PNG, or JPEG format. The extension is supplied by the oscilloscope depending on the selected file format. If you do not include the format in the command, the file is saved in the format which is shown in the Save Screen dialog box. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`ON | OFF` ON means that compression is on for the bitmap format (BMP). OFF means compression is off.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

`<format>` {BMP | GIF | TIF | JPEG | PNG}

Examples

```
OUTPUT 707;":DISK:SAVE:IMAGE " "FILE1" ",BMP,SCR,ON,INVERT"
or
OUTPUT 707;":DISK:SAVE:IMAGE " "FILE1" ",TIF,GRAT,ON"
or
OUTPUT 707;":DISK:SAVE:IMAGE " "FILE1" " "
```

SAVE:JITTer

Command `:DISK:SAVE:JITTer "<file_name>"`

The DISK:SAVE:JITTer command saves the jitter measurements shown in the RJDJ tab at the bottom of the oscilloscope screen along with the RJDJ histograms in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
OUTPUT 707;":DISK:SAVE:JITTER ""FILE1""
```

SAVE:MEASurements

Command

```
:DISK:SAVE:MEASurements "<file_name>"
```

The DISK:SAVE:MEASurements command saves the measurements shown in the measurements tab at the bottom of the oscilloscope screen in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
OUTPUT 707;":DISK:SAVE:MEASURMENTS ""FILE1"""
```

SAVE:SETup

Command

```
:DISK:SAVE:SETup "<file_name>"
```

The :DISK:SAVE:SETup command saves the current oscilloscope setup to a disk. The file will have a .set extension.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name. The default path is C:\SCOPE\SETUP.

Example

This example saves the channel 1 waveform to SETUP1 on the disk.

```
10 OUTPUT 707;":DISK:SAVE:SETUP ""SEUP1"""
20 END
```

SAVE:WAVeform

Command

```
:DISK:SAVE:WAVeform <source>,"<file_name>"
[,<format>[,<header>]]
```

The :DISK:SAVE:WAVeform command saves a waveform to a disk. If the source is ALL, all of the currently displayed waveforms are saved to the file. If you use a file extension as shown below in the <format> variable, then the type of file saved defaults to the extension type. If no format is specified and no extension is used, the file is saved in the INTernal format.

See the :WAVeform:VIEW command to determine how much data is saved.

`<source>` {ALL | CHANnel<N> | CLOCk | FUNCtion<N> | HISTogram | MTRend | MSPectrum | EQUalized | WMEMory<N>}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` An integer, 1 - 4

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

`<format>` {BIN | CSV | INTernal | TSV | TXT}

CSV stands for comma separated values and TSV stands for tab separated values.

The following file name extensions are used for the different formats.

BIN = file_name.bin

CSV = file_name.csv

INTernal = file_name.wfm

TSV = file_name.tsv

TXT = file_name.txt

`<header>` {{ON | 1} | {OFF | 0}}

Example

This example saves the channel 1 waveform to FILE1 on the disk in the CSV format with header on.

```
10 OUTPUT 707;":DISK:SAVE:WAVEFORM CHANNEL1,""FILE1"",CSV,ON"
20 END
```

CSV and TSV Header Format

Revision	Always 0 (zero).
Type	How the waveform was acquired: normal, raw, interpolate, average, or versus. When this field is read back into the scope, all modes, except versus, are converted to raw. The default value is raw.
Start	Starting point in the waveform of the first data point in the file. This is usually zero.
Points	The number of points in the waveform record. The number of points is set by the Memory Depth control. The default value is 1.
Count or Segments	For count, it is the number of hits at each time bucket in the waveform record when the waveform was created using an acquisition mode like averaging. For example, when averaging, a count of four would mean every waveform data point in the waveform record has been averaged at least four times. Count is ignored when it is read back into the scope. The default value is 0. Segments is used instead of Count when the data is acquired using the Segmented acquisition mode. This number is the total number of segments that were acquired.
XDispRange	The number of X display range columns (n) depends on the number of sources being stored. The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.

XDispOrg	The number of X display origin columns (n) depends on the number of sources being stored. The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.
XInc	The number of X increment columns (n) depends on the number of sources being store. The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.
XOrg	The number of X origin columns (n) depends on the number of sources being store. The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.
XUnits	The number of X units columns (n) depends on the number of sources being store. The X units is the unit of measure for each time value of the acquired data.
YDispRange	The number of Y display range columns (n) depends on the number of sources being store. The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. If the value is zero then no data has been acquired.
YDispOrg	The number of Y display origin columns (n) depends on the number of sources being store. The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. If the value is zero then no data has been acquired.

YInc	The number of Y increment columns (n) depends on the number of sources being store. The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. If the value is zero then no data has been acquired.
YOrg	The number of Y origin columns (n) depends on the number of sources being store. The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. If the value is zero then no data has been acquired.
YUnits	The number of Y units columns (n) depends on the number of sources being stored. The Y units is the unit of measure of each voltage value of the acquired waveform.
Frame	A string containing the model number and serial number of the scope in the format of MODEL#:SERIAL#.
Date	The date when the waveform was acquired. The default value is 27 DEC 1996.
Time	The time when the waveform was acquired. The default value is 01:00:00:00.
Max bandwidth	An estimation of the maximum bandwidth of the waveform. The default value is 0.
Min bandwidth	An estimation of the minimum bandwidth of the waveform. The default value is 0.
Time Tags	The Time Tags only occur when the data was acquired using the Segmented acquisition mode with time tags enabled and the file format is YValues. The number of columns depends on the number of Segments being saved.
Data	The data values follow this header entry.

BIN Header Format

File Header

There is only one file header in a binary file. The file header consists of the following information.

Cookie Two byte characters, AG, which indicates that the file is in the Agilent Binary Data file format.

Version Two bytes which represent the file version.

File Size An integer (4 byte signed) which is the number of bytes that are in the file.

Number of Waveforms An integer (4 byte signed) which is the number of waveforms that are stored in the file.

Waveform Header

The waveform header contains information about the type of waveform data that is stored following the waveform data header which is located after each waveform header. Because it is possible to store more than one waveform in the file, there will be a waveform header and a waveform data header for each waveform.

Header Size An integer (4 byte signed) which is the number of bytes in the header.

Waveform Type An integer (4 byte signed) which is the type of waveform that is stored in the file. The follow shows what each value means.

0 = Unknown

1 = Normal

2 = Peak Detect

3 = Average

4 = Horizontal Histogram

5 = Vertical Histogram

6 = Logic

Number of Waveform Buffers An integer (4 byte signed) which is the number of waveform buffers required to read the data. This value is one except for peak detect data and digital data.

Count	An integer (4 byte signed) which is the number of hits at each time bucket in the waveform record when the waveform was created using an acquisition mode like averaging. For example, when averaging, a count of four would mean every waveform data point in the waveform record has been averaged at least four times. The default value is 0.
X Display Range	A float (4 bytes) which is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.
X Display Origin	A double (8 bytes) which is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.
X Increment	A double (8 bytes) which is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.
X Origin	A double (8 bytes) which is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

X Units	An integer (4 byte signed) which is the number of X units columns (n) depends on the number of sources being stored. The X units is the unit of measure for each time value of the acquired data. X unit definitions are: 0 = Unkown 1 = Volt 2 = Second 3 = Constant 4 = Amp 5 = Decibel
Y Units	An integer (4 byte signed) which is the number of Y units columns (n) depends on the number of sources being stored. The Y units is the unit of measure of each voltage value of the acquired waveform. Y units definitions are: 0 = Unkown 1 = Volt 2 = Second 3 = Constant 4 = Amp 5 = Decibel
Date	A 16 character array which is the date when the waveform was acquired. The default value is 27 DEC 1996.
Time	A 16 character array which is the time when the waveform was acquired. The default value is 01:00:00:00.
Frame	A 24 character array which is the model number and serial number of the scope in the format of MODEL#:SERIAL#.
Waveform Label	A 16 character array which is the waveform label.
Time Tags	A double (8 bytes) which is the time tag value of the segment being saved.

Segment Index An unsigned integer (4 byte signed) which is the segment index of the data that follows the waveform data header.

Waveform Data Header

The waveform data header consists of information about the waveform data points that are stored immediately after the waveform data header.

Waveform Data Header Size An integer (4 byte signed) which is the size of the waveform data header.

Buffer Type A short (2 byte signed) which is the type of waveform data that is stored in the file. The following shows what each value means.

0 = Unknown data

1 = Normal 32 bit float data

2 = Maximum float data

3 = Minimum float data

4 = Time float data

5 = Counts 32 bit float data

6 = Digital unsigned 8 bit char data

Bytes Per Point A short (2 byte signed) which is the number of bytes per data point.

Buffer Size An integer (4 byte signed) which is the size of the buffer required to hold the data bytes.

Example Program for Reading Binary Data

The following is a programming example of reading a Binary Data (.bin) file and converting it to a CSV (.csv) file without a file header.

```
/* bintoascii.c */

/* Reads the binary file format.
   This program demonstrates how to import the Infiniium
   oscilloscope binary file format and how to export it to an
   ascii comma seperated file format.
*/
#pragma pack(4)

#include <stdio.h>              /* location of: printf() */
#include <stdlib.h>      /* location of: atof(), atoi() */
#include <string.h>     /* location of: strlen() */
#include "sicl.h"

/* Defines */
#define MAX_LENGTH 10000000
#define INTERFACE "lan[130.29.70.247]:inst0" /* Change the IP address
                                              * to the one found in
                                              * the Remote Setup
                                              * dialog box.
                                              */
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Type definitions */
typedef unsigned _int64 UINT64; /* This defines a 64-bit unsigned
                                 * integer for Microsoft platforms.
                                 */
```

```
/* Structure and Union definitions */
union DATATYPE
{
   char buffer[MAX_LENGTH];  /* Buffer for reading word format data */
   char byte[MAX_LENGTH];
   unsigned short word[MAX_LENGTH/2];
   UINT64 longlong[MAX_LENGTH/4];
};


typedef struct
{
   char Cookie[2];
   char Version[2];
   int  FileSize;
   int  NumberOfWaveforms;
} FileHeader;

const char COOKIE[2] = {'A', 'G'};
const char VERSION[2] = {'1', '0'};

#define DATE_TIME_STRING_LENGTH 16
#define FRAME_STRING_LENGTH  24
#define SIGNAL_STRING_LENGTH 16
```

```
typedef struct
{
   int    HeaderSize;
   int    WaveformType;
   int    NWaveformBuffers;
   int    Points;
   int    Count;
   float  XDisplayRange;
   double XDisplayOrigin;
   double XIncrement;
   double XOrigin;
   int    XUnits;
   int    YUnits;
   char   Date[DATE_TIME_STRING_LENGTH];
   char   Time[DATE_TIME_STRING_LENGTH];
   char   Frame[FRAME_STRING_LENGTH];
   char   WaveformLabel[SIGNAL_STRING_LENGTH];
   double TimeTag;
   unsigned int SegmentIndex;
} WaveformHeader;

typedef struct
{
   int   HeaderSize;
   short BufferType;
   short BytesPerPoint;
   int   BufferSize;
} WaveformDataHeader;

typedef enum
{
   PB_UNKNOWN,
   PB_NORMAL,
   PB_PEAK_DETECT,
   PB_AVERAGE,
   PB_HORZ_HISTOGRAM,
   PB_VERT_HISTOGRAM,
   PB_LOGIC
} WaveformType;
```

```
typedef enum
{
   PB_DATA_UNKNOWN,
   PB_DATA_NORMAL,
   PB_DATA_MAX,
   PB_DATA_MIN,
   PB_DATA_TIME,
   PB_DATA_COUNTS,
   PB_DATA_LOGIC
} DataType;

/* Prototypes */
void GetTimeConversionFactors( WaveformHeader waveformHeader,
                               double *xInc, double *xOrg );
void OutputNormalWaveform( WaveformHeader waveformHeader );
void OutputPeakDetectWaveform( WaveformHeader waveformHeader );
void OutputHistogramWaveform( WaveformHeader waveformHeader );
void OutputData( FILE *PeakFile,
                 WaveformDataHeader waveformDataHeader );

/* Globals */
double xOrg=0L, xInc=0L;     /* Values necessary to create time data */
union DATATYPE WaveFormData; /* Used to input and output data */
FILE *InputFile = NULL;
FILE *OutputFile;
errno_t err;
char *buffer;
float Volts[MAX_LENGTH];
float MaxVolts[MAX_LENGTH];
float MinVolts[MAX_LENGTH];
UINT64 HistogramData[MAX_LENGTH];
```

```
int main( int argc, char **argv )
{
   FileHeader fileHeader;
   WaveformHeader waveformHeader;

   if( argc > 1 )
   {
      InputFile = fopen( argv[1], "rb" );

      if( InputFile )
      {
         OutputFile = fopen( argv[2], "wb" );

         if( OutputFile )
         {
            /* Read the File Header */
            fread( &fileHeader, 1, sizeof( FileHeader ), InputFile );

            /* Make sure that this is an Agilent Binary File */
            if( (fileHeader.Cookie[0] == COOKIE[0]) &&
                (fileHeader.Cookie[1] == COOKIE[1]) )
            {
               fread( &waveformHeader, 1,
                      sizeof( WaveformHeader ), InputFile );

               switch( waveformHeader.WaveformType )
               {
                  case PB_NORMAL:
                  case PB_AVERAGE:
                     OutputNormalWaveform( waveformHeader );
                     break;
                  case PB_PEAK_DETECT:
                     OutputPeakDetectWaveform( waveformHeader );
                     break;
                  case PB_HORZ_HISTOGRAM:
                  case PB_VERT_HISTOGRAM:
                     OutputHistogramWaveform( waveformHeader );
                     break;
                  default:
```

```
                case PB_UNKNOWN:
                   printf( "Unknown waveform type: %d\n" );
                   break;
             }
          }
       }
       else
       {
          printf( "Unable to open output file %s\n", OutputFile);
       }
    }
    else
    {
       printf( "Unable to open input file %s\n", argv[1] );
    }

    fclose( InputFile );
    fclose( OutputFile );
 }
 else
 {
    printf( "Useage: bintoascii inputfile outputfile\n" );
 }

}
```

```
/*****************************************************************
 * Function name:   GetTimeConversionFactors
 *  Parameters:     double xInc which is the time between consecutive
 *                     sample points.
 *                  double xOrg which is the time value of the first
 *                     data point.
 *   Return value: none
 *   Description: This routine transfers the waveform conversion
 *                factors for the time values.
*****************************************************************/
void GetTimeConversionFactors( WaveformHeader waveformHeader,
                               double *xInc, double *xOrg )
{

   /* Read values which are used to create time values */

   *xInc = waveformHeader.XIncrement;
   *xOrg = waveformHeader.XOrigin;

}
```

```
/*****************************************************************
 * Function name:  OutputNormalWaveform
 *  Parameters:    WaveformHeader *waveformHeader which is a structure
 *                 that contains the waveform header information.
 *   Return value: none
 *   Description: This routine stores the time and voltage information
 *                about the waveform as time and voltage separated by
 *                commas to a file.
*****************************************************************/
void OutputNormalWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;
   double Time;

   BytesRead = fread( &waveformDataHeader, 1,
                      sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );
   while( !done )
   {
      BytesRead = fread( (char *) Volts, 1, MAX_LENGTH, InputFile );
      for( i = 0; i < (BytesRead/waveformDataHeader.BytesPerPoint); i++)
      {
         Time = (j * xInc) + xOrg; /* calculate time */
         j = j + 1;
         fprintf( OutputFile, "%e,%f\n", Time, Volts[i] );
      }
      if( BytesRead < MAX_LENGTH )
      {
         done = TRUE;
      }
   }
}
```

```
/*********************************************************************
 * Function name: OutputHistogramWaveform
 *  Parameters:   WaveformHeader *waveformHeader which is a structure
 *                   that contains the waveform header information.
 *  Return value: none
 *  Description:  This routine stores the time and hits information
 *                as time and hits separated by commas to a file.
*********************************************************************/
void OutputHistogramWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;

   fread( &waveformDataHeader, 1,
          sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );
   while( !done )
   {
      BytesRead = fread( (char *) HistogramData, 1, MAX_LENGTH,
                         InputFile );

      for( i = 0; i < (BytesRead/waveformDataHeader.BytesPerPoint); i++)
       {
          fprintf( OutputFile, "%d,%u64l\n", j, HistogramData[i] );
          j = j + 1;
       }
      if( BytesRead < MAX_LENGTH )
      {
         done = TRUE;
      }
  }
}
```

```
/*********************************************************************
 * Function name: OutputData
 *  Parameters:   FILE *PeakFile which is the pointer to the file
 *                   to be written.
 *                WaveformDataHeader waveformDataHeader
 *                   which is a structure that contains the waveform
 *                   header information.
 *  Return value: none
 *  Description: This routine stores the time, minimum voltage, and
 *               maximum voltage for the peak detect waveform as comma
 *               separated values to a file.
*********************************************************************/
void OutputData( FILE *PeakFile, WaveformDataHeader waveformDataHeader )
{
   int done = FALSE;
   size_t BytesRead = 0L;
   int NumberToRead;

   NumberToRead = waveformDataHeader.BufferSize;

   while( !done )
   {
      BytesRead = fread( (char *) Volts, 1, NumberToRead, InputFile ) +
                  BytesRead;

      fwrite( Volts, 1, BytesRead, PeakFile );

      if( BytesRead <= NumberToRead )
      {
         done = TRUE;
      }
   }
}
```

```
/*********************************************************************
 * Function name: OutputPeakDetectWaveform
 *  Parameters:   WaveformHeader waveformHeader which is a
 *                structure that contains the waveform header
 *                information.
 *  Return value: none
 *  Description: This routine stores the time, minimum voltage, and
 *               maximum voltage for the peak detect waveform as comma
 *               separated values to a file.
*********************************************************************/
void OutputPeakDetectWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;
   double Time;
   FILE *MaxFile;
   FILE *MinFile;

   fread( &waveformDataHeader, 1,
          sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );

   MaxFile = fopen( "maxdata.bin", "wb" );
   MinFile = fopen( "mindata.bin", "wb" );

   if( MaxFile && MinFile )
   {
      if( waveformDataHeader.BufferType == PB_DATA_MAX )
      {
         OutputData( MaxFile, waveformDataHeader );
         OutputData( MinFile, waveformDataHeader );
      }
      else
      {
         OutputData( MinFile, waveformDataHeader );
         OutputData( MaxFile, waveformDataHeader );
```

```
    }

    fclose( MaxFile );
    fclose( MinFile );

    MaxFile = fopen( "maxdata.bin", "rb" );
    MinFile = fopen( "mindata.bin", "rb" );

    while( !done )
    {
       BytesRead = fread( (char *) MaxVolts, 1, MAX_LENGTH, MaxFile );
       fread( (char *) MinVolts, 1, MAX_LENGTH, MinFile );

       for( i = 0; i < BytesRead/4; i++)
       {
          Time = (j * xInc) + xOrg; /* calculate time */
          j = j + 1;
          fprintf( OutputFile, "%9.5e,%f,%f\n", Time, MinVolts[i],
                   MaxVolts[i] );
       }

       if( BytesRead < MAX_LENGTH )
       {
          done = TRUE;
       }
    }

    fclose( MaxFile );
    fclose( MinFile );
}
```

SEGMented

Command

```
:DISK:SEGMented {ALL | CURRent}
```

The :DISK:SEGMented command sets whether all segments or just the current segment are saved to a file when the :DISK:SAVe:WAVeform command is issued and the source is a channel but not a waveform memory or function. Before segments can be saved, the :ACQuire:MODE must be set to the SEGMented mode and segments must be acquired.

Example

This example sets the disk segmented memory store method to CURRent.

```
10 OUTPUT 707;":DISK:SEGMENTED CURRENT"
20 END
```

Query

```
:DISK:SEGMented?
```

The :DISK:SEGMented? query returns disk segmented memory store method value.

Returned Format

```
[:DISK:SEGMented] {ALL | CURRent}<NL>
```

Example

This example places the disk store method in the string variable Method$, then prints the contents of the variable to the computer's screen.

```
10 DIM Method$[200]
20 OUTPUT 707;":DISK:SEGMENTED?"
30 ENTER 707; Method$
40 PRINT Method$
50 END
```

14

Display Commands

Display Commands

The DISPlay subsystem controls the display of data, text, and graticules, and the use of color.

These DISPlay commands and queries are implemented in the Infiniium Oscilloscopes:

- CGRade
- CGRade:LEVels?
- COLumn
- CONNect
- DATA?
- DCOLor (Default COLor)
- GRATicule
- LABel
- LINE
- PERSistence
- ROW
- SCOLor (Set COLor)
- STRing
- TAB
- TEXT

CGRade

Command

```
:DISPlay:CGRade {{ON | 1} | {OFF | 0}}
```

The :DISPlay:CGRade command sets the color grade persistence on or off.

When in the color grade persistence mode, all waveforms are mapped into a database and shown with different colors representing varying number of hits in a pixel. "Connected dots" display mode (:DISPlay:CONNect) is disabled when the color grade persistence is on.

The oscilloscope has three features that use a specific database. This database uses a different memory area than the waveform record for each channel. The three features that use the database are histograms, mask testing, and color grade persistence. When any one of these three features is turned on, the oscilloscope starts building the database. The database is the size of the graticule area and varies in size. Behind each pixel is a 53-bit counter. Each counter is incremented each time a pixel is hit by data from a channel or function. The maximum count (saturation) for each counter is 9,007,199,254,740,991. You can check for counter saturation by using the DISPlay:CGRade:LEVels? query. The color grade persistence uses colors to represent the number of hits on various areas of the display. The default color-grade state is off.

Example

This example sets the color grade persistence on.

```
10 OUTPUT 707;":DISPLAY:CGRADE ON"
20 END
```

Query `:DISPlay:CGRade?`

The DISPlay:CGRade query returns the current color-grade state.

Returned Format `[:DISPlay:CGRade] {1 | 0}<NL>`

Example This example returns the current color grade state.

```
10 OUTPUT 707;":DISPLAY:CGRADE?"
20 ENTER 707;Cgrade$
30 PRINT Cgrade$
40 END
```

CGRade:LEVels?

Query `:DISPlay:CGRade:LEVels?`

The :DISPlay:CGRade:LEVels? query returns the range of hits represented by each color. Fourteen values are returned, representing the minimum and maximum count for each of seven colors. The values are returned in the following order:

- White minimum value
- White maximum value
- Yellow minimum value
- Yellow maximum value
- Orange minimum value
- Orange maximum value
- Red minimum value
- Red maximum value
- Pink minimum value
- Pink maximum value
- Blue minimum value
- Blue maximum value
- Green minimum value
- Green maximum value

Returned Format `[DISPlay:CGRade:LEVels] <color format><NL>`

`<color format>` <intensity color min/max> is an integer value from 0 to 9,007,199,254,740,991

Example

This example gets the range of hits represented by each color and prints it on the computer screen:

```
10 DIM Setting$[50]          !Dimension variable
20 OUTPUT 707;":DISPLAY:CGRADE:LEVELS?"
30 ENTER 707;Cgrade$
40 PRINT Cgrade$
50 END
```

Colors start at green minimum, maximum, then blue, pink, red, orange, yellow, white. The format is a string where commas separate minimum and maximum values. The largest number in the string can be 9,007,199,254,740,991

An example of a possible returned string is as follows:

1,414,415,829,830,1658,1659,3316,3317,6633,6634,13267,13268,26535

COLumn

Command

```
:DISPlay:COLumn <column_number>
```

The :DISPlay:COLumn command specifies the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands.

`<column_number>` An integer representing the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The range of values is 0 to 90.

Example

This example sets the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands to column 10.

```
10  OUTPUT 707;":DISPLAY:COLUMN 10"
20  END
```

Query

```
:DISPlay:COLumn?
```

The :DISPlay:COLumn? query returns the column where the next :DISPlay:LINE or :DISPlay:STRing starts.

Returned Format

```
[:DISPlay:COLumn] <value><NL>
```

Example

This example returns the current column setting to the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:COLUMN?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

CONNect

Command

```
:DISPlay:CONNect {{ON | 1} | {OFF | 0}}
```

When enabled, :DISPlay:CONNect draws a line between consecutive waveform data points. This is also known as linear interpolation.

:DISPlay:CONNect is forced to OFF when color grade (:DISPlay:CGRade) persistence is on.

Example

This example turns on the connect-the-dots feature.

```
10  OUTPUT 707;":DISPLAY:CONNECT ON"
20  END
```

Query

```
:DISPlay:CONNect?
```

The :DISPlay:CONNect? query returns the status of the connect-the-dots feature.

Returned Format

```
[:DISPlay:CONNect] {1 | 0}<NL>
```

DATA?

Query

```
:DISPlay:DATA?
[<type>[,<screen_mode>[,<compression>
[,<inversion>]]]]
```

The :DISPlay:DATA? query returns information about the captured data. If no options to the query are specified, the default selections are BMP file type, SCReen mode, compression turned ON, and inversion set to NORMal.

`<type>` The bitmap type: BMP | JPG | GIF | TIF | PNG.

`<screen_mode>` The display setting: SCReen | GRATicule. Selecting GRATicule displays a 10-by-8 (unit) display graticule on the screen. See also :DISPlay:GRATicule.

`<compression>` The file compression feature: ON | OFF.

`<inversion>` The inversion of the displayed file: NORMal | INVert.

Returned Format `[:DISPlay:DATA] <binary_block_data><NL>`

`<binary_block_data>` Data in the IEEE 488.2 definite block format.

GRATicule

Commands

```
:DISPlay:GRATicule {GRID | FRAMe}
:DISPlay:GRATicule:INTensity <intensity_value>
:DISPlay:GRATicule:NUMBer {1 | 2 | 4}
:DISPlay:GRATicule:SETGrat <DispGratChan>, <number>
```

The :DISPlay:GRATicule command selects the type of graticule that is displayed. Infiniium oscilloscopes have a 10-by-8 (unit) display graticule grid GRID), a grid line is place on each vertical and horizontal division. When it is off (FRAMe), a frame with tic marks surrounds the graticule edges.

You can dim the grid's intensity or turn the grid off to better view waveforms that might be obscured by the graticule lines using the :DISPlay:GRATicule:INTensity command. Otherwise, you can use the grid to estimate waveform measurements such as amplitude and period.

When printing, the grid intensity control does not affect the hard copy. To remove the grid from a printed hard copy, you must turn off the grid before printing.

The :SETGrat command assigns the corresponding waveform to a specific grid on the display. For example, :DISP:GRAT:SETG HIST, 2 would assign the histogram to grid 2.

`<intensity_value>` A integer from 0 to 100, indicating the percentage of grid intensity.

`<DispGratChan>` CHN<N>, DIFF1, DIFF3, COMM2, COMM41, MEM<N> where N is between 1 and 4, FN<N> where N is between 1 and 4 (function), HIST

You can divide the waveform viewing area from one area into two or four separate viewing areas using the :DISPlay:GRATicule:NUMBer command. This allows you to separate waveforms without having to adjust the vertical position controls.

Example

This example sets up the oscilloscope's display background with a frame that is separated into major and minor divisions.

```
10  OUTPUT 707;":DISPLAY:GRATICULE FRAME"
20  END
```

Queries

```
:DISPlay:GRATicule?
:DISPlay:GRATicule:INTensity?
:DISPlay:GRATicule:NUMBer?
```

The :DISPlay:GRATicule?, :DISPlay:GRATicule:INTensity?, and DISPlay:GRATicule:NUMBer? queries return the type of graticule currently displayed, the intensity, or the number of viewing areas, depending on the query you request.

Returned Format

```
[:DISPlay:GRATicule] {GRID | FRAMe}<NL>
[:DISPlay:GRATicule:INTensity] <value><NL>
[:DISPlay:GRATicule:NUMBer {1 | 2 | 4}<NL>
```

Example

This example places the current display graticule setting in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:GRATICULE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

LABel

Command

```
:DISPlay:LABel {{ON | 1} | {OFF | 0}}
```

The :DISPlay:LABel command turns on or off the display of analog channel labels. Label names can be up to 6 characters long. The label name is assigned by using the CHANnel<n>:LABel command:

Example

This example turns on the display of all labels.

```
10  OUTPUT 707;":DISPLAY:LABEL ON"
20  END
```

Query

```
:DISPlay:LABel?
```

The :DISPlay:LABel? query returns the current state of the labels.

Returned Format

```
[:DISPlay:LABel] {1 | 0}<NL>
```

Example

This example places the current label state into the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:LABEL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

LINE

Command

```
:DISPlay:LINE "<string_argument>"
```

The :DISPlay:LINE command writes a quoted string to the screen, starting at the location specified by the :DISPlay:ROW and :DISPlay:COLumn commands. When using the C programming language, quotation marks as shown in the example delimit a string.

`<string_argument>` Any series of ASCII characters enclosed in quotation marks.

Example

This example writes the message “Infiniium Test” to the screen, starting at the current row and column location.

```
10  OUTPUT 707;":DISPLAY:LINE ""Infiniium Test"""
20  END
```

This example writes the message "Infiniium Test" to the screen using C. Quotation marks are included because the string is delimited.

```
printf("\"Infiniium Test\"");
```

You may write text up to column 94. If the characters in the string do not fill the line, the rest of the line is blanked. If the string is longer than the space available on the current line, the excess characters are discarded.

In any case, the ROW is incremented and the COLumn remains the same. The next :DISPlay:LINE command will write on the next line of the display. After writing the last line in the display area, the ROW is reset to 0.

PERSistence

Command

```
:DISPlay:PERSistence {MINimum | INFinite}
```

The :DISPlay:PERSistence command sets the display persistence. It works in both real time and equivalent time modes. The parameter for this command can be either MINimum (zero persistence) or INFinite

Example

This example sets the persistence to infinite.

```
10  OUTPUT 707;":DISPLAY:PERSISTENCE INFINITE"
20  END
```

Query

```
:DISPlay:PERSistence?
```

The :DISPlay:PERSistence? query returns the current persistence value.

Returned Format

```
[:DISPlay:PERSistence] {MINimum | INFinite}<NL>
```

Example

This example places the current persistence setting in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:PERSISTENCE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

ROW

Command

```
:DISPlay:ROW <row_number>
```

The :DISPlay:ROW command specifies the starting row on the screen for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The row number remains constant until another :DISPlay:ROW command is received, or the row is incremented by the :DISPlay:LINE command.

`<row_number>` An integer representing the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The range of values is 9 to 23.

Example

This example sets the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands to 10.

```
10  OUTPUT 707;":DISPLAY:ROW 10"
20  END
```

Query

```
:DISPlay:ROW?
```

The :DISPlay:ROW? query returns the current value of the row.

Returned Format

```
[:DISPlay:ROW] <row_number><NL>
```

Example

This example places the current value for row in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:ROW?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

SCOLor

Command

```
:DISPlay:SCOLor <color_name>, <hue>, <saturation>,
<luminosity>
```

The :DISPlay:SCOLor command sets the color of the specified display element. The display elements are described in Table 14-1.

`<color_name>` `{CGLevel1 | CGLevel2 | CGLevel3 | CGLevel4 | CGLevel5 | CGLevel6 | CGLevel7 | CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4 | DBACkgrnd | GRID | MARKers | MEASurements | MICons | MTPolygons | STEXt | WBACkgrnd | TINPuts | WOVerlap | TSCale | WMEMories | WINText | WINBackgrnd}`

Table 14-1 Color Names

Color Name	Definition
CGLevel1	**Color Grade Level 1 waveform display element.**
CGLevel2	**Color Grade Level 2 waveform display element.**
CGLevel3	**Color Grade Level 3 waveform display element.**
CGLevel4	**Color Grade Level 4 waveform display element.**
CGLevel5	**Color Grade Level 5 waveform display element.**
CGLevel6	**Color Grade Level 6 waveform display element.**
CGLevel7	**Color Grade Level 7 waveform display element.**
CHANnel1	**Channel 1 waveform display element.**
CHANnel2	**Channel 2 waveform display element.**
CHANnel3	**Channel 3 waveform display element.**
CHANnel4	**Channel 4 waveform display element.**
DBACkgrnd	**Display element for the border around the outside of the waveform viewing area.**
GRID	**Display element for the grid inside the waveform viewing area.**
MARKers	**Display element for the markers.**
MEASurements	**Display element for the measurements text.**
MICons	**Display element for measurement icons to the left of the waveform viewing area.**

Color Name	Definition
STEXt	**Display element for status messages displayed in the upper left corner of the display underneath the menu bar. Changing this changes the memory bar's color.**
WBACkgrnd	**Display element for the waveform viewing area's background.**
TINPuts	**Display element for line and aux trig colors.**
WOVerlap	**Display element for waveforms when they overlap each other.**
TSCale	**Display element for horizontal scale and offset control text.**
WMEMories	**Display element for waveform memories.**
WINText	**Display element used in dialog box controls and pull-down menus.**
WINBackgrnd	**Display element for the background color used in dialog boxes and buttons.**

`<hue>` An integer from 0 to 100. The hue control sets the color of the chosen display element. As hue is increased from 0%, the color changes from red, to yellow, to green, to blue, to purple, then back to red again at 100% hue. For color examples, see the sample color settings table in the Infiniium Oscilloscope online help file. Pure red is 100%, pure blue is 67%, and pure green is 33%.

`<saturation>` An integer from 0 to 100. The saturation control sets the color purity of the chosen display element. The saturation of a color is the purity of a color, or the absence of white. A 100% saturated color has no white component. A 0% saturated color is pure white.

`<luminosity>` An integer from 0 to 100. The luminosity control sets the color brightness of the chosen display element. A 100% luminosity is the maximum color brightness. A 0% luminosity is pure black.

Example

This example sets the hue to 50, the saturation to 70, and the luminosity to 90 for the markers.

```
10  OUTPUT 707;":DISPLAY:SCOLOR MARKERS,50,70,90"
20  END
```

Query

```
:DISPlay:SCOLor? <color_name>
```

The :DISPlay:SCOLor? query returns the hue, saturation, and luminosity for the specified color.

Returned Format

```
[:DISPlay:SCOLor] <color_name>, <hue>, <saturation>,
<luminosity><NL>
```

Example

This example places the current settings for the graticule color in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:SCOLOR? GRATICULE"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

STRing

Command

```
:DISPlay:STRing "<string_argument>"
```

The :DISPlay:STRing command writes text to the oscilloscope screen. The text is written starting at the current row and column settings. If the column limit is reached, the excess text is discarded. The :DISPlay:STRing command does not increment the row value, but :DISPlay:LINE does.

`<string_argument>` Any series of ASCII characters enclosed in quotation marks.

Example

This example writes the message “Example 1” to the oscilloscope's display starting at the current row and column settings.

```
10  OUTPUT 707;":DISPLAY:STRING ""Example 1"""
20  END
```

TAB

Command

```
:DISPlay:TAB <tab>
```

The :DISPlay:TAB command displays the corresponding tab indicated by the <tab> parameter.

`<tab>` MEASurement | MARKer | LIMittest | JITTr | HISTogram | MASKtest | EYE | COLorgrade | NAVigtation | STATus | SCALe

Example

This example sets the Status tab as the displayed one.

```
10 OUTPUT 707;":DISPLAY:TAB STATus"
20 END
```

TEXT

Command

```
:DISPlay:TEXT BLANk
```

The :DISPlay:TEXT command blanks the user text area of the screen.

Example

This example blanks the user text area of the oscilloscope's screen.

```
10  OUTPUT 707;":DISPLAY:TEXT BLANK"
20  END
```

15

Function Commands

Function Commands

The FUNCtion subsystem defines functions 1 - 4. The operands of these functions can be any of the installed channels in the oscilloscope, waveform memories 1 - 4, functions 1 - 4, or a constant. These FUNCtion commands and queries are implemented in the Infiniium Oscilloscopes:

- FUNCtion<N>?
- ABSolute
- ADD
- AVERage
- COMMonmode
- DIFF (Differentiate)
- DISPlay
- DIVide
- FFT:FREQuency
- FFT:RESolution?
- FFT:WINDow
- FFTMagnitude
- FFTPhase
- HIGHpass
- HORizontal
- HORizontal:POSition
- HORizontal:RANGe
- INTegrate
- INVert
- LOWPass
- MAGNify
- MAXimum
- MINimum
- MULTiply
- OFFSet
- RANGe
- SMOoth
- SQRT
- SQUare
- SUBTract
- VERSus
- VERTical
- VERTical:OFFset
- VERTical:RANGe

You can control the vertical scaling and offset functions remotely using the RANGe and OFFSet commands in this subsystem. You can obtain the horizontal scaling and position values of the functions using the :HORizontal:RANGe? and :HORizontal:POSition? queries in this subsystem.

If a channel is not on but is used as an operand, that channel will acquire waveform data.

If the operand waveforms have different memory depths, the function uses the shorter of the two.

If the two operands have the same time scales, the resulting function has the same time scale. If the operands have different time scales, the resulting function has no valid time scale. This is because operations are performed based on the displayed waveform data position, and the time relationship of the data records cannot be considered. When the time scale is not valid, delta time pulse parameter measurements have no meaning, and the unknown result indicator is displayed on the screen.

Constant operands take on the same time scale as the associated waveform operand.

FUNCtion<N>?

Query

```
:FUNCtion<N>?
```

The :FUNCtion<N>? query returns the currently defined source(s) for the function.

Returned Format

```
[:FUNCtion<N>:<operator>] {<operand>[,<operand>]}<NL>
```

<N> An integer, 1 - 4, representing the selected function.

<operator> Active math operation for the selected function: ADD, AVERage, COMMonmode, DIFF, DIVide, FFTMagnitude, FFTPhase, HIGHpass, INTegrate, INVert, LOWPass, MAGNify, MAXimum, MINimum, MULTiply, SMOoth, SUBTract, or VERSus.

<operand> Any allowable source for the selected FUNCtion, including channels, waveform memories 1-4, and functions 1-4. If the function is applied to a constant, the source returns the constant.

The channel number is an integer, 1 - 4.

Example

This example returns the currently defined source for function 1.

```
10  OUTPUT 707;":FUNCTION1?"
20  END
```

If the headers are off (see :SYSTem:HEADer), the query returns only the operands, not the operator.

```
10  :SYST:HEAD ON
20  :FUNC1:ADD CHAN1,CHAN2
30  :FUNC1?  !returns :FUNC1:ADD CHAN1,CHAN2
40  :SYST:HEAD OFF
50  :FUNC1?  !returns CHAN1,CHAN2
```

ABSolute

Command `:FUNCtion<N>:ABSolute <operand>`

The :FUNCtion<N>:ABSolute command takes the absolute value an operand.

`<operand>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is an integer, 1 - 4.
FUNCtion<N> and WMEMory<N> are:
An integer, 1 - 4, representing the selected function or waveform memory.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

Example This example turns on the absolute value command using channel 3.

```
10 OUTPUT 707;"MEASURE:ABSOLUTE CHANNEL3"
20 END
```

ADD

Command

```
:FUNCtion<N>:ADD <operand>,<operand>
```

The :FUNCtion<N>:ADD command defines a function that takes the algebraic sum of the two operands.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

Example

This example sets up function 1 to add channel 1 to channel 2.

```
10  OUTPUT 707;":FUNCTION1:ADD CHANNEL1,CHANNEL2"
20  END
```

AVERage

Command

```
:FUNCtion<N>:AVERage <operand>[,<averages>]
```

The :FUNCtion<N>:AVERage command defines a function that averages the operand based on the number of specified averages.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

`<averages>` An integer, 2 to 65534 specifying the number of waveforms to be averaged

Example

This example sets up function 1 to average channel 1 using 16 averages.

```
10  OUTPUT 707;":FUNCTION1:AVERAGE CHANNEL1,16"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

COMMonmode

Command

```
:FUNCtion<N>:COMMonmode <operand>,<operand>
```

The :FUNCtion<N>:COMMonmode command defines a function that adds the voltage values of the two operands and divides by 2, point by point.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

Example

This example sets up function 1 to view the common mode voltage value of channel 1 and channel 2.

```
10  OUTPUT 707;":FUNCTION1:COMMONMODE CHANNEL1,CHANNEL2"
20  END
```

DIFF (Differentiate)

Command

```
:FUNCtion<N>:DIFF
<operand>[,<low_pass_phase_align>}
```

The :FUNCtion<N>:DIFF command defines a function that computes the discrete derivative of the operand.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

`<low_pass_phase_align>` {{ON | 1} | {OFF | 0}

This parameter turns on or off the low pass and phase align filter.

Example

This example sets up function 2 to take the discrete derivative of the waveform on channel 2.

```
10  OUTPUT 707;":FUNCTION2:DIFF CHANNEL2"
20  END
```

DISPlay

Command

```
:FUNCtion<N>:DISPlay {{ON|1} | {OFF|0}}
```

The :FUNCtion<N>:DISPlay command either displays the selected function or removes it from the display.

<N> An integer, 1 - 4, representing the selected function.

Example

This example turns function 1 on.

```
10  OUTPUT 707;":FUNCTION1:DISPLAY ON"
20  END
```

Query

```
:FUNCtion<N>:DISPlay?
```

The :FUNCtion<N>:DISPlay? query returns the displayed status of the specified function.

Returned Format

```
[:FUNCtion<N>:DISPlay] {1|0}<NL>
```

Example

This example places the current state of function 1 in the variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":FUNCTION1:DISPLAY?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

DIVide

Command

```
:FUNCtion<N>:DIVide <operand>,<operand>
```

The :FUNCtion<N>:DIVide command defines a function that divides the first operand by the second operand.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is: A real number from -1E6 to 1E12.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

Example

This example sets up function 2 to divide the waveform on channel 1 by the waveform in waveform memory 4.

```
10  OUTPUT 707;":FUNCTION2:DIVIDE CHANNEL1,WMEMORY4"
20  END
```

FFT:FREQuency

Command `:FUNCtion<N>:FFT:FREQuency <center_frequency_value>`

The :FUNCtion<N>:FFT:FREQuency command sets the center frequency for the FFT when :FUNCtion<N>:FFTMagnitude is defined for the selected function.

`<N>` An integer, 1 - 4, representing the selected function.

`<center_frequency_value>` A real number for the value in Hertz, from -1E12 to 1E12.

Query `:FUNCtion<N>:FFT:FREQuency?`

The :FUNCtion<N>:FFT:FREQuency? query returns the center frequency value.

Returned Format `[FUNCtion<N>:FFT:FREQuency] <center_frequency_value><NL>`

FFT:REFerence

Command

```
:FUNCtion<N>:FFT:REFerence {DISPlay | TRIGger}
```

The :FUNCtion<N>:FFT:REFerence command sets the reference point for calculating the FFT phase function.

<N> An integer, 1 - 4, representing the selected function.

Example

This example sets the reference point to DISPlay.

```
10 OUTPUT 707;":FUNCTION<N>:FFT:REFERENCE DISPLAY
20 END
```

Query

```
:FUNCtion<N>:FFT:REFerence?
```

The :FUNCtion<N>:FFT:REFerence? query returns the currently selected reference point for the FFT phase function.

Returned Format

```
[:FUNCtion<N>:FFT:REFerence] {DISPlay | TRIGger}<NL>
```

Example

This example places the current state of the function 1 FFT reference point in the string variable, REF?, then prints the contents of the variable to the computer's screen.

```
10 DIM REF$[50]
20 OUTPUT 707;":FUNCTION1:FFT:REFERENCE?"
30 ENTER 707;REF$
40 PRINT REF$
50 END
```

FFT:RESolution?

Query `:FUNCtion<N>:FFT:RESolution?`

The :FUNCtion<N>:FFT:RESolution? query returns the current resolution of the FFT function.

Returned Format `[FUNCtion<N>:FFT:RESolution] <resolution_value><NL>`

`<N>` An integer from 1 to 4 representing the selected function.

`<resolution_value>` Resolution frequency.

The FFT resolution is determined by the sample rate and memory depth settings. The FFT resolution is calculated using the following equation:

FFT Resolution = Sample Rate / Effective Memory Depth

The effective memory depth is the highest power of 2 less than or equal to the number of sample points across the display. The memory bar in the status area at the top of the display indicates how much of the actual memory depth is across the display.

FFT:WINDow

Command

```
:FUNCtion<N>:FFT:WINDow {RECTangular | HANNing | FLATtop}
```

The :FUNCtion<N>:FFT:WINDow command sets the window type for the FFT function.

The FFT function assumes that the time record repeats. Unless there is an integral number of cycles of the sampled waveform in the record, a discontinuity is created at the beginning of the record. This introduces additional frequency components into the spectrum about the actual peaks, which is referred to as spectral leakage. To minimize spectral leakage, windows that approach zero smoothly at the beginning and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input waveforms.

- The RECTangular window is essentially no window, and all points are multiplied by 1. This window is useful for transient waveforms and waveforms where there are an integral number of cycles in the time record.
- The HANNing window is useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements.
- The FLATtop window is best for making accurate amplitude measurements of frequency peaks.

<N> An integer, 1 - 4, representing the selected function. This command presently selects all functions, regardless of which integer (1-4) is passed.

Example

This example sets the window type for the FFT function to RECTangular.

```
10 OUTPUT 707;":FUNCTION<N>:FFT:WINDOW RECTANGULAR
20 END
```

Query

```
:FUNCtion<N>:FFT:WINDow?
```

The :FUNCtion<N>:FFT:WINDow? query returns the current selected window for the FFT function.

Returned Format

```
[:FUNCtion<N>:FFT:WINDow] {RECTangular | HANNing |
FLATtop}<NL>
```

Example

This example places the current state of the function 1 FFT window in the string variable, WND?, then prints the contents of the variable to the computer's screen.

```
10 DIM WND$[50]
20 OUTPUT 707;":FUNCTION1:FFT:WINDOW?"
30 ENTER 707;WND$
40 PRINT WND$
50 END
```

FFTMagnitude

Command

```
:FUNCtion<N>:FFTMagnitude <operand>
```

The :FUNCtion<N>:FFTMagnitude command computes the Fast Fourier Transform (FFT) of the specified channel, function, or memory. The FFT takes the digitized time record and transforms it to magnitude and phase components as a function of frequency.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example sets up function 1 to compute the FFT of waveform memory 3.

```
10 OUTPUT 707;":FUNCTION1:FFTMAGNITUDE WMEMORY3"
20 END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

FFTPhase

Command

```
:FUNCtion<N>:FFTPhase <source>
```

The :FUNCtion<N>:FFTPhase command computes the Fast Fourier Transform (FFT) of the specified channel, function, or waveform memory. The FFT takes the digitized time record and transforms it into magnitude and phase components as a function of frequency.

<N> An integer, 1 - 4, representing the selected function.

<source> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example sets up function 1 to compute the FFT of waveform memory 3.

```
10 OUTPUT 707;":FUNCTION1:FFTPHASE WMEMORY3"
20 END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

HIGHpass

Command `:FUNCtion<N>:HIGHpass <source>,<bandwidth>`

The :FUNCtion<N>:HIGHpass command applies a single-pole high pass filter to the source waveform. The bandwidth that you set is the 3 dB bandwidth of the filter.

`<N>` An integer, 1 - 4, representing the selected function.

`<source>` {CHANnel<n> | FUNCtion<n> | WMEMory<n>}

CHANnel<n> is:

An integer, 1 - 2, for two channel Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<bandwidth>` A real number in the range of 50 to 50E9.

Example This example sets up function 2 to compute a high pass filter with a bandwidth of 1 MHz.

```
10  OUTPUT 707;":FUNCTION2:HIGHPASS CHANNEL4,1E6"
20  END
```

HORizontal

Command

```
:FUNCtion<N>:HORizontal {AUTO | MANual}
```

The :FUNCtion<N>:HORizontal command sets the horizontal tracking to either AUTO or MANual.

The HORizontal command also includes the following commands and queries, which are described on the following pages:

- POSition
- RANGe

<N> An integer, 1 - 4, representing the selected function.

Query

```
:FUNCtion<N>:HORizontal?
```

The :FUNCtion<N>:HORizontal? query returns the current horizontal scaling mode of the specified function.

Returned Format

```
[:FUNCtion<N>:HORizontal] {AUTO | MANual}<NL>
```

Example

This example places the current state of the function 1 horizontal tracking in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":FUNCTION1:HORIZONTAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

HORizontal:POSition

Command

```
:FUNCtion<N>:HORizontal:POSition <position_value>
```

The :FUNCtion<N>:HORizontal:POSition command sets the time value at center screen for the selected function. If the oscilloscope is not already in manual mode when you execute this command, it puts the oscilloscope in manual mode.

When you select :FUNCtion<N>:FFTMagnitude, the horizontal position is equivalent to the center frequency. This also automatically selects manual mode.

`<N>` An integer, 1 - 4, representing the selected function.

`<position_value>` A real number for the position value in time, in seconds, from -10E15 to 10E15.

Query

```
:FUNCtion<N>:HORizontal:POSition?
```

The :FUNCtion<N>:HORizontal:POSition? query returns the current time value at center screen of the selected function.

Returned Format

```
[:FUNCtion<N>:HORizontal:POSition] <position><NL>
```

Example

This example places the current horizontal position setting for function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:HORIZONTAL:POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

HORizontal:RANGe

Command

```
:FUNCtion<N>:HORizontal:RANGe <range_value>
```

The :FUNCtion<N>:HORizontal:RANGe command sets the current time range for the specified function. This automatically selects manual mode.

`<N>` An integer, 1 - 4, representing the selected function.

`<range_value>` A real number for the width of screen in current X-axis units (usually seconds), from -100E-15 to 100E15.

Query

```
:FUNCtion<N>:HORizontal:RANGe?
```

The :FUNCtion<N>:HORizontal:RANGe? query returns the current time range setting of the specified function.

Returned Format

```
[:FUNCtion<N>:HORizontal:RANGe] <range><NL>
```

Example

This example places the current horizontal range setting of function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:HORIZONTAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

INTegrate

Command

```
:FUNCtion<N>:INTegrate <operand>
```

The :FUNCtion<N>:INTegrate command defines a function that computes the integral of the specified operand's waveform.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example sets up function 1 to compute the integral of waveform memory 3.

```
10  OUTPUT 707;":FUNCTION1:INTEGRATE WMEMORY3"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

INVert

Command

```
:FUNCtion<N>:INVert <operand>
```

The :FUNCtion<N>:INVert command defines a function that inverts the defined operand's waveform by multiplying by -1.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example sets up function 2 to invert the waveform on channel 1.

```
10  OUTPUT 707;":FUNCTION2:INVERT CHANNEL1"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

LOWPass

Command `:FUNCtion<N>:LOWPass <source>,<bandwidth>`

The :FUNCtion<N>:LOWPass command applies a 4th order Bessel-Thompson pass filter to the source waveform. The bandwidth that you set is the 3 dB bandwidth of the filter.

`<N>` An integer, 1 - 4, representing the selected function.

`<source>` {CHANnel<n> | FUNCtion<n> | WMEMory<n>}

CHANnel<n> is:

An integer, 1 - 2, for two channel Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<bandwidth>` A real number in the range of 50 to 50E9.

Example This example sets up function 2 to compute a low pass filter with a bandwidth of 1 MHz.

```
10  OUTPUT 707;":FUNCTION2:LOWPASS CHANNEL4,1E6"
20  END
```

MAGNify

Command

```
:FUNCtion<N>:MAGNify <operand>
```

The :FUNCtion<N>:MAGNify command defines a function that is a copy of the operand. The magnify function is a software magnify. No hardware settings are altered as a result of using this function. It is useful for scaling channels, another function, or memories with the RANGe and OFFSet commands in this subsystem.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example creates a function (function 1) that is a magnified version of channel 1.

```
10  OUTPUT 707;":FUNCTION1:MAGNIFY CHANNEL1"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

MAXimum

Command

```
:FUNCtion<N>:MAXimum <operand>
```

The :FUNCtion<N>:MAXmum command defines a function that computes the maximum of each time bucket for the defined operand's waveform.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example sets up function 2 to compute the maximum of each time bucket for channel 4.

```
10  OUTPUT 707;":FUNCTION2:MAXIMUM CHANNEL4"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

MINimum

Command

```
:FUNCtion<N>:MINimum <operand>
```

The :FUNCtion<N>:MINimum command defines a function that computes the minimum of each time bucket for the defined operand's waveform.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example sets up function 2 to compute the minimum of each time bucket for channel 4.

```
10  OUTPUT 707;":FUNCTION2:MINIMUM CHANNEL4"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

MULTiply

Command

```
:FUNCtion<N>:MULTiply <operand>,<operand>
```

The :FUNCtion<N>:MULTiply command defines a function that algebraically multiplies the first operand by the second operand.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example defines a function that multiplies channel 1 by waveform memory 1.

```
10  OUTPUT 707;":FUNCTION1:MULTIPLY CHANNEL1,WMEMORY1"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

OFFSet

Command

```
:FUNCtion<N>:OFFSet <offset_value>
```

The :FUNCtion<N>:OFFSet command sets the voltage represented at the center of the screen for the selected function. This automatically changes the mode from auto to manual.

`<N>` An integer, 1 - 4, representing the selected function.

`<offset_value>` A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited to being within the vertical range that can be represented by the function data.

Example

This example sets the offset voltage for function 1 to 2 mV.

```
10  OUTPUT 707;":FUNCTION1:OFFSET 2E-3"
20  END
```

Query

```
:FUNCtion<N>:OFFSet?
```

The :FUNCtion<N>:OFFSet? query returns the current offset value for the selected function.

Returned Format

```
[:FUNCtion<N>:OFFSet] <offset_value><NL>
```

Example

This example places the current setting for offset on function 2 in the numeric variable, Value, then prints the result to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

RANGe

Command `:FUNCtion<N>:RANGe <full_scale_range>`

The :FUNCtion<N>:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual.

`<N>` An integer, 1 - 4, representing the selected function.

`<full_scale_range>` A real number for the full-scale vertical range, from -100E15 to 100E15.

Example This example sets the full-scale range for function 1 to 400 mV.

```
10  OUTPUT 707;":FUNCTION1:RANGE 400E-3"
20  END
```

Query `:FUNCtion<N>:RANGe?`

The :FUNCtion<N>:RANGe? query returns the current full-scale range setting for the specified function.

Returned Format `[:FUNCtion<N>:RANGe] <full_scale_range><NL>`

Example This example places the current range setting for function 2 in the numeric variable "Value," then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

SMOoth

Command

```
:FUNCtion<N>:SMOoth <operand>[,<points>]
```

The :FUNCtion<N>:SMOoth command defines a function that assigns the smoothing operator to the operand with the number of specified smoothing points.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12

`<points>` An integer, odd numbers from 3 to 4001 specifying the number of smoothing points.

Example

This example sets up function 1 using assigning smoothing operator to channel 1 using 5 smoothing points.

```
10  OUTPUT 707;":FUNCTION1:SMOOTH CHANNEL1,5"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

SQRT

Command `:FUNCtion<N>:SQRT <operand>`

The :FUNCtion<N>:SQRT command takes the square root of the operand.

`<operand>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is an integer, 1 - 4.
FUNCtion<N> and WMEMory<N> are:
An integer, 1 - 4, representing the selected function or waveform memory.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

Example This example turns on the square root function using channel 3.

```
10 OUTPUT 707;"MEASURE:SQRT CHANNEL3"
20 END
```

SQUare

Command

```
:FUNCtion<N>:SQUare <operand>
```

The :FUNCtion<N>:SQUare command takes the square value of the operand.

`<operand>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

Example

This example turns on the square value command using channel 3.

```
10 OUTPUT 707;"MEASURE:SQUARE CHANNEL3"
20 END
```

SUBTract

Command

```
:FUNCtion<N>:SUBTract <operand>,<operand>
```

The :FUNCtion<N>:SUBTract command defines a function that algebraically subtracts the second operand from the first operand.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example defines a function that subtracts waveform memory 1 from channel 1.

```
10  OUTPUT 707;":FUNCTION1:SUBTRACT CHANNEL1,WMEMORY1"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

VERSus

Command

```
:FUNCtion<N>:VERSus <operand>,<operand>
```

The :FUNCtion<N>:VERSus command defines a function for an X-versus-Y display. The first operand defines the Y axis and the second defines the X axis. The Y-axis range and offset are initially equal to that of the first operand, and you can adjust them with the RANGe and OFFSet commands in this subsystem.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is an integer, 1 - 4.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E12.

Example

This example defines function 1 as an X-versus-Y display. Channel 1 is the X axis and waveform memory 2 is the Y axis.

```
10  OUTPUT 707;":FUNCTION1:VERSUS WMEMORY2,CHANNEL1"
20  END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

VERTical

Command

```
:FUNCtion<N>:VERTical {AUTO | MANual}
```

The :FUNCtion<N>:VERTical command sets the vertical scaling mode of the specified function to either AUTO or MANual.

This command also contains the following commands and queries:

- OFFset
- RANge

<N> An integer, 1 - 4, representing the selected function.

Query

```
:FUNCtion<N>:VERTical?
```

The :FUNCtion<N>:VERTical? query returns the current vertical scaling mode of the specified function.

Returned Format

```
[:FUNCtion<N>:VERTical] {AUTO | MANual}<NL>
```

Example

This example places the current state of the vertical tracking of function 1 in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":FUNCTION1:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

VERTical:OFFSet

Command

```
:FUNCtion<N>:VERTical:OFFSet <offset_value>
```

The :FUNCtion<N>:VERTical:OFFSet command sets the voltage represented at center screen for the selected function. This automatically changes the mode from auto to manual.

`<N>` An integer, 1 - 4, representing the selected function.

`<offset_value>` A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited only to being within the vertical range that can be represented by the function data.

Query

```
:FUNCtion<N>:VERTical:OFFset?
```

The :FUNCtion<N>:VERTical:OFFSet? query returns the current offset value of the selected function.

Returned Format

```
[:FUNCtion<N>:VERTical:OFFset] <offset_value><NL>
```

Example

This example places the current offset setting for function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

VERTical:RANGe

Command

```
:FUNCtion<N>:VERTical:RANGe <full_scale_range>
```

The :FUNCtion<N>:VERTical:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual, if the oscilloscope is not already in manual mode.

`<N>` An integer, 1 - 4, representing the selected function.

`<full_scale_range>` A real number for the full-scale vertical range, from -100E15 to 100E15.

Query

```
:FUNCtion<N>:VERTical:RANGe?
```

The :FUNCtion<N>:VERTical:RANGe? query returns the current range setting of the specified function.

Returned Format

```
[:FUNCtion<N>:VERTical:RANGe] <range><NL>
```

Example

This example places the current vertical range setting of function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

16

Hardcopy Commands

Hardcopy Commands

The HARDcopy subsystem commands set various parameters for printing the screen. The print sequence is activated when the root level command :PRINt is sent.

These HARDcopy commands and queries are implemented in the Infiniium Oscilloscopes:

- AREA
- DPRinter
- FACTors
- IMAGe
- PRINTers?

AREA

Command

```
:HARDcopy:AREA {GRATicule | SCReen}
```

The :HARDcopy:AREA command selects which data from the screen is to be printed. When you select GRATicule, only the graticule area of the screen is printed (this is the same as choosing Waveforms Only in the Configure Printer dialog box). When you select SCReen, the entire screen is printed.

Example

This example selects the graticule for printing.

```
10  OUTPUT 707;":HARDCOPY:AREA GRATICULE"
20  END
```

Query

```
:HARDcopy:AREA?
```

The :HARDcopy:AREA? query returns the current setting for the area of the screen to be printed.

Returned Format

```
[:HARDcopy:AREA] {GRATicule | SCReen}<NL>
```

Example

This example places the current selection for the area to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:AREA?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

DPRinter

Command

```
:HARDcopy:DPRinter {<printer_number> | <printer_string>}
```

The :HARDcopy:DPRinter command selects the default printer to be used.

`<printer_number>` An integer representing the attached printer. This number corresponds to the number returned with each printer name by the :HARDcopy:PRINters? query.

`<printer_string>` A string of alphanumeric characters representing the attached printer.

The :HARDcopy:DPRinter command specifies a number or string for the printer attached to the oscilloscope. The printer string must exactly match the character strings in the File->Print Setup dialog boxes, or the strings returned by the :HARDcopy:PRINters? query.

Examples

This example sets the default printer to the second installed printer returned by the :HARDcopy:PRINters? query.

```
10  OUTPUT 707;":HARDCOPY:DPRINTER 2"
20  END
```

This example sets the default printer to the installed printer with the name "HP Laser".

```
10  OUTPUT 707;":HARDCOPY:DPRINTER ""HP Laser"""
20  END
```

Query

```
:HARDcopy:DPRinter?
```

The :HARDcopy:DPRinter? query returns the current printer number and string.

Returned Format

```
[:HARDcopy:DPRinter?]
{<printer_number>,<printer_string>,DEFAULT}<NL>
```

Or, if there is no default printer (no printers are installed), only a <NL> is returned.

Example

This example places the current setting for the hard copy printer in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:DPRinter?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

Programs Must Wait After Changing the Default Printer

It takes several seconds to change the default printer. Any programs that try to set the default printer must wait (10 seconds is a safe amount of time) for the change to complete before sending other commands. Otherwise, the oscilloscope will become unresponsive.

FACTors

Command

```
:HARDcopy:FACTors {{ON | 1} | {OFF | 0}}
```

The :HARDcopy:FACTors command determines whether the oscilloscope setup factors will be appended to screen or graticule images. FACTors ON is the same as choosing Include Setup Information in the Configure Printer dialog box.

Example

This example turns on the setup factors.

```
10  OUTPUT 707;":HARDCOPY:FACTORS ON"
20  END
```

Query

```
:HARDcopy:FACTors?
```

The :HARDcopy:FACTors? query returns the current setup factors setting.

Returned Format

```
[:HARDcopy:FACTors] {1 | 0}<NL>
```

Example

This example places the current setting for the setup factors in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:FACTORS?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

IMAGe

Command

```
:HARDcopy:IMAGe {NORMal | INVert}
```

The :HARDcopy:IMAGe command prints the image normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Colors in the Configure Printer dialog box.

Example

This example sets the hard copy image output to normal.

```
10  OUTPUT 707;":HARDCOPY:IMAGE NORMAL"
20  END
```

Query

```
:HARDcopy:IMAGe?
```

The :HARDcopy:IMAGe? query returns the current image setting.

Returned Format

```
[:HARDcopy:IMAGe] {NORMal | INVert}<NL>
```

Example

This example places the current setting for the hard copy image in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:IMAGE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

PRINters?

Query

```
:HARDcopy:PRINters?
```

The :HARDcopy:PRINters? query returns the currently available printers.

Returned Format

```
[:HARDcopy:PRINters?]
<printer_count><NL><printer_data><NL>[,<printer_data><NL>]
```

<printer_count> The number of printers currently installed.

<printer_data> The printer number and the name of an installed printer. The word DEFAULT appears next to the printer that is the currently selected default printer.

The <printer_data> return string has the following format:
<printer_number>,<printer_string>{,DEFAULT}

Example

This example places the number of installed printers into the variable Count, loops through it that number of times, and prints the installed printer names to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:PRINTERS?"
30  ENTER 707;Count
40  IF Count>0 THEN
50  FOR Printer_number=1 TO Count
60  ENTER 707;Setting$
70  PRINT Setting$
80  NEXT Printer_number
90  END IF
100 END
```

17

Histogram Commands

Histogram Commands

The HISTogram commands and queries control the histogram features. A histogram is a probability distribution that shows the distribution of acquired data within a user-definable histogram window.

You can display the histogram either vertically, for voltage measurements, or horizontally, for timing measurements.

The most common use for histograms is measuring and characterizing noise or jitter on displayed waveforms. Noise is measured by sizing the histogram window to a narrow portion of time and observing a veritcal histogram that measures the noise on a waveform. Jitter is measured by sizing the histogram window to a narrow portion of voltage and observing a horizontal histogram that measures the jitter on an edge.

These HISTogram commands and queries are implemented in the Infiniium Oscilloscopes:

- AXIS
- MODE
- SCALe:SIZE
- WINDow:DEFault
- WINDow:SOURce
- WINDow:X1Position|LLIMit
- WINDow:X2Position|RLIMit
- WINDow:Y1Position|TLIMit
- WINDow:Y2Position|BLIMit

Histograms and the database

The histograms, mask testing, and color grade persistence use a specific database that uses a different memory area from the waveform record for each channel. When any of these features are turned on, the oscilloscope starts building the database. The database is the size of the graticule area. Behind each pixel is a 21-bit counter that is incremented each time data from a channel or function hits a pixel. The maximum count (saturation) for each counter is 2,097,151. You can use the DISPlay:CGRade:LEVels command to see if any of the counters are close to saturation.

The database continues to build until the oscilloscope stops acquiring data or all both features (color grade persistence and histograms) are turned off. You can clear the database by turning off all three features that use the database.

The database does not differentiate waveforms from different channels or functions. If three channels are on and the waveform from each channel happens to light the same pixel at the same time, the counter is incremented by three. However, it is not possible to tell how many hits came from each waveform. To separate waveforms, you can position the waveforms vertically with the channel offset. By separating the waveforms, you can avoid overlapping data in the database caused by multiple waveforms. Even if the display is set to show only the most recent acquisition, the database keeps track of all pixel hits while the database is building.

Remember that color grade persistence, mask testing, and histograms all use the same database. Suppose that the database is building because color grade persistence is ON; when mask testing or histograms are turned on, they can use the information already established in the database as though they had been turned on the entire time.

To avoid erroneous data, clear the display after you change oscilloscope setup conditions or DUT conditions and acquire new data before extracting measurement results.

AXIS

Command

```
:HISTogram:AXIS {VERTical | HORizontal}
```

The :HISTogram:AXIS command selects the type of histogram. A horizontal histogram can be used to measure time related information like jitter. A vertical histogram can be used to measure voltage related information like noise.

Example

This example defines a vertical histogram.

```
10 OUTPUT 707;":HISTOGRAM:AXIS VERTICAL"
20 END
```

Query

```
:HISTogram:AXIS?
```

The :HISTogram:AXIS? query returns the currently selected histogram type.

Returned Format

```
[:HISTogram:AXIS] {VERTical | HORizontal}<NL>
```

Example

This example returns the histogram type and prints it to the computer's screen.

```
10 DIM Axis$[50]
20 OUTPUT 707;":HISTOGRAM:AXIS?"
30 ENTER 707;Axis$
40 PRINT Axis$
50 END
```

MODE

Command

```
:HISTogram:MODE {OFF | MEASurement | WAVeforms}
```

The MEASurement parameter is only available when the E2681A Jitter Analysis option is installed.

The :HISTogram:MODE command selects the histogram mode. The histogram may be off, set to track the waveforms, or set to track the measurement when the E2681A Jitter Analysis Software is installed. When the E2681A Jitter Analysis Software is installed, sending the :MEASure:JITTer:HISTogram ON command will automatically set :HISTOgram:MODE to MEASurement.

Example

This example sets the histogram mode to track the waveform.

```
10 OUTPUT 707;":HISTOGRAM:MODE WAVEFORM"
20 END
```

Query

```
:HISTogram:MODE?
```

The :HISTogram:MODE? query returns the currently selected histogram mode.

Returned Format

```
[:HISTogram:MODE] {OFF | MEASurement | WAVeform}<NL>
```

Example

This example returns the result of the mode query and prints it to the computer's screen.

```
10 DIM Mode$[10]
20 OUTPUT 707;":HISTOGRAM:MODE?"
30 ENTER 707;Mode$
40 PRINT Mode$
50 END
```

SCALe:SIZE

Command

```
:HISTogram:SCALe:SIZE <size>
```

The :HISTogram:SCALe:SIZE command sets histogram size for vertical and horizontal mode.

`<size>` The size is from 1.0 to 8.0 for the horizontal mode and from 1.0 to 10.0 for the vertical mode.

Example

This example sets the histogram size to 3.5.

```
10 OUTPUT 707;":HISTOGRAM:SCALE:SIZE 3.5"
20 END
```

Query

```
:HISTogram:SCALe:SIZE?
```

The :HISTogram:SCALe:SIZE? query returns the correct size of the histogram.

Returned Format

```
[:HISTogram:SCALe:SIZE] <size><NL>
```

Example

This example returns the result of the size query and prints it to the computer's screen.

```
10 DIM Size$[50]
20 OUTPUT 707;":HISTOGRAM:SCALE:SIZE?"
30 ENTER 707;Size$
40 PRINT Size$
50 END
```

WINDow:DEFault

Command

```
:HISTogram:WINDow:DEFault
```

The :HISTogram:WINDow:DEFault command positions the histogram markers to a default location on the display. Each marker will be positioned one division off the left, right, top, and bottom of the display.

Example

This example sets the histogram window to the default position.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:DEFAULT"
20 END
```

WINDow:SOURce

Command

```
:HISTogram:WINDow:SOURce {CHANnel<N> | FUNCtion<N> | WMEMory<N>}
```

The :HISTogram:WINDow:SOURce command selects the source of the histogram window. The histogram window will track the source's vertical and horizontal scale.

<N> An integer, 1 through 4.

Example

This example sets the histogram window's source to Channel 1.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:SOURCE CHANNEL1"
20 END
```

Query

```
:HISTogram:WINDow:SOURce?
```

The :HISTogram:WINDow:SOURce? query returns the currently selected histogram window source.

Returned Format

```
[:HISTogram:WINDow:SOURce] {CHANnel<N> | FUNCtion<N> | WMEMory<N>}<NL>
```

Example

This example returns the result of the window source query and prints it to the computer's screen.

```
10 DIM Winsour$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:SOURCE?"
30 ENTER 707;Winsour$
40 PRINT Winsour$
50 END
```

WINDow:LLIMit

Command `:HISTogram:WINDow:LLIMit <left_limit>`

The :HISTogram:WINDow:LLIMit command moves the Ax marker (left limit) of the histogram window. The histogram window determines the portion of the display used to build the database for the histogram. The histogram window markers will track the scale of the histogram window source.

`<left_limit>` A real number that represents the left boundary of the histogram window.

Example This example sets the left limit position to -200 microseconds.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:LLIMit -200E-6"
20 END
```

Query `:HISTogram:WINDow:LLIMit?`

The :HISTogram:WINDow:LLIMit? query returns the value of the left limit histogram window marker.

Returned Format `[:HISTogram:WINDow:LLIMit] <left_limit><NL>`

Example This example returns the result of the left limit position query and prints it to the computer's screen.

```
10 DIM LL$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:LLIMIT?"
30 ENTER 707;LL$
40 PRINT LL$
50 END
```

WINDow:RLIMit

Command

```
:HISTogram:WINDow:RLIMit <right_limit>
```

The :HISTogram:WINDow:RLIMit command moves the Bx marker (right limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

`<right_limit>` A real number that represents the right boundary of the histogram window.

Example

This example sets the Bx marker to 200 microseconds.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:RLIMit 200E-6"
20 END
```

Query

```
:HISTogram:WINDow:RLIMit?
```

The :HISTogram:WINDow:RLIMit? query returns the value of the right histogram window marker.

Returned Format

```
[:HISTogram:WINDow:RLIMit] <right_limit><NL>
```

Example

This example returns the result of the Bx position query and prints it to the computer's screen.

```
10 DIM RL$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:RLIMit?"
30 ENTER 707;RL$
40 PRINT RL$
50 END
```

WINDow:BLIMit

Command

```
:HISTogram:WINDow:BLIMit <bottom_limit>
```

The :HISTogram:WINDow:BLIMit command moves the Ay marker (bottom limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

`<bottom_limit>` A real number that represents the bottom boundary of the histogram window.

Example

This example sets the position of the Ay marker to -250 mV.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:BLIMit -250E-3"
20 END
```

Query

```
:HISTogram:WINDow:BLIMit?
```

The :HISTogram:WINDow:BLIMit? query returns the value of the Ay histogram window marker.

Returned Format

```
[:HISTogram:WINDow:BLIMit] <bottom_limit><NL>
```

Example

This example returns the result of the Ay position query and prints it to the computer's screen.

```
10 DIM BL$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:BLIMit?"
30 ENTER 707;BL$
40 PRINT BL$
50 END
```

WINDow:TLIMit

Command

```
:HISTogram:WINDow:TLIMit <top_limit>
```

The :HISTogram:WINDow:TLIMit command moves the By marker (top limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

`<top_limit>` A real number that represents the top boundary of the histogram window.

Example

This example sets the position of the By marker to 250 mV.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:TLIMit 250E-3"
20 END
```

Query

```
:HISTogram:WINDow:TLIMit?
```

The :HISTogram:WINDow:TLIMit? query returns the value of the By histogram window marker.

Returned Format

```
[:HISTogram:WINDow:TLIMit] <top_limit><NL>
```

Example

This example returns the result of the By position query and prints it to the computer's screen.

```
10 DIM TL$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:TLIMit?"
30 ENTER 707;TL$
40 PRINT TL$
50 END
```

InfiniiScan (ISCan) Commands

InfiniiScan (ISCan) Commands

The ISCan commands and queries control the InfiniiScan feature of the oscilloscope. InfiniiScan provides several ways of searching through the waveform data to find unique events.

The ISCan subsystem contains the following commands:

- DELay
- MEASurement
- MODE
- NONMonotonic
- RUNT
- SERial
- ZONE

DELay

Command `:ISCan:DELay {OFF | <delay_time>}`

The :ISCan:DELay command sets the delay time from when the hardware trigger occurs and when InfiniiScan tries to find the waveform event that has been defined.

`OFF` Turns off the delay from the hardware trigger.

`<delay_time>` Sets the amount of time that the InfiniiScan trigger is delayed from the hardware trigger.

Example The following example causes the oscilloscope to delay by 1 µs.

```
10  OUTPUT 707;":ISCAN:DELay 1E-06"
20  END
```

Query `:ISCan:DELay?`

The query returns the current set delay value.

Returned Format `[:ISCan:DELay] {OFF | <delay_time>}<NL>`

Example The following example returns the current delay value and prints the result to the controller's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:DELAY?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

MEASurement:FAIL

Command

```
:ISCan:MEASurement:FAIL {INSide | OUTSide}
```

The :ISCan:MEASurement:FAIL command sets the fail condition for an individual measurement. The conditions for a test failure are set on the measurement selected by the :ISCan:MEASurement command.

When a measurement failure is detected by the limit test the oscilloscope triggers and the trigger action is executed.

`INSide` INside causes the oscilloscope to fail a test when the measurement results are within the parameters set by the :ISCan:MEASurement:LIMit and :ISCan:MEASurement:ULIMit commands.

`OUTSide` OUTside causes the oscilloscope to fail a test when the measurement results exceed the parameters set by the :ISCan:MEASurement:LLIMit and the :ISCan:MEASurement:ULIMit commands.

Example

The following example causes the oscilloscope to trigger when the measurements are outside the lower or upper limits.

```
10  OUTPUT 707;":ISCAN:MEASUREMENT:FAIL OUTSIDE"
20  END
```

Query

```
:ISCan:MEASurement:FAIL?
```

The query returns the current set fail condition.

Returned Format

```
[:ISCan:MEASurement:FAIL] {INSide | OUTSide}<NL>
```

Example

The following example returns the current fail condition and prints the result to the controller's screen.

```
10  DIM FAIL$[50]
20  OUTPUT 707;":ISCAN:MEASUREMENT:FAIL?"
30  ENTER 707;FAIL$
40  PRINT FAIL$
50  END
```

MEASurement:LLIMit

Command

```
:ISCan:MEASurement:LLIMit <lower_value>
```

The :ISCan:MEASurement:LLIMit (lower limit) command sets the lower test limit for the currenly selected measurement. The :ISCan:MEASurement command selects the measurement used.

`<lower_value>` A real number.

Example

The following example sets the lower test limit to 1.0.

```
10  OUTPUT 707;":ISCAN:MEASUREMENT:LLIMIT 1.0"
20  END
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the measurement limit to trigger when the signal is outside the specified limit.

Query

```
:ISCan:MEASurement:LLIMit?
```

The query returns the current value set by the command.

Returned Format

```
[:ISCan:MEASurement:LLIMit]<lower_value><NL>
```

Example

The following example returns the current lower test limit and prints the result to the controller's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:MEASUREMENT:LLIMIT?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

MEASurement

Command

```
:ISCan:MEASurement {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}
```

The :ISCan:MEASurement command selects the current source for Measurement Limit Test Trigger. It selects one of the active measurements as referred to by their position in the Measurement tab area at the bottom of the screen. Measurements are numbered from left to right in the Measurements tab area of the screen.

Example

The following example selects the first measurement as the source for the limit testing commands.

```
10  OUTPUT 707;":ISCAN:MEASUREMENT MEAS1"
20  END
```

Query

```
:ISCan:MEASurement?
```

The query returns the currently selected measurement source.

Returned Format

```
[:ISCan:MEASurement]{MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}
<NL>
```

Example

The following example returns the currently selected measurement source for the limit testing commands.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":ISCAN:MEASUREMENT?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

See Also

Measurements are started by the commands in the Measurement Subsystem.

MEASurement:ULIMit

Command

```
:ISCan:MEASurement:ULIMit <upper_value>
```

The :ISCan:MEASurement:ULIMit (upper limit) command sets the upper test limit for the active measurement currently selected by the :ISCan:MEASurement command.

`<upper_value>` A real number.

Example

The following example sets the upper limit of the currently selected measurement to 500 mV.

```
10  OUTPUT 707;":ISCAN:MEASUREMENT:ULIMIT 500E-3"
20  END
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the :ISCan:MEASurement:FAIL OUTside command to specify that the oscilloscope will trigger when the voltage exceeds 500 mV.

Query

```
:ISCan:MEASurement:ULIMit?
```

The query returns the current upper limit of the limit test.

Returned Format

```
[:ISCan:MEASurement:ULIMit] <upper_value><NL>
```

Example

The following example returns the current upper limit of the limit test and prints the result to the controller's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:MEASUREMENT:ULIMit?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

MODE

Command

```
:ISCan:MODE {OFF | MEASurement | NONMontonic |
RUNT | SERial | ZONE}
```

The :ISCan:MODE command selects the type of InfiniiScan trigger mode. The Measurement, Runt, Zone Qualify, and Non-monotonic Edge InfiniiScan modes can be set using this command.

`OFF` Turns off the InfiniiScan trigger mode.

`MEASurement` Sets the Measurement Limit trigger mode.

`NONMontonic` Sets the Non-monotonic edge trigger mode.

`RUNT` Sets the Runt trigger mode.

`SERial` Sets the Serial trigger mode.

`ZONE` Sets the Zone Qualify trigger mode.

Example

The following example selects the runt trigger.

```
10  OUTPUT 707;":ISCAN:MODE RUNT"
20  END
```

Query

```
:ISCan:MODE?
```

The query returns the currently selected IniniiScan trigger mode.

Returned Format

```
[:ISCan:MEASurement]{OFF | MEASurement | NONMonotonic |
RUNT | SERial | ZONE}<NL>
```

Example

The following example returns the currently selected InfiniiScan trigger mode.

```
10  DIM MODE$[50]
20  OUTPUT 707;":ISCAN:MODE?"
30  ENTER 707;MODE$
40  PRINT MODE$
50  END
```

NONMonotonic:EDGE

Command `:ISCan:NONMonotonic:EDGE {EITHer | FALLing | RISing}`

The :ISCan:NONMonotonic:EDGE command selects the rising edge, the falling edge, or either edge for the Non-monotonic edge trigger mode.

`EITHer` Sets the edge used by the Non-monotonic edge trigger to both rising and falling edges.

`FALLing` Sets the edge used by the Non-monotonic edge trigger to falling edges.

`RISing` Sets the edge used by the Non-monotonic edge trigger to rising edges.

Example The following example selects the falling edge non-monotonic trigger.

```
10  OUTPUT 707;":ISCAN:NONMONOTONIC:EDGE FALLING"
20  END
```

Query `:ISCan:NONMonotonic:EDGE?`

The query returns the currently selected edge type for the Non-Monotonic Edge trigger.

Returned Format `[:ISCan:NONMonotonic:EDGE]{EITHer | FALLing | RISing}<NL>`

Example The following example returns the currently selected edge type used for the Non-monotonic Edge trigger mode.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":ISCAN:NONMONOTONIC:EDGE?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

NONMonotonic:HYSTeresis

Command

```
:ISCan:NONMonotonic:HYSTeresis <value>
```

The :ISCan:NONMonotonic:HYSTeresis command sets the hysteresis value used for the Non-monotonic Edge trigger.

`<value>` is a real number for the hysteresis.

Example

The following example sets the hysteresis value used by the Non-monotonic trigger mode to 10 mV.

```
10  OUTPUT 707;":ISCAN:NONMONOTONIC:HYSTERESIS 1E-2"
20  END
```

Query

```
:ISCan:NONMonotonic:HYSTersis?
```

The query returns the hysteresis value used by the Non-monotonic Edge trigger mode.

Returned Format

```
[:ISCan:NONMonotonic:HYSTeresis]<value><NL>
```

Example

The following example returns and prints the value of the hysteresis.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:NONMONOTONIC:HYSTERESIS?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

NONMonotonic:SOURce

Command

```
:ISCan:NONMonotonic:SOURce CHANnel<N>
```

The :ISCan:NONMonotonic:SOURce command sets the source used for the Non-monotonic Edge trigger.

<N> is an integer from 1-4.

Example

The following example sets the source used by the Non-monotonic trigger mode to channel 1.

```
10  OUTPUT 707;":ISCAN:NONMONOTONIC:SOURCE CHANNEL1"
20  END
```

Query

```
:ISCan:NONMonotonic:SOURce?
```

The query returns the source used by the Non-monotonic Edge trigger mode.

Returned Format

```
[:ISCan:NONMonotonic:SOURce]CHANnel<N><NL>
```

Example

The following example returns the currently selected source for the Non-monotonic Edge trigger mode.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":ISCAN:NONMONTONIC:SOURCE?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

RUNT:HYSTeresis

Command

```
:ISCan:RUNT:HYSTeresis <value>
```

The :ISCan:RUNT:HYSTeresis command sets the hysteresis value used for the Runt trigger.

`<value>` is a real number for the hysteresis.

Example

The following example sets the hysteresis value used by the Runt trigger mode to 10 mV.

```
10  OUTPUT 707;":ISCAN:RUNT:HYSTERESIS 1E-2"
20  END
```

Query

```
:ISCan:RUNT:HYSTersis?
```

The query returns the hysteresis value used by the Runt trigger mode.

Returned Format

```
[:ISCan:RUNT:HYSTeresis]<value><NL>
```

Example

The following example returns and prints the value of the hysteresis.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:RUNT:HYSTERESIS?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

RUNT:LLEVel

Command

```
ISCan:RUNT:LLEVel <lower_level>
```

The :ISCan:RUNT:LLEVel (lower level) command sets the lower level limit for the Runt trigger mode.

`<lower_level>` A real number.

Example

The following example sets the lower level limit to 1.0 V.

```
10  OUTPUT 707;":ISCAN:RUNT:LLEVel 1.0"
20  END
```

Query

```
:ISCan:RUNT:LLEVel?
```

The query returns the lower level limit set by the command.

Returned Format

```
[:ISCan:RUNT:LLEVel] <lower_level><NL>
```

Example

The following example returns the current lower level used by the Runt trigger and prints the result to the controller's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:RUNT:LLEVel?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

RUNT:SOURce

Command

```
:ISCan:RUNT:SOURce CHANnel<N>
```

The :ISCan:RUNT:SOURce command sets the source used for the Runt trigger.

<N> is an integer from 1-4.

Example

The following example sets the source used by the Runt trigger mode to channel 1.

```
10  OUTPUT 707;":ISCAN:RUNT:SOURCE CHANNEL1"
20  END
```

Query

```
:ISCan:RUNT:SOURce?
```

The query returns the source used by the Runt trigger mode.

Returned Format

```
[:ISCan:RUNT:SOURce]CHANnel<N><NL>
```

Example

The following example returns the currently selected source for the Runt trigger mode.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":ISCAN:RUNT:SOURCE?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

RUNT:ULEVel

Command

```
:ISCan:RUNT:ULEVel <upper_level>
```

The :ISCan:RUNT:ULEVel (upper level) command sets the upper level limit for the Runt trigger mode.

`<upper_level>` A real number.

Example

The following example sets the upper level value used by the Runt trigger mode to 500 mV.

```
10  OUTPUT 707;":ISCAN:RUNT:ULEVEL 500E-3"
20  END
```

Query

```
:ISCan:RUNT:ULEVel?
```

The query returns the current upper level value used by the Runt trigger.

Returned Format

```
[:ISCan:RUNT:ULEVel] <upper_level><NL>
```

Example

The following example returns the current upper level used by the Runt trigger and prints the result to the controller's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":ISCAN:RUNT:ULEVel?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

SERial:PATTern

Command `:ISCan:SERial:PATTern <pattern>`

The :ISCan:SERial:PATTern command sets the pattern used for the Serial trigger.

`<pattern>` is a 1, 0, or X binary character string of up to 80 characters. The pattern can only be expressed in the binary format.

Example The following example sets the pattern used by the Serial trigger to 101100.

```
10  OUTPUT 707;":ISCAN:SERIAL:PATTERN ""101100"""
20  END
```

Query `:ISCan:SERial:PATTern?`

The query returns the pattern used by the Serial trigger mode.

Returned Format `[:ISCan:SERial:PATTern]<pattern><NL>`

Example The following example returns the currently selected pattern for the Serial trigger mode.

```
10  DIM PATTERN$[80]
20  OUTPUT 707;":ISCAN:SERIAL:PATTERN?"
30  ENTER 707;PATTERN$
40  PRINT PATTERN$
50  END
```

SERial:SOURce

Command

```
:ISCan:SERial:SOURce CHANnel<N>
```

The :ISCan:SERial:SOURce command sets the source used for the Serial trigger.

<N> is an integer from 1-4.

Example

The following example sets the source used by the Serial trigger mode to channel 1.

```
10  OUTPUT 707;":ISCAN:SERIAL:SOURCE CHANNEL1"
20  END
```

Query

```
:ISCan:SERial:SOURce?
```

The query returns the source used by the Serial trigger mode.

Returned Format

```
[:ISCan:SERial:SOURce]CHANnel<N><NL>
```

Example

The following example returns the currently selected source for the Serial trigger mode.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":ISCAN:SERIAL:SOURCE?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

ZONE<N>:MODE

Command

```
:ISCan:ZONE<N>:MODE {INTersect | NOTintersect}
```

The :ISCan:ZONE<N>:MODE command sets the Zone Qualify trigger mode. For the INTersect mode, the waveform must enter the zone region to qualify as a valid waveform. For NOTintersect mode, the waveform cannot enter a zone region to qualify as a valid waveform.

<N> is an integer from 1-4.

Example

The following example sets the mode to intersect for zone 1.

```
10  OUTPUT 707;":ISCAN:ZONE1:MODE INTERSECT"
20  END
```

Query

```
:ISCan:ZONE<N>:MODE?
```

The query returns the mode used by zone 1.

Returned Format

```
[:ISCan:ZONE<N>:MODE]{INTersect | NOTintersect}<NL>
```

Example

The following example returns the currently selected mode for zone 1.

```
10  DIM MODE$[50]
20  OUTPUT 707;":ISCAN:ZONE1:MODE?"
30  ENTER 707;MODE$
40  PRINT MODE$
50  END
```

ZONE<N>:PLACement

Command

```
:ISCan:ZONE<N>:PLACement
<width>,<height>,<x_center>,<y_center>
```

The :ISCan:ZONE<N>:PLACement command sets the location and size of a zone for the zone qualify trigger mode.

`<N>` is an integer from 1-4.

`<width>` a real number defining the width of a zone in seconds.

`<height>` is a real number defining the height of a zone in volts.

`<x_center>` is a real number defining the x coordinate of the center of the zone in seconds.

`<y_center>` is a real number defining the y coordinate of the center of the zone in volts.

Example

The following example sets the size of zone 1 to be 500 ps wide and 0.5 volts high and centered about the xy coordinate of 1.5 ns and 1 volt.

```
10  OUTPUT 707;":ISCAN:ZONE1:PLACEMENT 500e-12,0.5,1.5e-9,1"
20  END
```

Query

```
:ISCan:ZONE<N>:PLACement?
```

The query returns the placement values used by zone 1.

Returned Format

```
[:ISCan:ZONE<N>:PLACement]<width>,<height>,<x_center>,
<y_center><NL>
```

Example

The following example returns the current placement values for zone 1.

```
10  DIM PLACEMENT$[50]
20  OUTPUT 707;":ISCAN:ZONE1:PLACEMENT?"
30  ENTER 707;PLACEMENT$
40  PRINT PLACEMENT$
50  END
```

ZONE:SOURce

Command

```
:ISCan:ZONE:SOURce CHANnel<N>
```

The :ISCan:ZONE:SOURce command sets the source used for the zone qualify trigger.

<N> is an integer from 1-4.

Example

The following example sets the source used by the zone qualify trigger to channel 1.

```
10  OUTPUT 707;":ISCAN:ZONE:SOURCE CHANNEL1"
20  END
```

Query

```
:ISCan:ZONE:SOURce?
```

The query returns the source used by the zone qualify trigger.

Returned Format

```
[:ISCan:ZONE:SOURce]CHANnel<N><NL>
```

Example

The following example returns the currently selected source for zone qualify trigger.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":ISCAN:ZONE:SOURCE?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

ZONE<N>:STATe

Command `:ISCan:ZONE<N>:STATe {{ON | 1} | {OFF | 0}}`

The :ISCan:ZONE<N>:STATe command turns a zone off or on for the zone qualify trigger.

<N> is an integer from 1-4.

Example The following example turns on zone 2.

```
10  OUTPUT 707;":ISCAN:ZONE2:STATE ON"
20  END
```

Query `:ISCan:ZONE<N>:STATe?`

The query returns the state value for a zone.

Returned Format `[:ISCan:ZONE<N>:STATe]{1 | 0}<NL>`

Example The following example returns the current state value for zone 2.

```
10  DIM STATE$[50]
20  OUTPUT 707;":ISCAN:ZONE2:STATE?"
30  ENTER 707;STATE$
40  PRINT STATE$
50  END
```

19

Limit Test Commands

Limit Test Commands

The Limit Test commands and queries control the limit test features of the oscilloscope. Limit testing automatically compares measurement results with pass or fail limits. The limit test tracks up to four measurements. The action taken when the test fails is also controlled with commands in this subsystem.

The Limit Test subsystem contains the following commands:

- FAIL
- LLIMit (Lower Limit)
- MEASurement
- RESults?
- TEST
- ULIMit (Upper Limit)

FAIL

Command `:LTESt:FAIL {INSide | OUTSide}`

The :LTESt:FAIL command sets the fail condition for an individual measurement. The conditions for a test failure are set on the source selected with the last LTESt:MEASurement command.

When a measurement failure is detected by the limit test, the fail action conditions are executed, and there is the potential to generate an SRQ.

`INSide` FAIL INside causes the oscilloscope to fail a test when the measurement results are within the parameters set by the LLTESt:LIMit and LTESt:ULIMit commands.

`OUTSide` FAIL OUTside causes the oscilloscope to fail a test when the measurement results exceed the parameters set by LTESt:LLIMit and LTESt:ULIMit commands.

Example The following example causes the oscilloscope to fail a test when the measurements are outside the lower and upper limits.

```
10  OUTPUT 707;":LTEST:FAIL OUTSIDE"
20  END
```

Query `:LTESt:FAIL?`

The query returns the current set fail condition.

Returned Format `[:LTESt:FAIL] {INSide | OUTSide}<NL>`

Example The following example returns the current fail condition and prints the result to the controller's screen.

```
10  DIM FAIL$[50]
20  OUTPUT 707;":LTEST:FAIL?"
30  ENTER 707;FAIL$
40  PRINT FAIL$
50  END
```

LLIMit

Command

```
:LTESt:LLIMit <lower_value>
```

The :LTESt:LLIMit (Lower LIMit) command sets the lower test limit for the active measurement currently selected by the :LTESt:MEASurement command.

`<lower_value>` A real number.

Example

The following example sets the lower test limit to 1.0.

```
10  OUTPUT 707;":LTEST:LLIMIT 1.0"
20  END
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the limit test to fail when the signal is outside the specified limit.

Query

```
:LTESt:LLIMit?
```

The query returns the current value set by the command.

Returned Format

```
[:LTESt:LLIMit]<lower_value><NL>
```

Example

The following example returns the current lower test limit and prints the result to the controller's screen.

```
10  DIM LLIM$[50]
20  OUTPUT 707;":LTEST:LLIMIT?"
30  ENTER 707;LLIM$
40  PRINT LLIM$
50  END
```

MEASurement

Command

```
:LTESt:MEASurement {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}
```

The :LTESt:MEASurement command selects the current source for Limit Test for the ULIMit and LLIMit commands. It selects one of the active measurements as referred to by their position in the measurement window on the bottom of the screen. Measurements are numbered from left to right.

Example

The following example selects the first measurement as the source for the limit testing commands.

```
10  OUTPUT 707;":LTEST:MEASUREMENT MEAS1"
20  END
```

Query

```
:LTESt:MEASurement?
```

The query returns the currently selected measurement source.

Returned Format

```
[:LTESt:MEASurement]{MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}
<NL>
```

Example

The following example returns the currently selected measurement source for the limit testing commands.

```
10  DIM SOURCE$[50]
20  OUTPUT 707;":LTEST:MEASUREMENT?"
30  ENTER 707;SOURCE$
40  PRINT SOURCE$
50  END
```

See Also

Measurements are started in the Measurement Subsystem.

RESults?

Query

```
:LTESt:RESults? {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}
```

The query returns the measurement results for selected measurement. The values returned are the failed minimum value (Fail Min), the failed maximum value (Fail Max), and the total number of measurements made (# of Meas).

Returned Format

```
[:LTESt:RESults] <fail_min>,<fail_max>,<num_meas><NL>
```

`<fail_min>` A real number representing the total number of measurements that have failed the minimum limit.

`<fail_max>` A real number representing the total number of measurements that have failed the maximum limit.

`<num_meas>` A real number representing the total number of measurements that have been made.

Example

The following example returns the values for the limit test of measurement 1.

```
10  DIM RESULTS$[50]
20  OUTPUT 707;":LTEST:RESults? MEAS1"
30  ENTER 707;RESULTS$
40  PRINT RESULTS$
50  END
```

See Also

Measurements are started in the Measurement Subsystem.

TEST

Command

```
:LTESt:TEST {{ON | 1} {OFF | 0}}
```

The LTESt:TEST command controls the execution of the limit test function. ON allows the limit test to run over all of the active measurements. When the limit test is turned on, the limit test results are displayed on screen in a window below the graticule.

Example

The following example turns off the limit test function.

```
10  OUTPUT 707;":LTEST:TEST OFF"
20  END
```

Query

```
:LTESt:TEST?
```

The query returns the state of the TEST control.

Returned Format

```
[:LTESt:TEST] {1 | 0} <NL>
```

Example

The following example returns the current state of the limit test and prints the result to the controller's screen.

```
10  DIM TEST$[50]
20  OUTPUT 707;":LTEST:TEST?"
30  ENTER 707;TEST$
40  PRINT TEST$
50  END
```

The result of the MEAS:RESults? query has two extra fields when LimitTESt:TEST is ON (failures, total). Failures is a number and total is the total number of measurements made.

ULIMit

Command

```
:LTESt:ULIMit <upper_value>
```

The :LTESt:ULIMit (Upper LIMit) command sets the upper test limit for the active measurement currently selected by the last :LTESt:MEASurement command.

`<upper_value>` A real number.

Example

The following example sets the upper limit of the currently selected measurement to 500 milli.

```
10  OUTPUT 707;":LTEST:ULIMIT 500E-3"
20  END
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the LTESt:FAIL OUTside command to specify that the limit subsystem will fail a measurement when the voltage exceeds 500 mV.

Query

```
:LTESt:ULIMit?
```

The query returns the current upper limit of the limit test.

Returned Format

```
[:LTESt:ULIMit] <upper_value><NL>
```

Example

The following example returns the current upper limit of the limit test and prints the result to the controller's screen.

```
10  DIM ULIM$[50]
20  OUTPUT 707;":LTEST:ULIMIT?"
30  ENTER 707;ULIM$
40  PRINT ULIM$
50  END
```

20

Marker Commands

Marker Commands

The commands in the MARKer subsystem specify and query the settings of the time markers (X axis) and current measurement unit markers (volts, amps, and watts for the Y axis). You typically set the Y-axis measurement units using the :CHANnel:UNITs command.

These MARKer commands and queries are implemented in the Infiniium Oscilloscopes:

- CURsor?
- MODE
- X1Position
- X2Position
- X1Y1source
- X2Y2source
- XDELta?
- Y1Position
- Y2Position
- YDELta?

Guidelines for Using Queries in Marker Modes

In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other queries, such as the X1Position? and X2Position?, and VSTArt? and VSTOp? queries. If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected.

CURSor?

Query

```
:MARKer:CURSor? {DELTa | STARt | STOP}
```

The :MARKer:CURSor? query returns the time and current measurement unit values of the specified marker (if markers are in Track Waveforms mode) as an ordered pair of time and measurement unit values.

- If DELTA is specified, the value of delta Y and delta X are returned.
- If START is specified, marker A's x-to-y positions are returned.
- If STOP is specified, marker B's x-to-y positions are returned.

Returned Format

```
[:MARKer:CURSor] {DELTa | STARt | STOP}
{<Ax, Ay> | <Bx, By> | <deltaX, deltaY>}<NL>
```

Example

This example returns the current position of the X cursor and measurement unit marker 1 to the string variable, Position$. The program then prints the contents of the variable to the computer's screen.

```
10  DIM Position$[50]!Dimension variable
20  OUTPUT 707;":MARKER:CURSOR? START"
30  ENTER 707;Position$
40  PRINT Position$
50  END
```

C A U T I O N

The :MARKer:CURSor? query may change marker mode and results.
In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other marker queries, such as the X1Position? and X2Position?, and VSTArt? and VSTOp? queries.

If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected. In addition, measurement results may not be what you expected.

MODE

Command

```
:MARKer:MODE {OFF | MANual | WAVeform |
MEASurement | HISTogram | MTESt}
```

The :MARKer:MODE command sets the marker mode.

OFF	Removes the marker information from the display.
MANual	Enables manual placement of markers A and B.
WAVeform	Tracks the current waveform.
MEASurement	Tracks the most recent measurement.
HISTogram	Tracks the the histogram window.
MEASurement	Tracks the mask scale.

Example

This example sets the marker mode to waveform.

```
10  OUTPUT 707;":MARKER:MODE WAVEFORM"
20  END
```

Query

```
:MARKer:MODE?
```

The :MARKer:MODE? query returns the current marker mode.

Returned Format

```
[:MARKer:MODE] {OFF | MANual | WAVeform | MEASurement |
HISTogram | MTESt}<NL>
```

Example

This example places the current marker mode in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":MARKER:MODE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

X1Position

Command `:MARKer:X1Position <Ax_position>`

The :MARKer:X1Position command sets the Ax marker position, and moves the Ax marker to the specified time with respect to the trigger time.

`<Ax_position>` A real number for the time at the Ax marker in seconds.

Example This example sets the Ax marker to 90 ns.

```
10  OUTPUT 707;":MARKER:X1POSITION 90E-9"
20  END
```

Query `:MARKer:X1Position?`

The :MARKer:X1Position? query returns the time at the Ax marker position.

Returned Format `[:MARKer:X1Position] <Ax_position><NL>`

Example This example returns the current setting of the Ax marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:X1POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

See Also :MARKer:TSTArt

X2Position

Command

```
:MARKer:X2Position <Bx_position>
```

The :MARKer:X2Position command sets the Bx marker position and moves the Bx marker to the specified time with respect to the trigger time.

`<Bx_position>` A real number for the time at the Bx marker in seconds.

Example

This example sets the Bx marker to 90 ns.

```
10  OUTPUT 707;":MARKER:X2POSITION 90E-9"
20  END
```

Query

```
:MARKer:X2Position?
```

The :MARKer:X2Position? query returns the time at Bx marker in seconds.

Returned Format

```
[:MARKer:X2Position] <Bx_position><NL>
```

Example

This example returns the current position of the Bx marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:X2POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

X1Y1source

Command

```
:MARKer:X1Y1source {CHANnel<N> | FUNCtion<N> |
WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}
```

The :MARKer:X1Y1source command sets the source for the Ax and Ay markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

Example

This example selects channel 1 as the source for markers Ax and Ay.

```
10  OUTPUT 707;":MARKER:X1Y1SOURCE CHANNEL1"
20  END
```

Query

```
:MARKer:X1Y1source?
```

The :MARKer:X1Y1source? query returns the current source for markers Ax and Ay.

Returned Format

```
[:MARKer:X1Y1source] {CHANnel<N> | FUNCtion<N> | WMEMory<N>
| CLOCk | MTRend | MSPectrum | EQUalized}<NL>
```

Example

This example returns the current source selection for the Ax and Ay markers to the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
```

```
20  OUTPUT 707;":MARKER:X1Y1SOURCE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

X2Y2source

Command

```
:MARKer:X2Y2source {CHANnel<N> | FUNCtion<N> |
WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}
```

The :MARKer:X2Y2source command sets the source for the Bx and By markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

Example

This example selects channel 1 as the source for markers Bx and By.

```
10  OUTPUT 707;":MARKER:X2Y2SOURCE CHANNEL1"
20  END
```

Query

```
:MARKer:X2Y2source?
```

The :MARKer:X2Y2source? query returns the current source for markers Bx and By.

Returned Format

```
[:MARKer:X2Y2source] {CHANnel<N> | FUNCtion<N> | WMEMory<N>
| CLOCk | MTRend | MSPectrum | EQUalized}<NL>
```

Example

This example returns the current source selection for the Bx and By markers to the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
```

```
20  OUTPUT 707;":MARKER:X2Y2SOURCE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

XDELta?

Query `:MARKer:XDELta?`

The :MARKer:XDELta? query returns the time difference between Ax and Bx time markers.

Xdelta = time at Bx – time at Ax

Returned Format `[:MARKer:XDELta] <time><NL>`

`<time>` Time difference between Ax and Bx time markers in seconds.

Example This example returns the current time between the Ax and Bx time markers to the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:XDELTA?"
30  ENTER 707;Time
40  PRINT Time
50  END
```

Y1Position

Command

```
:MARKer:Y1Position <Ay_position>
```

The :MARKer:Y1Position command sets the Ay marker position on the specified source.

`<Ay_position>` A real number for the current measurement unit value at Ay (volts, amps, or watts).

Example

This example sets the Ay marker to 10 mV.

```
10  OUTPUT 707;":MARKER:Y1POSITION 10E-3"
20  END
```

Query

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns the current measurement unit level at the Ay marker position.

Returned Format

```
[:MARKer:Y1Position] <Ay_position><NL>
```

Example

This example returns the current setting of the Ay marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:Y1POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

Y2Position

Command

```
:MARKer:Y2Position <By_position>
```

The :MARKer:Y2Position command sets the By marker position on the specified source.

`<By_position>` A real number for the current measurement unit value at By (volts, amps, or watts).

Example

This example sets the By marker to -100 mV.

```
10  OUTPUT 707;":MARKER:Y2POSITION -100E-3"
20  END
```

Query

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns the current measurement unit level at the By marker position.

Returned Format

```
[:MARKer:Y2Position] <By_position><NL>
```

Example

This example returns the current setting of the By marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:Y2POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

YDELta?

Query `:MARKer:YDELta?`

The :MARKer:YDELta? query returns the current measurement unit difference between Ay and By.

Ydelta = value at By – value at Ay

Returned Format `[:MARKer:YDELta] <value><NL>`

`<value>` Measurement unit difference between Ay and By.

Example This example returns the voltage difference between Ay and By to the numeric variable, Volts, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:YDELTA?"
30  ENTER 707;Volts
40  PRINT Volts
50  END
```

21

Mask Test Commands

Mask Test Commands

The MTESt subsystem commands and queries control the mask test features. Mask Testing automatically compares measurement results with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

These MTESt commands and queries are implemented in the Infiniium Oscilloscopes. The FOLDing command is only available when the E2688A Clock Recovery Software is installed.

- ALIGn
- AlignFIT
- AMASk:CREate
- AMASk:SOURce
- AMASk:SAVE | STORe
- AMASk:UNITs
- AMASk:XDELta
- AMASk:YDELta
- AUTO
- AVERage
- AVERage:COUNt
- COUNt:FAILures?
- COUNt:FUI?
- COUNt:FWAVeforms?
- COUNt:UI?
- COUNt:WAVeforms?
- DELete
- ENABle
- FOLDing (Clock Recovery software only)
- HAMPlitude
- IMPedance
- INVert
- LAMPlitude

- LOAD
- NREGions?
- PROBe:IMPedance?
- RUMode
- RUMode:SOFailure
- SCALe:BIND
- SCALe:X1
- SCALe:XDELta
- SCALe:Y1
- SCALe:Y2
- SOURce
- STARt | STOP
- STIMe
- TITLe?
- TRIGger:SOURce

ALIGn

Command

`:MTESt:ALIGn`

The :MTESt:ALIGn command automatically aligns and scales the mask to the current waveform on the display. The type of mask alignment performed depends on the current setting of the Use File Setup When Aligning control. See the :MTESt:AUTO command for more information.

Example

This example aligns the current mask to the current waveform.

```
10 Output 707;":MTEST:ALIGN"
20 END
```

AlignFIT

Command

```
:MTESt:AlignFIT {EYEAMI | EYECMI | EYENRZ | FANWidth
| FAPeriod | FAPWidth | FYNWidth | FYPWidth | NONE
| NWIDth | PWIDth | TMAX | TMIN}
```

The :MTESt:AlignFIT command specifies the alignment type for aligning a mask to a waveform. The pulse mask standard has rules that determine which controls the oscilloscope can adjust or change during the alignment process. An X in a column indicates that the control can be adjusted for each of the alignment types of Table 21-1.

Table 21-1 Available Alignment Types

Alignment Type	Waveform Type	Horizontal Position	0 Level Voltage	1 Level Voltage	Vertical Offset	Invert Waveform
EYEAMI	**AMI**	**X**	**X**	**X**		
EYECMI	**CMI**	**X**	**X**	**X**		
EYENRZ	**NRZ**	**X**	**X**	**X**		
FANWidth	**Negative**	**X**			**X**	**X**
FAPeriod	**Full Period**	**X**	**X**			
FAPWidth	**Positive**	**X**			**X**	**X**
FYNWidth	**Negative**	**X**	**X**			**X**
FYPWidth	**Positive**	**X**	**X**			**X**
NONE	**Automask**					
NWIDth	**Negative Pulse**	**X**	**X**	**X**		**X**
PWIDth	**Positive Pulse**	**X**	**X**	**X**		**X**
TMAX	**Positive Sine Pulse**	**X**	**X**	**X**		**X**
TMIN	**Negative Sine Pulse**	**X**	**X**	**X**		**X**

Example

This example specifies the alignment type to be EYEAMI.

```
10  Output 707;":MTEST:ALIGNFIT EYEAMI"
20  END
```

Query

```
:MTESt:AlignFIT?
```

The :MTEST:AlignFIT? query returns the alignment type used for the mask.

Returned Format

```
[:MTESt:AlignFIT] {EYEAMI | EYECMI | EYENRZ |
FANWidth | FAPeriod | FAPWidth | FYNWidth |
FYPWidth | NONE | NWIDth | PWIDth | TMAX | TMIN}<NL>
```

AMASk:CREate

Command

```
:MTESt:AMASk:CREate
```

The :MTESt:AMASk:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the AMASk:XDELta, AMASk:YDELta, and AMASk:UNITs commands. The mask only encompasses the portion of the waveform visible on the display, so you must ensure that the waveform is acquired and displayed consistently to obtain repeatable results.

The :MTESt:SOURce command selects the channel and should be set before using this command.

Example

This example creates an automask using the current XDELta and YDELta units settings.

```
10 OUTPUT 707;":MTEST:AMASK:CREATE"
20 END
```

AMASk:SOURce

Command

```
:MTESt:AMASk:SOURce CHANnel<number>
```

The :MTESt:AMASk:SOURce command selects the source for the interpretation of the AMASk:XDELta and AMASk:YDELta parameters when AMASk:UNITs is set to CURRent. When UNITs are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel:UNITs command, of the selected source. Suppose that UNITs are CURRent and that you set SOURce to CHANNEL1, which is using units of volts. Then you can define AMASk:XDELta in terms of volts and AMASk:YDELta in terms of seconds.

`<number>` An integer, 1 through 4.

Example

This example sets the automask source to Channel 1.

```
10 OUTPUT 707;"MTEST:AMASK:SOURCE CHANNEL1"
20 END
```

Query

```
:MTESt:AMASk:SOURce?
```

The :MTESt:AMASk:SOURce? query returns the currently set source.

Returned Format

```
[:MTESt:AMASk:SOURce] CHANnel<number><NL>
```

Example

This example gets the source setting for automask and prints the result on the computer display.

```
10 DIM Amask_source$[30]
20 OUTPUT 707;"MTEST:AMASK:SOURCE?"
30 ENTER 707;Amask_source$
40 PRINT Amask_source$
50 END
```

AMASk:SAVE | STORe

Command

```
: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.

`<filename>` An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The filename assumes the present working directory if a path does not precede the file name. The default save path is C:\Documents and Settings\All Users\Documents\Infiniium\masks. The filename must have a .msk or .MSK extension or the command will fail.

Example

This example saves the automask generated mask to a file named "FILE1.MSK".

```
10 OUTPUT 707;":MTEST:AMASK:SAVE""FILE1.MSK"""
20 END
```

AMASk:UNITs

Command `:MTESt:AMASk:UNITs {CURRent | DIVisions}`

The :MTESt:AMASk:UNITs command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by AMASk:XDELta and AMASk:YDELta commands.

`CURRent` When set to CURRent, the mask test subsystem uses the units as set by the :CHANnel:UNITs command, usually time for ΔX and voltage for ΔY.

`DIVisions` When set to DIVisions, the mask test subsystem uses the graticule as the measurement system, so tolerance settings are specified as parts of a screen division. The mask test subsystem maintains separate XDELta and YDELta settings for CURRent and DIVisions. Thus, XDELta and YDELta are not converted to new values when the UNITs setting is changed.

Example This example sets the measurement units for automasking to the current :CHANnel:UNITs setting.

```
10 OUTPUT 707;"MTEST:AMASK:UNITS CURRENT"
20 END
```

Query `:MTESt:AMASk:UNITs?`

The AMASk:UNITs query returns the current measurement units setting for the mask test automask feature.

Returned Format `[:MTESt:AMASk:UNITs] {CURRent | DIVision}<NL>`

Example This example gets the automask units setting, then prints the setting on the screen of the computer.

```
10 DIM Automask_units$[10]
20 OUTPUT 707;"MTEST:AMASK:UNITS?"
30 ENTER 707;Automask_units$
40 PRINT Automask_units$
50 END
```

AMASk:XDELta

Command

```
:MTESt:AMASk:XDELta <xdelta_value>
```

The :MTESt:AMASk:XDELta command sets the tolerance in the X direction around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to horizontal values of the waveform to determine the boundaries of the mask.

`<xdelta_value>` A value for the horizontal tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ±250 ms. If the setting for AMASk:UNITs is DIVisions, the same xdelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Example

This example sets the units to divisions and sets the ΔX tolerance to one-eighth of a division.

```
10 OUTPUT 707;"MTEST:AMASK:UNITS DIVISIONS"
20 OUTPUT 707;":MTEST:AMASK:XDELTA 125E-3"
30 END
```

Query

```
:MTESt:AMASk:XDELta?
```

The AMASk:XDELta? query returns the current setting of the ΔX tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the AMASk:UNITs query.

Returned Format

```
[:MTESt:AMASk:XDELta] <xdelta_value><NL>
```

Example

This example gets the measurement system units and ΔX settings for automasking from the oscilloscope and prints the results on the computer screen.

```
10 DIM Automask_units$[10]
20 DIM Automask_xdelta$[20]
30 OUTPUT 707;"MTEST:AMASK:UNITS?"
40 ENTER 707;Automask_units$
50 OUTPUT 707;":MTEST:AMASK:XDELTA?"
60 ENTER 707;Automask_xdelta$
70 PRINT Automask_units$
80 PRINT Automask_xdelta$
90 END
```

AMASk:YDELta

Command

```
:MTESt:AMASk:YDELta <ydelta_value>
```

The :MTESt:AMASk:YDELta command sets the vertical tolerance around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to vertical values of the waveform to determine the boundaries of the mask.

This command requires that mask testing be enabled, otherwise a settings conflict error message is displayed. See :MTESt:ENABle for information on enabling mask testing.

`<ydelta_value>` A value for the vertical tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ±250 mV. If the setting for AMASk:UNITs is DIVisions, the same ydelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Example

This example sets the units to current and sets the ΔY tolerance to 30 mV, assuming that the current setting specifies volts in the vertical direction.

```
10 OUTPUT 707;"MTEST:AMASK:UNITS CURRENT"
20 OUTPUT 707;":MTEST:AMASK:YDELTA 30E-3"
30 END
```

Query

```
:MTESt:AMASk:YDELta?
```

The AMASk:YDELta? query returns the current setting of the ΔY tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the AMASk:UNITs query.

Returned Format

```
[:MTESt:AMASk:YDELta] <ydelta_value><NL>
```

Example

This example gets the measurement system units and ΔY settings for automasking from the oscilloscope and prints the results on the computer screen.

```
10 DIM Automask_units$[10]
20 DIM Automask_ydelta$[20]
30 OUTPUT 707;"MTEST:AMASK:UNITS?"
40 ENTER 707;Automask_units$
50 OUTPUT 707;":MTEST:AMASK:YDELTA?"
60 ENTER 707;Automask_ydelta$
70 PRINT Automask_units$
80 PRINT Automask_ydelta$
90 END
```

AUTO

Command

```
:MTESt:AUTO {{ON | 1} | {OFF | 0}}
```

The :MTESt:AUTO command enables (ON) or disables (OFF) the Use File Setup When Aligning control. This determines which type of mask alignment is performed when the :MTESt:ALIGn command is sent. When enabled, the oscilloscope controls are changed to the values which are determined by the loaded mask file. This alignment guarantees that the aligned mask and any subsequent mask tests meet the requirements of the standard.

When disabled, the alignment is performed using the current oscilloscope settings. This may be useful when troubleshooting problems during the design phase of a project.

Example

This example enables the Use File Settings When Aligning control.

```
10 OUTPUT 707;"MTEST:AUTO ON"
20 END
```

Query

```
:MTESt:AUTO?
```

The :MTESt:AUTO? query returns the current value of the Use File Setup When Aligning control.

Returned Format

```
[:MTESt:AUTO] {1 | 0} <NL>
```

Example

```
10  OUTPUT 707;":MTEST:AUTO?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

AVERage

Command

```
:MTESt:AVERage {{ON | 1} | {OFF | 0}}
```

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

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

Averaging is not available in PDETect mode.

Example

This example turns averaging on.

```
10 OUTPUT 707;"MTEST:AVERAGE ON"
20 END
```

Query

```
:MTESt:AVERage?
```

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

Returned Format

```
[:MTESt:AVERage] {1 | 0} <NL>
```

Example

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

```
10  DIM Setting$[50]   !Dimension variable
20  OUTPUT 707;"MTEST:AVERAGE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

AVERage:COUNt

Command `:MTESt:AVERage:COUNt <count_value>`

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

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

`<count_value>` An integer, 2 to 65534, specifying the number of data values to be averaged.

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

```
10  OUTPUT 707;":MTESt:AVERage:COUNT 16"
20  END
```

Query `:MTESt:AVERage:COUNt?`

The :MTESt:AVERage:COUNt? query returns the currently selected count value.

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

`<value>` An integer, 2 to 65534, specifying the number of data values to be averaged.

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

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":MTEST:AVERAGE:COUNT?"
30  ENTER 707;Result
40  PRINT Result
50  END
```

COUNt:FAILures?

Query `:MTESt:COUNt:FAILures? REGion<number>`

The MTESt:COUNt:FAILures? query returns the number of failures that occurred within a particular mask region.

The value 9.999E37 is returned if mask testing is not enabled or if you specify a region number that is unused.

`<number>` An integer, 1 through 8, designating the region for which you want to determine the failure count.

Returned Format `[:MTESt:COUNt:FAILures] REGion<number><number_of_failures><NL>`

`<number_of_failures>` The number of failures that have occurred for the designated region.

Example This example determines the current failure count for region 3 and prints it on the computer screen.

```
10 DIM Mask_failures$[50]
20 OUTPUT 707;"MTEST:COUNT:FAILURES? REGION3"
30 ENTER 707;Mask_failures$
40 PRINT Mask_failures$
50 END
```

COUNt:FUI?

Query `:MTESt:COUNt:FUI?`

The MTESt:COUNt:FUI? query returns the number of unit interval failures that have occured.

Returned Format `[:MTESt:COUNt:FUI?] <unit_interval_failures> <NL>`

`<unit_interval_failures>` The number of unit interval failures.

Example This example determines the current number of unit interval failures and prints it to the computer screen.

```
10 DIM Failures$[50]
20 OUTPUT 707;"MTEST:COUNT:FUI?"
30 ENTER 707;Failures$
40 PRINT Failures$
50 END
```

COUNt:FWAVeforms?

Query

```
:MTESt:COUNt:FWAVeforms?
```

The :MTESt:COUNt:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms, so if you wish to determine failures by region number, use the COUNt:FAILures? query.

This count may not always be available. It is available only when the following conditions are true:

- Mask testing was turned on before the histogram or color grade persistence, and
- No mask changes have occurred, including scaling changes, editing, or new masks.

The value 9.999E37 is returned if mask testing is not enabled, or if you have modified the mask.

Returned Format

```
[:MTESt:COUNt:FWAVeforms] <number_of_failed_waveforms><NL>
```

`<number_of_failed_waveforms>` The total number of failed waveforms for the current test run.

Example

This example determines the number of failed waveforms and prints the result on the computer screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:FWAVEFORMS?
30 ENTER 707;Mask_fwaveforms$
40 PRINT Mask_fwaveforms$
50 END
```

COUNt:UI?

Query

```
:MTESt:COUNt:UI?
```

The MTESt:COUNt:UI? query returns the number of unit intervals that have been mask tested.

Returned Format

```
[:MTESt:COUNt:UI?] <unit_intervals_tested> <NL>
```

`<unit_intervals_tested>` The number of unit intervals tested.

Example

This example determines the current number of unit intervals tested and prints it to the computer screen.

```
10 DIM Unit_intervals$[50]
20 OUTPUT 707;"MTEST:COUNT:uUI?"
30 ENTER 707;Unit_intervals$
40 PRINT Unit_intervals$
50 END
```

COUNt:WAVeforms?

Query

```
:MTESt:COUNt:WAVeforms?
```

The :MTESt:COUNt:WAVeforms? query returns the total number of waveforms acquired in the current mask test run. The value 9.999E37 is returned if mask testing is not enabled.

Returned Format

```
[:MTESt:COUNt:WAVeforms] <number_of_waveforms><NL>
```

`<number_of_waveforms>` The total number of waveforms for the current test run.

Example

This example determines the number of waveforms acquired in the current test run and prints the result on the computer screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:WAVEFORMS?"
30 ENTER 707;Mask_waveforms
40 PRINT Mask_waveforms
50 END
```

DELete

Command

```
:MTESt:DELete
```

The :MTESt:DELete command clears the currently loaded mask.

Example

This example clears the currently loaded mask.

```
10 OUTPUT 707;"MTEST:DELETE"
20 END
```

ENABle

Command

```
:MTESt:ENABle {{ON | 1} | {OFF | 0}}
```

The :MTESt:ENABle command enables or disables the mask test features.

ON Enables the mask test features.

OFF Disables the mask test features.

Example

This example enables the mask test features.

```
10 OUTPUT 707;":MTEST:ENABLE ON"
20 END
```

Query

```
:MTESt:ENABle?
```

The :MTESt:ENABle? query returns the current state of mask test features.

Returned Format

```
[MTESt:ENABle] {1 | 0}<NL>
```

Example

This example places the current value of the mask test state in the numeric variable Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF
20  OUTPUT 707;":MTEST:ENABLE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

FOLDing

This command is only available when the E2688A Clock Recovery Software is installed.

Command

```
:MTESt:FOLDing {{ON | 1} | {OFF | 0}}
```

The :MTESt:FOLDing command enables (ON) or disables (OFF) the display of the real time eye. When enabled, an eye diagram of the data.

Example

This example enables the display of the real time eye.

```
10 OUTPUT 707;"MTEST:FOLDING ON"
20 END
```

Query

```
:MTESt:FOLDing?
```

The :MTESt:FOLDing? query returns the current state of clock recovery folding.

Returned Format

```
[:MTESt:FOLDing] {1 | 0} <NL>
```

Example

```
10  OUTPUT 707;":MTEST:FOLDING?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

FOLDing:BITS

This command is only available when the E2688A Clock Recovery Software is installed.

Command

```
:MTESt:FOLDing:BITS {BOTH | DEEMphasis | TRANsition}
```

The :MTESt:FOLDing:BITS command determines the type of data bits used to create the eye pattern. The transition bits are greater in amplitude than the deemphasis bits. The PCI Express standard requires that compliance mask testing be done for both bit types.

Example

This example sets bit type to transition bits.

```
10 OUTPUT 707;"MTEST:FOLDING:BITS TRANSITION"
20 END
```

Query

```
:MTESt:FOLDing:BITS?
```

The :MTESt:FOLDing:BITS? query returns the current setting of the real time eye bits.

Returned Format

```
[:MTESt:FOLDing:BITS] {BOTH | DEEMphasis | TRANsition} <NL>
```

Example

```
10  OUTPUT 707;":MTEST:FOLDING:BITS?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

HAMPlitude

Command

```
:MTESt:HAMPlitude <upper_limit>
```

The :MTESt:HAMPlitude command sets the maximum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the upper limit used during mask testing.

`<upper_limit>` A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

Example

This example sets the maximum pulse amplitude to 3.6 volts.

```
10  OUTPUT 707;"MTEST:HAMPLITUDE 3.6"
20  END
```

Query

```
:MTESt:HAMPlitude?
```

The :MTESt:HAMPlitude? query returns the current value of the maximum pulse amplitude.

Returned Format

```
[MTESt:HAMPlitude] <upper_limit><NL>
```

`<upper_limit>` A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

Example

This example returns the current upper pulse limit and prints it to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"  !Response headers off
20  OUTPUT 707;"MTEST:HAMPLITUDE?"
30  ENTER 707;ULimit
40  PRINT ULimit
50  END
```

IMPedance

Command

```
:MTESt:IMPedance {NONE | IMP75 | IMP100 | IMP110 | IMP120}
```

The :MTESt:IMPedance command sets the desired probe impedance of the channel being used for mask testing. This impedance value is used when starting a mask test to determine whether or not the correct Infiniium probe is connected and in the case of the E2621A if the switch is set to the correct impedance value.

Infiniium has an AutoProbe interface that detects probes that have Probe ID resistors. If one of these probes is connected to the channel being mask tested and is not the correct probe for the selected impedance, a warning dialog box appears when the mask test is started from the human interface.

This command is meant to be used in the setup section of a mask file.

`NONE` Disables the probe impedance check.

`IMP75` Enables the probe impedance check for the E2622A probe.

`IMP100` Enables the probe impedance check for the E2621A probe with the switch set to the 100 ohm position.

`IMP110` Enables the probe impedance check for the E2621A probe with the switch set to the 110 ohm position.

`IMP120` Enables the probe impedance check for the E2621A probe with the switch set to the 120 ohm position.

Example

This example sets the probe impedance of the channel being used for mask testing to 100 ohms.

```
10 OUTPUT 707;"MTEST:IMPEDANCE IMP100"
20 END
```

Query `:MTESt:IMPedance?`

The :MTESt:IMPedance? query returns the current value of the mask test impedance.

Returned Format `[:MTESt:IMPedance] {NONE | IMP75 | IMP100 | IMP110 | IMP120}<NL>`

Example This example returns the current value of the mask test impedance and prints the result to the computer screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"  !Response headers off
20  OUTPUT 707;":MTEST:IMPEDANCE?"
30  ENTER 707;Impedance
40  PRINT Impedance
50  END
```

INVert

Command

```
:MTESt:INVert {{ON | 1} | {OFF | 0}}
```

The :MTESt:INVert command inverts the mask for testing negative-going pulses. The trigger level and mask offset are also adjusted. Not all masks support negative-going pulse testing, and for these masks, the command is ignored.

Example

This example inverts the mask for testing negative-going pulses.

```
10  OUTPUT 707;"MTEST:INVERT ON"
20  END
```

Query

```
:MTESt:INVert?
```

The :MTESt:INVert? query returns the current inversion setting.

Returned Format

```
[:MTESt:INVert] {1 | 0}<NL>
```

LAMPlitude

Command `:MTESt:LAMPlitude <lower_limit>`

The :MTESt:LAMPlitude command sets the minimum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the lower limit used during mask testing.

`<lower_limit>` A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example sets the minimum pulse amplitude to 2.4 volts.

```
10  OUTPUT 707;"MTEST:LAMPLITUDE 2.4"
20  END
```

Query `:MTESt:LAMPlitude?`

The :MTESt LAMPlitude? query returns the current value of the minimum pulse amplitude.

Returned Format `[:MTESt:LAMPlitude] <lower_limit><NL>`

`<lower_limit>` A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example returns the current lower pulse limit and prints it to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF  !Response headers off
20  OUTPUT 707;"MTEST:LAMPLITUDE?"
30  ENTER 707;ULimit
40  PRINT ULimit
50  END
```

LOAD

Command

```
:MTESt:LOAD "<filename>"
```

The :MTESt:LOAD command loads the specified mask file. The default path for mask files is C:\Documents and Settings\All Users\Documents\Infiniium\MASKS. To use a different path, specify the complete path and file name.

`<filename>` An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used).

Example

This example loads the mask file named "140md_itu_1.msk".

```
10  OUTPUT 707;"MTEST:LOAD""c:\Documents and Settings\All
Users\Documents\Infiniium\masks\140md_itu_1.msk"""
20  END
```

NREGions?

Query `:MTESt:NREGions?`

The :MTESt:NREGions? query returns the number of regions that define the mask.

Returned Format `[:MTESt:NREGions] <regions><NL>`

`<regions>` An integer from 0 to 8.

Example This example returns the number of mask regions.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":MTEST:NREGIONS?"
30  ENTER 707;Regions
40  PRINT Regions
50  END
```

PROBe:IMPedance?

Query `:MTESt:PROBe:IMPedance?`

The :MTESt:PROBe:IMPedance? query returns the impedance setting for the E2621A and E2622A probes for the current mask test channel.

Returned Format `[:MTESt:PROBe:IMPedance] <impedance><NL>`

`<impedance>` An unquoted string: 75, 100, 110, 120, or NONE

Example This example returns the impedance setting for the probe.

```
10  DIM Impedance$[20]
20  OUTPUT 707;":SYSTEM:HEADER OFF"
30  OUTPUT 707;":MTEST:PROBE:IMPEDANCE?"
40  ENTER 707;Impedance$
50  PRINT Impedance$
60  END
```

RUMode

Command

```
:MTESt:RUMode {FORever | TIME, <time> | {WAVeforms,
<number_of_waveforms>}}
```

The :MTESt:RUMode command determines the termination conditions for the mask test. The choices are FORever, TIME, or WAVeforms.

If WAVeforms is selected, a second parameter is required indicating the number of failures that can occur or the number of samples or waveforms that are to be acquired.

`FORever` FORever runs the Mask Test until the test is turned off. This is used when you want a measurement to run continually and not to stop after a fixed number of failures. For example, you may want the Mask Test to run overnight and not be limited by a number of failures.

`TIME` TIME sets the amount of time in minutes that a mask test will run before it terminates.

`<time>` A real number: 0.1 to 1440.0

`WAVeforms` WAVeforms sets the maximum number of waveforms that are required before the mask test terminates.

`<number_of_waveforms>` An integer: 1 to 1,000,000,000.

Example

This example sets the mask test subsystem run until mode to continue testing until 500,000 waveforms have been gathered.

```
10 OUTPUT 707;"MTEST:RUMODE WAVEFORMS,500E3"
20 END
```

Query

```
:MTESt:RUMode?
```

The query returns the currently selected termination condition and value.

Returned Format

```
[:MTESt:RUMode] {FOREver | TIME,<time> | {WAVeforms,
<number_of_waveforms>}}<NL>
```

Example

This example gets the current setting of the mask test run until mode from the oscilloscope and prints it on the computer screen.

```
10 DIM MTEST_Runmode$[50]
20 OUTPUT 707; "MTEST:RUMODE?"
30 ENTER 707;":MTEST_Runmode$
40 PRINT MTEST_Runmode$
50 END
```

RUMode:SOFailure

Command

```
:MTESt:RUMode:SOFailure {{ON | 1} | {OFF | 0}}
```

The :MTESt:RUMode:SOFailure command enables or disables the Stop On Failure run until criteria. When a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Example

This example enables the Stop On Failure run until criteria.

```
10  OUTPUT 707;":MTEST:RUMODE:SOFAILURE ON"
20  END
```

Query

```
:MTESt:SOFailure?
```

The :MTESt:SOFailure? query returns the current state of the Stop on Failure control.

Returned Format

```
[:MTESt:SOFailure] {1 | 0}<NL>
```

SCALe:BIND

Command

```
:MTESt:SCALe:BIND {{ON | 1} | {OFF | 0}}
```

The :MTESt:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control. If the Bind 1 & 0 Levels control is enabled, the 1 Level and the 0 Level controls track each other. Adjusting either the 1 Level or the 0 Level control shifts the position of the mask up or down without changing its size. If the Bind 1 & 0 Levels control is disabled, adjusting either the 1 Level or the 0 Level control changes the vertical height of the mask.

If the Bind -1 & 0 Levels control is enabled, the -1 Level and the 0 Level controls track each other. Adjusting either the -1 Level or the 0 Level control shifts the position of the mask up or down without changing its size. If the Bind -1 & 0 Levels control is disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

Example

This example enables the Bind 1 & 0 Levels control.

```
10 OUTPUT 707;"MTEST:SCALE:BIND ON"
20 END
```

Query

```
:MTESt:SCALe:BIND?
```

The :MTESt:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Returned Format

```
[:MTESt:SCALe:BIND?] {1 | 0}<NL>
```

SCALe:X1

Command

```
:MTESt:SCALe:X1 <x1_value>
```

The :MTESt:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation:

$$X = (X \times \Delta X) + X1$$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

`<x1_value>` A time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Example

This example sets the X1 coordinate at 150 ms.

```
10 OUTPUT 707;":MTEST:SCALE:X1 150E-3"
20 END
```

Query

```
:MTESt:SCALe:X1?
```

The :MTESt:SCALe:X1? query returns the current X1 coordinate setting.

Returned Format

```
[:MTESt:SCALe:X1] <x1_value><NL>
```

Example

This example gets the current setting of the X1 coordinate from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_x1$[50]
20 OUTPUT 707;":MTEST:SCALE:X1?"
30 ENTER 707;Scale_x1$
40 PRINT Scale_x1$
50 END
```

SCALe:XDELta

Command

```
:MTESt:SCALe:XDELta <xdelta_value>
```

The :MTESt:SCALe:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then a change in data rate without corresponding changes in the waveform can easily be handled by changing ΔX.

The X-coordinate of polygon vertices is normalized using this equation:

$$X = (X \times \Delta X) + X1$$

`<xdelta_value>` A time value specifying the distance of the X2 marker with respect to the X1 marker.

Example

Assume that the period of the waveform you wish to test is 1 ms. Then the following example will set ΔX to 1 ms, ensuring that the waveform's period is between the X1 and X2 markers.

```
10 OUTPUT 707;":MTEST:SCALE:XDELTA 1E-6:
20 END
```

Query

```
:MTESt:SCALe:XDELta?
```

The :MTESt:SCALe:XDELta? query returns the current value of ΔX.

Returned Format

```
[:MTESt:SCALe:XDELta] <xdelta_value><NL>
```

Example

This example gets the value of ΔX from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_xdelta$[50]
20 OUTPUT 707;":MTEST:SCALE:XDELTA?"
30 ENTER 707;Scale_xdelta$
40 PRINT Scale_xdelta$
50 END
```

SCALe:Y1

Command `:MTESt:SCALe:Y1 <y_value>`

The :MTESt:SCALe:Y1 command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2 according to the equation:

$$Y = (Y \times (Y2\text{-}Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

`<y1_value>` A voltage value specifying the point at which Y=0.

Example This example sets the Y1 marker to -150 mV.

```
10 OUTPUT 707; ":MTEST:SCALE:Y1 -150E-3"
20 END
```

Query `:MTESt:SCALe:Y1?`

The SCALe:Y1? query returns the current setting of the Y1 marker.

Returned Format `[:MTESt:SCALe:Y1] <y1_value><NL>`

Example This example gets the setting of the Y1 marker from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_y1$[50]
20 OUTPUT 707;":MTEST:SCALE:Y1?"
30 ENTER 707;Scale_y1$
40 PRINT Scale_y1$
50 END
```

SCALe:Y2

Command

```
:MTESt:SCALe:Y2 <y2_value>
```

The :MTESt:SCALe:Y2 command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2 according to the following equation: $Y = (Y \times (Y2\text{-}Y1)) + Y1$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

`<y2_value>` A voltage value specifying the location of the Y2 marker.

Example

This example sets the Y2 marker to 2.5 V.

```
10 OUTPUT 707;":MTEST:SCALE:Y2 2.5"
20 END
```

Query

```
:MTESt:SCALe:Y2?
```

The SCALe:Y2? query returns the current setting of the Y2 marker.

Returned Format

```
[:MTESt:SCALe:Y2] <y2_value><NL>
```

Example

This example gets the setting of the Y2 marker from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_y2$[50]
20 OUTPUT 707;":MTEST:SCALE:Y2?"
30 ENTER 707;Scale_y2$
40 PRINT Scale_y2$
50 END
```

SOURce

Command

```
:MTESt:SOURce {CHANnel<N> | FUNCtion<N> | EQUalized}
```

The :MTESt:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> An integer, 1 - 4.

Example

This example selects channel 1 as the mask test source.

```
10 OUTPUT 707; "MTEST:SOURCE CHANNEL1"
20 END
```

Query

```
:MTESt:SOURce?
```

The :MTESt:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Returned Format

```
[:MTESt:SOURce] {CHANnel<N> | FUNCtion<N> | EQUalized}<NL>
```

Example

This example gets the mask test source setting and prints the result on the computer display.

```
10 DIM Amask_source$[30]
20 OUTPUT 707;"MTEST:SOURCE?"
30 ENTER 707;Amask_source$
40 PRINT Amask_source$
50 END
```

STARt | STOP

Command

```
:MTESt:{STARt | STOP}
```

The :MTESt:{STARt | STOP} command starts or stops the mask test. The :MTESt:STARt command also starts the oscilloscope acquisition system. The :MTESt:STOP command does not stop the acquisition system.

Example

This example starts the mask test and acquisition system.

```
10  OUTPUT 707;"MTEST:START"
20  END
```

STIMe

Command `:MTESt:STIMe <timeout>`

The :MTESt:STIMe command sets the timeout value for the Autoalign feature. If the oscilloscope is unable to align the mask to your waveform within the specified timeout value, it will stop trying to align and will report an alignment failure.

`<timeout>` An integer from 1 to 120 seconds representing the time between triggers (not the time that it takes to finish the alignment.)

Example This example sets the timeout value for the Autoalign feature to 10 seconds.

```
10 OUTPUT 707;"MTEST:STIMe 10"
20 END
```

Query `:MTESt:STIMe?`

The query returns timeout value for the Autoalign feature.

Returned Format `[:MTESt:STIMe] <timeout><NL>`

Example This example gets the timeout setting and prints the result on the computer display.

```
10 OUTPUT 707;"MTEST:STIME?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

TITLe?

Query `:MTESt:TITLe?`

The :MTESt:TITLe? query returns the mask title which is a string of up to 23 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

Returned Format `[:MTESt:TITLe] <mask_title><NL>`

`<mask_title>` A string of up to 23 ASCII characters which is the mask title.

Example This example places the mask title in the string variable and prints the contents to the computer's screen.

```
10  DIM Title$[24]
20  OUTPUT 707;":MTEST:TITLE?"
30  ENTER 707;Title$
40  PRINT Title$
50  END
```

TRIGger:SOURce

Command `:MTESt:TRIGger:SOURce CHANnel<N>`

The :MTESt:TRIGger:SOURce command sets the channel or function to use as the trigger. Mask testing must be enabled before using this command.

<N> An integer, 1 - 4.

Example This example sets the mask trigger source to channel 1.

```
10 OUTPUT 707;"MTEST:TRIGGER:SOURCE CHANNEL1"
20 END
```

Query `:MTESt:TRIGger:SOURce?`

The query returns the currently selected mask test trigger source.

Returned Format `[:MTESt:TRIGger] CHANnel<N><NL>`

Example This example gets the trigger source setting and prints the result on the computer display.

```
10 DIM Amask_source$[30]
20 OUTPUT 707;"MTEST:TRIGGER:SOURCE?"
30 ENTER 707;Amask_source$
40 PRINT Amask_source$
50 END
```

22

Measure Commands

Measure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms. These MEASure commands and queries are implemented in the Infiniium Oscilloscopes.

- AREA
- BWIDth
- CDRRATE
- CGRade:CROSsing
- CGRade:DCDistortion
- CGRade:EHEight
- CGRade:EWIDth
- CGRade:JITTer
- CGRade:QFACtor
- CLEar | SCRatch
- CROSsing
- DELTatime
- DUTYcycle
- FALLtime
- FFT:DFRequency (delta frequency)
- FFT:DMAGnitude (delta magnitude)
- FFT:FREQuency
- FFT:MAGNitude
- FFT:PEAK1
- FFT:PEAK2
- FFT:THReshold
- FREQuency
- HISTogram:HITS
- HISTogram:M1S
- HISTogram:M2S
- HISTogram:M3S
- HISTogram:MAX

- HISTogram:MEAN
- HISTogram:MEDian
- HISTogram:MIN
- HISTogram:PEAK
- HISTogram:PP
- HISTogram:STDDev
- NAME
- NPULses
- NWIDth
- OVERshoot
- PERiod
- PHASe
- PPULses
- PREShoot
- PWIDth
- QUALifier<M>
- RESults?
- RISetime
- SCRatch | CLEar
- SENDvalid
- SETuptime
- SLEWrate
- SOURce
- STATistics
- TEDGe
- TMAX
- TMIN
- TVOLt
- VAMPlitude
- VAVerage
- VBASe
- VLOWer
- VMAX
- VMIDdle
- VMIN
- VPP
- VRMS

- VTIMe
- VTOP
- VUPPer
- WINdow

E2688A High Speed Serial Software commands

The following MEASure commands are available when the E2688A High Speed Serial Software is installed.

- CLOCk
- CLOCk:METHod
- CLOCk:VERTical:OFFset
- CLOCk:VERTical:RANGe
- TIEData
- TIEFilter:STARt
- TIEFilter:STATe
- TIEFilter:STOP
- TIEFilter:TYPE
- Also see the MTESt:FOLDing command in the mask test subsystem.

E2681A EZJIT Jitter Analysis Software commands

The following MEASure commands are available when the E2681A EZJIT Jitter Analysis Software is installed.

- CTCDutycycle
- CTCJitter
- CTCNwidth
- CTCPwidth
- DATarate
- HOLDtime
- JITTer:HISTogram
- JITTer:MEASurement
- JITTer:SPECtrum
- JITTer:STATistics
- JITTer:TRENd
- NCJitter
- NPERiod
- NUI
- SETuptime
- TIEClock2

- TIEData
- UITouijitter
- UNITinterval
- DUTYcycle, FREQuency, PERiod, and PHASe have an additional <direction> parameter.

N5400A and N5401A Jitter Analysis Software commands

The following MEASure commands are available when the N5400A or N5401A Jitter Analysis Software is installed.

- CLOCk
- CLOCk:METHod
- CTCDutycycle
- CTCJitter
- CTCNwidth
- CTCPwidth
- DATarate
- HOLDtime
- JITTer:HISTogram
- JITTer:MEASurement
- JITTer:SPECtrum
- JITTer:STATistics
- JITTer:TRENd
- NCJitter
- :RJDJ:ALL?
- :RJDJ:BER
- :RJDJ:EDGE
- :RJDJ:INTerpolate
- :RJDJ:PLENgth
- :RJDJ:SOURce
- :RJDJ:STATe
- :RJDJ:TJRJDJ?
- :RJDJ:UNITs
- SETuptime
- TIEClock2
- TIEData
- UNITinterval
- DUTYcycle, FREQuency, PERiod, and PHASe have an additional <direction> parameter.

FFT Commands

The :MEASure:FFT commands control the FFT measurements that are accessible through the Measure subsystem.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope.

- For a period or frequency measurement, at least one and a half complete cycles must be displayed.
- For a pulse width measurement, the entire pulse must be displayed.
- For a rise time measurement, the leading (positive-going) edge of the waveform must be displayed.
- For a fall time measurement, the trailing (negative-going) edge of the waveform must be displayed.

In jitter mode with jitter statistics enabled, measurements are made on all data regardless of what is on screen.

User-Defined Thresholds

If you choose to set user-defined thresholds, they must be set before actually sending the measurement command or query.

Measurement Error

If a measurement cannot be made because of a lack of data, because the source waveform is not displayed, the requested measurement is not possible (for example, a period measurement on an FFT waveform), or for some other reason, the following results are returned:

- 9.99999E+37 is returned as the measurement result.
- If SENDvalid is ON, the error code is also returned as well as the questionable value.

Making Measurements

If more than one period, edge, or pulse is displayed, time measurements are made on the first, left-most portion of the displayed waveform.

When any of the defined measurements are requested, the oscilloscope first determines the top (100%) and base (0%) voltages of the waveform. From this information, the oscilloscope determines the other important voltage values (10%, 90%, and 50% voltage values) for making measurements.

The 10% and 90% voltage values are used in the rise time and fall time measurements when standard thresholds are selected. The 50% voltage value is used for measuring frequency, period, pulse width, and duty cycle with standard thresholds selected.

You can also make measurements using user-defined thresholds instead of the standard thresholds.

When the command form of a measurement is used, the oscilloscope is placed in the continuous measurement mode. The measurement result will be displayed on the front panel. There may be a maximum of 5 measurements running continuously. Use the SCRatch command to turn off the measurements.

When the query form of the measurement is used, the measurement is made one time, and the measurement result is returned.

- If the current acquisition is complete, the current acquisition is measured and the result is returned.
- If the current acquisition is incomplete and the oscilloscope is running, acquisitions will continue to occur until the acquisition is complete. The acquisition will then be measured and the result returned.
- If the current acquisition is incomplete and the oscilloscope is stopped, the measurement result will be 9.99999e+37 and the incomplete result state will be returned if SENDvalid is ON.

All measurements are made using the entire display, except for VAVerage and VRMS which allow measurements on a single cycle. Therefore, if you want to make measurements on a particular cycle, display only that cycle on the screen.

Measurements are made on the displayed waveforms specified by the SOURce command. The SOURce command lets you specify two sources. Most measurements are only made on a single source. Some measurements, such as the DELTatime measurement, require two sources.

If the waveform is clipped, the measurement result may be questionable. In this case, the value returned is the most accurate value that can be made using the current scaling. You might be able to obtain a more accurate measurement by adjusting the vertical scale to prevent the waveform from being clipped.

AREA

Command

```
:MEASure:AREA {CYCLe | DISPlay}[,<source>]
```

The :MEASure:AREA command turns on the area measurement. The area measurement measures between the waveform, or a selected cycle of the waveform, and the waveform ground. When measuring Area, it is sometimes useful to use the Subtract Math Operator to remove any dc offset from a waveform you want to measure. Also see Math/FFT Functions for more details.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example turns on the area measurement which measures between the waveform and ground. Only that portion of the waveform which is in the waveform viewing area is measured.

```
10 OUTPUT 707;"MEASURE:AREA DISPLAY"
20 END
```

Query

```
:MEASure:AREA?
```

The :MEASure:AREA? query returns the area measurement.

Returned Format

```
[:MEASure:AREA]<value>[,<result_state>]<NL>
```

Example

This example places the current selection for the area to be measured in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10 DIM Selection$[50]
20 OUTPUT 707;"MEASure:AREA?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

BWIDth

Command

```
:MEASure:BWIDth <source>,<idle_time>
```

The :MEASure:BWIDth command measures the width of bursts in your waveform. The idle time is the minimum time between bursts.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MSPectrum | MTRend | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` is an integer, 1 - 4.

`<idle_time>` Amount of idle time between bursts.

Example

This example measures the width of bursts for the waveform on channel one and sets the idle time to 1 microsecond.

```
10 OUTPUT 707;"MEASURE:BWIDTH CHANNEL1,1E-6"
20 END
```

Query

```
:MEASure:BWIDth? <source>,<idle_time>
```

The :MEASure:BWIDth? query returns the width of the burst being measured.

Returned Format

```
[:MEASure:BWIDth ]<burst_width><NL>
```

Example

This example returns the width of the burst being measured, in the string variable, Burstwidth$, then prints the contents of the variable to the computer's screen.

```
10 DIM Burstwidth$[50]
20 OUTPUT 707;"MEASure:BWIDTH? CHANNEL1,1E-6"
30 ENTER 707;Burstwidth$
40 PRINT Burstwidth$
50 END
```

CDRRATE

Command

```
:MEASure:CDRRATE <source>
```

The :MEASure:CDRRATE command determines the data rate (clock recovery rate) from the clock recovery method being used. It yields one data point per acquisition so trending cannot be performed on this measurement.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MSPectrum | MTRend | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1- 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1- 4, representing the selected function or waveform memory

Example

This example measures the clock recovery rate of channel 1.

```
10 OUTPUT 707;"MEASURE:CDRRATE CHANNEL1"
20 END
```

Example

This example places the current data rate of the channel 1 waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:CDRRATE? CHANNEL1”
30  ENTER 707;Value
40  PRINT Value
50  END
```

CGRade:CROSsing

Command `:MEASure:CGRade:CROSsing`

The :MEASure:CGRade:CROSsing command enables the crossing level percent measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

Example This example measures the crossing level.

```
10 OUTPUT 707;"MEASURE:CGRADE:CROSSING"
20 END
```

Query `:MEASure:CGRade:CROSsing?`

The :MEASure:CGRade:CROSsing? query returns the crossing level percent measurement of the current eye diagram on the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:CROSsing]<value>[,<result_state>]<NL>`

`<value>` The crossing level.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current crossing level in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"        !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:CROSSING?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:DCDistortion

Command

```
:MEASure:CGRade:DCDistortion <format>
```

The :MEASure:CGRade:DCDistortion command enables the duty cycle distortion measurement on the current eye pattern. The parameter specifies the format for reporting the measurement. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

`<format>` {TIME | PERCent}

Example

This example measures the duty cycle distortion.

```
10 OUTPUT 707;"MEASURE:CGRADE:DCDISTORTION TIME"
20 END
```

Query

```
:MEASure:CGRade:DCDistortion? <format>
```

The :MEASure:CGRade:DCDistortion query returns the duty cycle distortion measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:DCDistortion]<value>[,<result_state>]<NL>
```

`<value>` The duty cycle distortion.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example

This example places the current duty cycle distortion in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MEASURE:CGRADE:DCDISTORTION? PERCENT"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:EHEight

Command

```
:MEASure:CGRade:EHEight <format>
```

The :MEASure:CGRade:EHEight command enables the eye height measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

Example

This example enables the eye height measurement.

```
10 OUTPUT 707;"MEASURE:CGRADE:EHEIGHT"
20 END
```

Query

```
:MEASure:CGRade:EHEight?
```

The :MEASure:CGRade:EHEight? query returns the eye height measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:EHEight]<value>[,<result_state>]<NL>
```

`<value>` The eye height.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example

This example places the current eye height in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:EHEIGHT?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:EWIDth

Command

```
:MEASure:CGRade:EWIDth
```

The :MEASure:CGRade:EWIDth command enables the eye width measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

Example

This example measures the eye width.

```
10 OUTPUT 707;"MEASURE:CGRADE:EWIDTH"
20 END
```

Query

```
:MEASure:CGRade:EWIDth?
```

The :MEASure:CGRade:EWIDth? query returns the eye width measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:EWIDth]<value>[,<result_state>]<NL>
```

`<value>` The eye width.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example

This example places the current eye width in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:EWIDTH?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:EWINdow

Command

```
:MEASure:CGRade:EWINdow,<start>,<stop>
[,<start_after>]
```

The :MEASure:CGRade:EWINdow command is used to change the starting point and the stopping point of the window used to make the eye pattern measurements of eye height, eye crossing %, and eye q-factor. In addition, the number of waveform hits can be set to ensure that enough data has been collected to make accurate measurements.

`<start>` An integer from 1 to 100 for horizontal starting point. (Default value is 40%.)

`<stop>` An integer from 1 to 100 for horizontal stopping point. (Default value is 60%.)

`<start_after>` An integer from 1 to 63,488 for number of hits to acquire before making measurements. (Default value is 1.)

Example

This example sets the eye window starting point to 2%, the stopping point to 75% and the start after to 5,000 hits.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:CGRADE:EWINDOW 2,75,5000"
30  END
```

Query

```
:MEASure:CGRade:EWINdow?
```

The :MEASure:CGRade:EWINdow query returns the starting point, the ending point, and the start after setting for the eye pattern measurements.

Returned Format

```
[:MEASure:CGRade:EWIDdow] <start>,<stop>,<start_after> <NL>
```

The following example returns the values for the eye window.

Example

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:CGRADE:EWINDOW?"
30  ENTER 707;Start,Stop,Startafter
40  PRINT Start,Stop,Startafter
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

CGRade:JITTer

Command `:MEASure:CGRade:JITTer <format>`

The :MEASure:CGRade:JITTer measures the jitter at the eye diagram crossing point. The parameter specifies the format, peak-to-peak or RMS, of the returned results. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

`<format>` {PP | RMS}

Example This example measures the jitter.

```
10 OUTPUT 707;"MEASURE:CGRADE:JITTER RMS"
20 END
```

Query `:MEASure:CGRade:JITTer? <format>`

The :MEASure:CGRade:JITTer? query returns the jitter measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:JITTer]<value>[,<result_state>]<NL>`

`<value>` The jitter.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current jitter in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:JITTER? RMS"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:QFACtor

Command

```
:MEASure:CGRade:QFACtor
```

The :MEASure:CGRade:QFACtor command measures the Q factor. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

Example

This example measures the Q factor.

```
10 OUTPUT 707;"MEASURE:CGRADe:QFACTOR"
20 END
```

Query

```
:MEASure:CGRade:QFACtor?
```

The :MEASure:CGRade:QFACtor? query returns the Q factor measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:QFACtor]<value>[,<result_state>]<NL>
```

`<value>` The Q factor.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example

This example places the Q factor in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:QFACTOR"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CLEar

Command

```
:MEASure:{CLEar | SCRatch}
```

The :MEASure:CLEar command clears the measurement results from the screen and disables all previously enabled measurements.

Example

This example clears the current measurement results from the screen.

```
10 OUTPUT 707;":MEASURE:CLEAR"
20 END
```

CLOCk

This command is only available when the E2688A High Speed Serial Software.

Command

```
:MEASure:CLOCk {{{ON|1},CHANnel<N>} | {OFF|0}}
```

The :MEASure:CLOCk command turns the recovered clock display on or off and sets the clock recovery channel source.

<N> An integer, 1 - 4.

Example

This example turns the recovered clock display on for channel 1.

```
10 OUTPUT 707;":MEASURE:CLOCK ON,CHANNEL1"
20 END
```

Query

```
:MEASure:CLOCk?
```

The :MEASure :CLOCk? query returns the state of the recovered clock display.

Returned format

```
[:MEASure:CLOCk] {1 | 0}<NL>
```

Example

This example places the current setting of the recovered clock display in the variable Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:CLOCK?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

CLOCk:METHod

This command is only available when the E2688A High Speed Serial Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:CLOCk:METHod
{FOPLL,<data_rate>,<loop_bandwidth>} |
{EQFOPLL,<data_rate>,<loop_bandwidth>}
{SOPLL,<data_rate>,<loop_bandwidth>,
    <damping_factor>} |
{EQSOPLL,<data_rate>,<loop_bandwidth>,
    <damping_factor>}
{PCIE,{DEEMphasis | TRANsition | BOTH}} |
{FC,{FC1063 | FC2125 | FC425}} |
{EXPFOPLL,<source>,{RISing | FALLing | BOTH},
    <multiplier>,<clock_freq>,<loop_bandwidth>} |
{EXPSOPLL,<source>,{RISing | FALLing | BOTH},
    <multiplier>,<clock_freq>,<loop_bandwidth>,
    <damping_fact>} |
{EXPlicit,<source>,{RISing | FALLing | BOTH}
    [,<multiplier>]} |
{FIXed,{AUTO | {SEMI[,<data_rate>]} | <data_rate>}}

{FLEXR,<baud_rate>} | {FLEXT,<baud_rate>}
```

The :MEASure:CLOCk:METHod command sets the clock recovery method to FOPLL (first order phase-locked loop), SOPLL (second order phase-locked loop), EQFOPLL (equalized first order phase-locked loop), EQSOPLL (equalized second order phase-locked loop), PCIE (PCI Express), FC (Fibre Channel), EXPFOPLL (Explicit First Order PLL), EXPSOPLL (Explicit Second Order PLL), EXPlicit (Explicit Clock), FIXed (Constant Frequency), FLEXR (FlexRay Receiver), or FLEXT (FlexRay Transmitter).

The EQUalized clock recovery methods are only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1 - 4.
FUNCtion<N> and WMEMory<N> are:
An integer, 1 - 4, representing the selected function or waveform memory.

<data_rate> A real number for the base data rate in Hertz.

<damping_factor> A real number for the damping factor of the PLL in bits per second.

<loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

<track_freq> A real number used for the tracking frequency of the PLL.

<damping_fact> A real number used for the damping factor of the PLL.

<baud_rate> A real number used for the baud rate.

Example

This example sets the clock recovery method to phase-locked loop.

```
10 OUTPUT 707;":MEASURE:CLOCK:METHOD FOPLL,2E9,1.19E6"
20 END
```

Query

```
:MEASure:CLOCk:METHod?
```

The :MEASure :CLOCk:METHod? query returns the state of the clock recovery method.

Returned format

```
[:MEASure:CLOCk:METHod]
{FOPLL,<data_rate>,<loop_bandwidth>} |
{EQFOPLL,<data_rate>,<loop_bandwidth>}
{SOPLL,<data_rate>,<loop_bandwidth>,<damping_factor>} |
{EQSOPLL,<data_rate>,<loop_bandwidth>,<damping_factor>}
{PCIE,{DEEMphasis | TRANsition | BOTH}} |
{FC,{FC1063 | FC2125 | FC425}} |
{EXPFOPLL <source>,{RISing | FALLing | BOTH},
<multiplier>,<clock_freq>,<track_freq>} |
{EXPSOPLL <source>,{RISing | FALLing | BOTH},
<multiplier>,<clock_freq>,<track_freq>,<damping_fact>} |
{EXPlict,<source>,{RISing | FALLing | BOTH},<multiplier>} |
{FIXed,{AUTO | {SEMI,<data_rate>} | <data_rate>}}
```

```
{FLEXR,<baud_rate>} | {FLEXT,<baud_rate>}
```

Example

This example places the current setting of the clock recovery method in the variable Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:CLOCK:METHOD?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

CLOCk:VERTical

This command is only available when the E2688A High Speed Serial Software is installed.

Command

```
:MEASure:CLOCk:VERTical {AUTO | MANual}
```

The :MEASure:CLOCk:VERTIcal command sets the recovered clock vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the recovered clock vertical scale mode to automatic.

```
10 OUTPUT 707;":MEASURE:CLOCk:VERTical AUTO"
20 END
```

Query

```
:MEASure:CLOCk:VERTical?
```

The :MEASure:CLOCk:VERTical? query returns the current recovered clock vertical scale mode setting.

Returned format

```
[:MEASure:CLOCk:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the recovered clock vertical scale mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:CLOCK:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

CLOCk:VERTical:OFFSet

This command is only available when the E2688A High Speed Serial Software is installed.

Command

```
:MEASure:CLOCk:VERTical:OFFSet <offset>
```

The :MEASure:CLOCk:VERTial:OFFSet command sets the recovered clock vertical offset.

`<offset>` A real number for the recovered clock vertical offset.

Example

This example sets the clock recovery vertical offset to 1 volt.

```
10  OUTPUT 707;":MEASURE:CLOCK:VERTICAL:OFFSET 1"
20  END
```

Query

```
:MEASure:CLOCk:VERTical:OFFSet?
```

The :MEASure:CLOCk:VERTIcal:OFFSet? query returns the clock recovery vertical offset setting.

Returned format

```
[:MEASure:CLOCk:VERTical:OFFSet] <value><NL>
```

`<value>` The clock recovery vertical offset setting.

Example

This example places the current value of recovered clock vertical offset in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:CLOCK:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

CLOCk:VERTical:RANGe

This command is only available when the E2688A High Speed Serial Software is installed.

Command

```
:MEASure:CLOCk:VERTical:RANGe <range>
```

The :MEASure:CLOCk:VERTial:RANGe command sets the recovered clock vertical range.

`<range>` A real number for the full-scale recovered clock vertical range.

Example

This example sets the recovered clock vertical range to 16 volts (2 volts times 8 divisions.)

```
10  OUTPUT 707;":MEASURE:CLOCK:VERTICAL:RANGE 16"
20  END
```

Query

```
:MEASure:CLOCk:VERTical:RANGe?
```

The :MEASure:CLOCk:VERTical:RANGe? query returns the recovered clock vertical range setting.

Returned Format

```
[:MEASure:CLOCk:VERTical:RANGe] <value><NL>
```

`<value>` The recovered clock vertical range setting.

Example

This example places the current value of recovered clock vertical range in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:CLOCK:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

CROSsing

Command `:MEASure:CROSsing <source1>, <source2>, <hysteresis>`

The :MEASure:CROSsing command measures the voltage where two signals cross (uses edges closest to the center of the screen)

`<source1>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<source2>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<hysteresis>` a real number

Example This example measures the voltage where channel 1 and 2 cross (hysteresis set to 50 mV).

```
10 OUTPUT 707;"MEASURE:CROSsing CHAN1, CHAN2, 50e-3"
20 END
```

Query `:MEASure:CROSsing?`

The :MEASure:CROSsing? query returns the crossing measurement.

Returned Format `[:MEASure:CROSsing]<value><NL>`

CTCDutycycle

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:CTCDutycycle <source>,<direction>
```

The :MEASure:CYCDutycycle command measures the cycle-to-cycle duty cycle jitter (%) of the waveform.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing}

Specifies direction of waveform edge to make measurement.

Example

This example measures the cycle-to-cycle duty cycle on the rising edge of channel 1.

```
10 OUTPUT 707;"MEASURE:CTCDUTYCYCLE CHANNEL1,RISING"
20 END
```

Query `:MEASure:CTCDutycycle? <source>,<direction>`

The :MEASure:CTCDutycycle? query returns the cycle-to-cycle duty cycle jitter (%) measurement.

Returned Format `[:MEASure:CTCDutycycle <value>[,<result_state>]<NL>`

`<value>` The cycle-to-cycle duty cycle jitter (%) of the waveform.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle duty cycle of channel 1 in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CTCDUTYCYCLE CHANNEL1,RISING"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CTCJitter

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:CTCJitter <source>,<direction>
```

The :MEASure:CYCJitter command measures the cycle-to-cycle jitter of the waveform.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<direction> {RISing | FALLing}

Specifies direction of waveform edge to make measurement.

Example

This example measures the cycle-to-cycle jitter on the rising edge of channel 1.

```
10 OUTPUT 707;"MEASURE:CTCJITTER CHANNEL1,RISING"
20 END
```

Query `:MEASure:CTCJitter? <source>,<direction>`

The :MEASure:CTCJitter? query returns the cycle-to-cycle jitter time measurement.

Returned Format `[:MEASure:CTCJitter <value>[,<result_state>]<NL>`

`<value>` The cycle-to-cycle jitter time of the waveform.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle jitter of channel 1 in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CTCJITTER CHANNEL1,RISING"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CTCNwidth

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command `:MEASure:CTCNwidth [<source>]`

The :MEASure:CTCNwidth command measures the cycle-to-cycle -width jitter of the waveform.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example measures the cycle-to-cycle -width of channel 1.

```
10 OUTPUT 707;"MEASURE:CTCNWIDTH CHANNEL1"
20 END
```

Query `:MEASure:CTCNwidth? [<source>]`

The :MEASure:CTCNwidth? query returns the cycle-to-cycle -width jitter measurement.

Returned Format `[:MEASure:CTCNwidth <value>[,<result_state>]<NL>`

`<value>` The cycle-to-cycle - width jitter of the waveform.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example

This example places the cycle-to-cycle - width of channel 1 in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CTCNWIDTH CHANNEL1"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CTCPwidth

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command `:MEASure:CTCPwidth [<source>]`

The :MEASure:CYCPwidth command measures the cycle-to-cycle + width jitter of the waveform.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example measures the cycle-to-cycle - width of channel 1.

```
10 OUTPUT 707;"MEASURE:CTCPWIDTH CHANNEL1"
20 END
```

Query `:MEASure:CTCPwidth? [<source>]`

The :MEASure:CTCPwidth? query returns the cycle-to-cycle + width jitter measurement.

Returned Format `[:MEASure:CTCPwidth <value>[,<result_state>]<NL>`

`<value>` The cycle-to-cycle + width jitter of the waveform.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example

This example places the cycle-to-cycle + width of channel 1 in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CTCPWIDTH CHANNEL1"
30 ENTER 707;Value
40 PRINT Value
50 END
```

DATarate

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:DATarate <source>[,{AUTO |
(SEMI,<data_rate>)}]
```

The :MEASure:DATarate command measures the data rate in bits per second for the selected source. Use the :MEASure:UNITinterval command/query to measure the unit interval of the source

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1- 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1- 4, representing the selected function or waveform memory.

`<data_rate>` A real number specifying the data rate.

Example

This example measures the data rate of channel 1.

```
10 OUTPUT 707;"MEASURE:DATARATE CHANNEL1"
20 END
```

Query

```
:MEASure:DATarate? <source>[,{Auto |
(SEMI,<data_rate>)}]
```

The :MEASure:DATarate? query returns the measured data rate.

Returned Format

```
[:MEASure:DATarate] <value>[,<result_state>]<NL>
```

`<value>` Data rate frequency in bits per second for the selected source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the current data rate of the channel 1 waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:DATARATE? CHANNEL1”
30  ENTER 707;Value
40  PRINT Value
50  END
```

DELTatime

Command `:MEASure:DELTatime [<source>[,<source>]]`

The :MEASure:DELTatime command measures the delta time between two edges. If one source is specified, the delta time from the leading edge of the specified source to the trailing edge of the specified source is measured. If two sources are specified, the delta time from the leading edge on the first source to the trailing edge on the second source is measured.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DELTatime command. The rest of the parameters for this command are specified with the :MEASure:DEFine command.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example measures the delta time between channel 1 and channel 2.

```
10  OUTPUT 707;":MEASURE:DELTATIME CHANNEL1,CHANNEL2"
20  END
```

Query `:MEASure:DELTatime? [<source>[,<source>]]`

The :MEASure:DELTatime? query returns the measured delta time value.

Returned Format `[:MEASure:DELTatime] <value>[,<result_state>]<NL>`

`<value>` Delta time from the first specified edge on one source to the next specified edge on another source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of delta time in the numeric variable, Value, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:DELTATIME?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

DELTatime:DEFine

The MEASuure:DELTatime:DEFine command sets the type of direction, the number of the edge, and the edge position for the delta time measurement.

Command

```
:MEASure:DELTatime:DEFine <start_edge_direction>,
<start_edge_number>,<start_edge_position>,
<stop_edge_direction>,<stop_edge_number>,
<stop_edge_position>
```

`<start_edge_direction>` {RISing | FALLing | EITHer} for start directions.

`<start_edge_number>` An integer from 1 to 65534 for start edge numbers.

`<start_edge_position>` {UPPer | MIDDle | LOWer} for start edge positions.

`<stop_edge_direction>` {RISing | FALLing | EITHer} for stop directions.

`<stop_edge_number>` An integer from 1 to 65534 for stop edge numbers.

`<stop_edge_position>` {UPPer | MIDDle | LOWer} for stop edge positions.

Example

This example sets the delta time starting edge to a rising edge on the 5th edge at the middle position and the stoping edge to falling on the 50th edge at the lower position.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:DELTATIME:DEFINE
RISING,5,MIDDLE,FALLING,50,LOWER"
30  END
```

Query

```
:MEASure:DELTatime:DEFine?
```

The :MEASure:DELTatime:DEFine? query returns the measured delta time value.

Returned Format

```
[:MEASure:DELTatime:DEFine] <start_edge_direction>,
<start_edge_number>,<start_edge_position>,
<stop_edge_direction>,<stop_edge_number>,
<stop_edge_position><NL>
```

Example

This example places the current value of delta time in the numeric variable, Value, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
5  DIM Startd$[50],Startp$[50],Stopd$[50],Stopp$[50] !Dimension
variables
10   OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20   OUTPUT 707;":MEASURE:DELTATIME:DEFINE?"
30   ENTER 707;Startd$,Start,Startp$,Stopd$,Stop,Stopp$
40   PRINT Startp$,Start,Startp$,Stopd$,Stop,Stopp$
50   END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

DUTYcycle

Command `:MEASure:DUTYcycle [<source>],<direction>`

The <direction> parameter is only available when the E2681A Jitter Analysis Software or N5400A/N5t401A Software is installed. When <direction> is specified, the <source> parameter is required.

The :MEASure:DUTYcycle command measures the ratio (%) of the positive pulse width to the period. Sources are specified with the :MEASure:SOURce command or with the optional <source> parameter following the :MEASure:DUTYcycle command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing}

Specifies direction of edge to start measurement.

Example This example measures the duty cycle of the channel 1 waveform.

```
10  OUTPUT 707;":MEASURE:DUTYCYCLE CHANNEL1"
20  END
```

Query `:MEASure:DUTYcycle? [<source>],<direction>`

The :MEASure:DUTYcycle? query returns the measured duty cycle (%) of the specified source.

Returned Format `[:MEASure:DUTYcycle] <value>[,<result_state>]<NL>`

`<value>` The ratio (%) of the positive pulse width to the period.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current duty cycle of the channel 1 waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:DUTYCYCLE? CHANNEL1”
30  ENTER 707;Value
40  PRINT Value
50  END
```

FALLtime

Command `:MEASure:FALLtime [<source>]`

The :MEASure:FALLtime command measures the time at the upper threshold of the falling edge, measures the time at the lower threshold of the falling edge, then calculates the fall time. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FALLtime command.

The first displayed falling edge is used for the fall-time measurement. To make this measurement requires 4 or more sample points on the falling edge of the waveform.

Fall time = time at lower threshold point – time at upper threshold point.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example measures the fall time of the channel 1 waveform.

```
10  OUTPUT 707;":MEASURE:FALLTIME CHANNEL1"
20  END
```

Query `:MEASure:FALLtime? [<source>]`

The :MEASure:FALLtime? query returns the fall time of the specified source.

Returned Format `[:MEASure:FALLtime] <value>[,<result_state>]<NL>`

`<value>` Time at lower threshold - time at upper threshold.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value for fall time in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:FALLTIME? CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

FFT:DFRequency

Command `:MEASure:FFT:DFRequency [<source>]`

The :MEASure:FFT:DFRequency command enables the delta frequency measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT:DFR command.

The source must be a function that is set to FFTMagnitude, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query `:MEASure:FFT:DFRequency? [<source>]`

The :MEASure:FFT:DFRequency? query returns the FFT delta frequency of the specified peaks.

Returned Format

```
[:MEASure:FFT:DFRequency]
<delta_frequency>[,<result_state>]<NL>
```

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Related Commands :MEASure:FFT:PEAK1, :MEASure:FFT:PEAK2, :MEASure:FFT:THReshold

Example This example measures the frequency difference between the peaks specified by the :meas:fft:peak1 and :meas:fft:peak2 for channel 4.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":func4:fftm chan4"!Perform FFT on channel 4
30  OUTPUT 707;":func4:disp on"!Display the FFT
40  OUTPUT 707;":meas:FFT:thr-47"!Set peak threshold at-47 dBm
50  OUTPUT 707;":meas:FFT:Peak1 2"!Meas diff between peak 2 and 3
60  OUTPUT 707;":meas:FFT:Peak2 3"
```

```
70  OUTPUT 707;":meas:FFT:dfr func4"!Perform dfrequency meas
80  OUTPUT 707;":meas:FFT:dfr? func4"!Query oscilloscope for
measurement
90  ENTER 707;Frequency
100 PRINT Frequency
110 END
```

FFT:DMAGnitude

Command `:MEASure:FFT:DMAGnitude [<source>]`

The :MEASure:FFT:DMAGnitude command enables the delta magnitude measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query `:MEASure:FFT:DMAGnitude? [<source>]`

The :MEASure:FFT:DMAGnitude? query returns the delta magnitude of the specified peaks.

Returned Format

```
[:MEASure:FFT:DMAGnitude]
<delta_magnitude>[,<result_state>]<NL>
```

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Related Commands :MEASure:FFT:PEAK1, :MEASure:FFT:PEAK2, :MEASure:FFT:THReshold

Example This example measures the magnitude difference between the peaks specified by the :meas:fft:peak1 and :meas:fft:peak2 for channel 4.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":func4:fftm chan4"!Perform FFT on channel 4
30  OUTPUT 707;":func4:disp on"!Display the FFT
40  OUTPUT 707;":meas:FFT:thr-47"!Set peak threshold at-47 dBm
50  OUTPUT 707;":meas:FFT:Peak1 2"!Meas diff between peak 2 and 3
60  OUTPUT 707;":meas:FFT:Peak2 3"
```

```
70  OUTPUT 707;":meas:FFT:dmag func4"!Perform dfrequency meas
80  OUTPUT 707;":meas:FFT:dmag? func4"!Query oscilloscope for
measurement
90  ENTER 707;Magnitude
100 PRINT Magnitude
110 END
```

FFT:FREQuency

Command `:MEASure:FFT:FREQuency [<source>]`

The :MEASure:FFT:FREQuency command enables the frequency measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query `:MEASure:FFT:FREQuency? [<source>]`

The :MEASure:FFT:FREQuency? query returns the frequency measurement.

Returned Format `[:MEASure:FFT:FREQuency] <frequency>[,<result_state>]<NL>`

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example measures the frequency the peak specified by the :meas:fft:peak1 for channel 4.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":func4:fftm chan4"!Perform FFT on channel 4
30  OUTPUT 707;":func4:disp on"!Display the FFT
40  OUTPUT 707;":meas:FFT:thr-47"!Set peak threshold at-47 dBm
50  OUTPUT 707;":meas:FFT:Peak1 2"!Meas amplitude of peak 2
60  OUTPUT 707;":meas:FFT:freq func4"!Perform frequency meas
70  OUTPUT 707;":meas:FFT:freq? func4"!Query oscilloscope for
measurement
80  ENTER 707;Frequency
90 PRINT Frequency
```

```
100 END
```

FFT:MAGNitude

Command `:MEASure:FFT:MAGNitude [<source>]`

The :MEASure:FFT:MAGNitude command measures the magnitude of the FFT. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query `:MEASure:FFT:MAGNitude?`

The :MEASure:FFT:MAGNitude? query returns the magnitude value of the FFT.

Returned Format `[:MEASure:FFT:FMAGNitude] <magnitude>[,<result_state>]<NL>`

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example measures the magnitude of the peak specified by the :meas:fft:peak for channel 4.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":func4:fftm chan4"!Perform FFT on channel 4
30  OUTPUT 707;":func4:disp on"!Display the FFT
40  OUTPUT 707;":meas:FFT:thr -47"!Set peak threshold at -47 dBm
50  OUTPUT 707;":meas:FFT:Peak1 2"!Meas magnitude of peak 2
60  OUTPUT 707;":meas:FFT:magn func4"!Perform dfrequency meas
70  OUTPUT 707;":meas:FFT:magn? func4"!Query oscilloscope for
measurement
80  ENTER 707;Magnitude
90 PRINT Magnitude
```

```
100 END
```

FFT:PEAK1

Command `:MEASure:FFT:PEAK1 <1st_peak_number>`

The :MEASure:FFT:PEAK1command sets the peak number of the first peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<N> or WMEMory<N>.

`<1st_peak_number>` An integer, 1 to 100 specifying the number of the first peak.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query `:MEASure:FFT:PEAK1?`

The :MEASure:FFT:PEAK1? query returns the peak number currently set as the first peak.

Returned Format `[:MEASure:FFT:PEAK1] <1st_peak_number><NL>`

See Also :MEASure:FFT:THReshold

Also see the example for :MEASure:FFT:DFRequency in this chapter.

FFT:PEAK2

Command

```
:MEASure:FFT:PEAK2 <2nd_peak_number>
```

The :MEASure:FFT:PEAK2 command sets the peak number of the second peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<N> or WMEMory<N>.

`<2nd_peak_number>` An integer, 1 to 100 specifying the number of the second peak.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query

```
:MEASure:FFT:PEAK2?
```

The :MEASure:FFT:PEAK2? query returns the peak number currently set as the second peak.

Returned Format

```
[:MEASure:FFT:PEAK1] <2nd_peak_number><NL>
```

See Also

:MEASure:FFT:THReshold

Also see the example for :MEASure:FFT:DFRequency in this chapter.

FFT:THReshold

Command `:MEASure:FFT:THReshold <threshold_value>`

The :MEASure:FFT:THReshold command sets the peak search threshold value in dB. The dB after the threshold value is optional.

`<threshold_value>` A real number specifying the threshold for peaks.

Query `:MEASure:FFT:THReshold?`

The :MEASure:FFT:THReshold? query returns the peak search threshold value.

Returned Format `[:MEASure:FFT:THReshold] <threshold_value><NL>`

These :MEASure commands also operate on FFT functions:

Measure Command	Measurement Performed
:TMAX	**The frequency of the maximum value in the spectrum.**
:TMIN	**The frequency of the minimum value in the spectrum.**
:VMAX	**The maximum value in the spectrum.**
:VMIN	**The minimum value in the spectrum.**
:VPP	**The range of values in the spectrum.**
:VTIM	**The value at a specified frequency.**

See Also Also see the example for :MEASure:FFT:DFRequency in this chapter.

FREQuency

Command `:MEASure:FREQuency [<source>[,<direction>]]`

The <direction> parameter is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed. When <direction> is specified, the <source> parameter is required.

The :MEASure:FREQuency command measures the frequency of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels if standard thresholds are selected). The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FREQuency command.

The algorithm is:

If the first edge on the screen is rising,

then

frequency = 1/(time at second rising edge - time at first rising edge)

else

frequency = 1/(time at second falling edge - time at first falling edge).

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing}

Specifies direction of edge for measurement.

Example This example measures the frequency of the channel 1 waveform.

```
10  OUTPUT 707;":MEASURE:FREQUENCY CHANNEL1"
20  END
```

Query `:MEASure:FREQuency? [<source>[,<direction>]]`

The :MEASure:FREQuency? query returns the measured frequency.

Returned Format `[:MEASure:FREQuency] <value>[,<result_state>]<NL>`

`<value>` The frequency value in Hertz of the first complete cycle on the screen using the mid-threshold levels of the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current frequency of the waveform in the numeric variable, Freq, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:FREQUENCY? CHANNEL1"
30  ENTER 707;Freq
40  PRINT Freq
50  END
```

HISTogram:HITS

Command `:MEASure:HISTogram:HITS [<source>]`

The :MEASure:HISTogram:HITS command measures the number of hits within the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the HITS command. The HISTogram:HITS measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Example This example measures the number of hits within the histogram stored in WMEMory1.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:HITS WMEMORY1"
20 END
```

Query `:MEASure:HISTogram:HITS? [<source>]`

The :MEASure:HISTogram:HITS? query returns the number of hits within the histogram.

Returned Format `[:MEASure:HISTogram:HITS]<value>[,<result_state>]<NL>`

`<value>` The number of hits in the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the number of hits within the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:HITS? WMEMORY1"
30 ENTER 707;Histhits
40 PRINT Histhits
50 END
```

HISTogram:M1S

Command `:MEASure:HISTogram:M1S [<source>]`

The :MEASure:HISTogram:M1S command enables the percentage of points measurement that are within one standard deviation of the mean of the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the M1S command. The HISTogram:M1S measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Example This example measures the percentage of points that are within one standard deviation of the mean of the histogram of the data stored in waveform memory 3.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:M1S WMEMORY3"
20 END
```

Query `:MEASure:HISTogram:M1S? [<source>]`

The :MEASure:HISTogram:M1S? query returns the measurement of the percentage of points within one standard deviation of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:M1S]<value>[,<result_state>]<NL>`

`<value>` The percentage of points within one standard deviation of the mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within one standard deviation of the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M1S? WMEMORY1"
30 ENTER 707;Histm1s
40 PRINT Histm1s
50 END
```

HISTogram:M2S

Command `:MEASure:HISTogram:M2S [<source>]`

The :MEASure:HISTogram:M2S command enables the percentage of points measurement that are within two standard deviations of the mean of the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the M2S command. The HISTogram:M2S measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Example This example measures the percentage of points that are within two standard deviations of the mean of the histogram whose source is specified using the MEASure:SOURce command.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:M2S WMEMORY1"
20 END
```

Query `:MEASure:HISTogram:M2S? [<source>]`

The :MEASure:HISTogram:M2S? query returns the measurement of the percentage of points within two standard deviations of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:M2S]<value>[,<result_state>]<NL>`

`<value>` The percentage of points within two standard deviations of the mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within two standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M2S? WMEMORY1"
30 ENTER 707;Histm2s
40 PRINT Histm2s
50 END
```

HISTogram:M3S

Command `:MEASure:HISTogram:M3S [<source>]`

The :MEASure:HISTogram:M3S command enables the percentage of points measurement that are within three standard deviations of the mean of the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the M3S command. The HISTogram:M3S measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Example This example measures the percentage of points that are within three standard deviations of the mean of the histogram.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:M3S HISTOGRAM"
20 END
```

Query `:MEASure:HISTogram:M3S? [<source>]`

The :MEASure:HISTogram:M3S? query returns the measurement of the percentage of points within three standard deviations of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:M3S]<value>[,<result_state>]<NL>`

`<value>` The percentage of points within three standard deviations of the mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within three standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M3S? WMEMORY1"
30 ENTER 707;Histm3s
40 PRINT Histm3s
50 END
```

HISTogram:MAX?

Query `:MEASure:HISTogram:MAX? [<source>]`

The :MEASure:HISTogram:MAX? query returns the measurement of the maximum value of the histogram.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:MAX]<value>[,<result_state>]<NL>`

`<value>` The maximum value of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the maximum value of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MAX?"
30 ENTER 707;Histmax
40 PRINT Histmax
50 END
```

HISTogram:MEAN?

Query `:MEASure:HISTogram:MEAN? [<source>]`

The :MEASure:HISTogram:MEAN? query returns the measurement of the mean of the histogram.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:MEAN]<value>[,<result_state>]<NL>`

`<value>` The mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MEAN? WMEMORY1"
30 ENTER 707;Histmean
40 PRINT Histmean
50 END
```

HISTogram:MEDian?

Query `:MEASure:HISTogram:MEDian? [<source>]`

The :MEASure:HISTogram:MEDian? query returns the measurement of the median of the histogram.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:MEDian]<value>[,<result_state>]<NL>`

`<value>` The median of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the median of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MEDIAN? WMEMORY1"
30 ENTER 707;Histmed
40 PRINT Histmed
50 END
```

HISTogram:MIN?

Query `:MEASure:HISTogram:MIN? [<source>]`

The :MEASure:HISTogram:MIN? query returns the measurement of the maximum value of the histogram.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:MIN]<value>[,<result_state>]<NL>`

`<value>` The minimum value of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the minimum value of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MIN?"
30 ENTER 707;Histmin
40 PRINT Histmin
50 END
```

HISTogram:PEAK?

Query `:MEASure:HISTogram:PEAK? [<source>]`

The :MEASure:HISTogram:PEAK? query returns the number of hits in the greatest peak of the histogram measurement.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:PEAK]<value>[,<result_state>]<NL>`

`<value>` The number of hits in the histogram peak.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the number of hits in the greatest peak of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:PEAK? WMEMORY1"
30 ENTER 707;Histpeak
40 PRINT Histpeak
50 END
```

HISTogram:PP?

Query `:MEASure:HISTogram:PP? [<source>]`

The :MEASure:HISTogram:PP? query returns the measurement of the width of the histogram.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:PP]<value>[,<result_state>]<NL>`

`<value>` The width of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the width of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:PP? WMEMORY1"
30 ENTER 707;Histpp
40 PRINT Histpp
50 END
```

HISTogram:STDDev?

Query `:MEASure:HISTogram:STDDev? [<source>]`

The :MEASure:HISTogram:STDDev? query returns the measurement of standard deviation of the histogram.

`<source>` {WMEMory<number> | HISTogram }

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

Returned Format `[:MEASure:HISTogram:STDDev]<value>[,<result_state>]<NL>`

`<value>` The standard deviation of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the standard deviation of the histogram whose source is specified using the MEASure:SOURce command and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:STDDEV? WMEMORY1"
30 ENTER 707;Histsttd
40 PRINT Histsttd
50 END
```

HOLDtime

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command `:MEASure:HOLDtime [<data_source>,<data_source_dir>, <clock_source>,<clock_ source_dir>]`

The :MEASure:HOLDtime command measures the hold time between the specified clock and data sources.

`<data_source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<clock_source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<data_source_dir>` {RISing | FALLing | BOTH}

Selects the direction of the data source edge.

`<clock_source_dir>` {RISing | FALLing}

Selects the direction of the clock source edge.

Example

This example measures the hold time from the rising edge of channel 1 to the rising edge of channel 2.

```
10  OUTPUT 707;":MEASURE:HOLDTIME CHAN1,RIS,CHAN2,RIS"
20  END
```

Query

```
:MEASure:HOLDtime?
[<data_source>,<data_source_dir>,<clock_source>,
<clock_ source_dir>]
```

The :MEASure:HOLDtime? query returns the measured hold time between the specified clock and data source.

Returned Format

```
{:MEASure:SETuptime] <value><NL>
```

<value> Hold time in seconds.

Example

This example places the current value of hold time in the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:HOLDTIME? CHAN1,RIS,CHAN2,RIS"
30  ENTER 707;Time
40  PRINT Time
50  END
```

See Also

Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

JITTer:HISTogram

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:HISTogram {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:HISTogram command turns the measurement histogram display on or off when a jitter measurement is displayed.

Example

This example turns the jitter measurement histogram display on.

```
10 OUTPUT 707;"MEASURE:JITTER:HISTOGRAM ON"
20 END
```

Query

```
:MEASure:JITTer:HISTogram?
```

The :MEASure :JITTer:HISTogram? query returns the state of measurement histogram display.

Returned format

```
[:MEASure:JITTer:HISTogram] {1 | 0}
```

Example

This example places the current setting of the jitter spectrum mode in the variable Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:HISTOGRAM?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

JITTer:MEASurement

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:MEASurement {MEASurement<N>}
```

The :MEASure :JITTer:MEASurement command selects which measurement displayed on the oscilloscope you are performing the jitter analysis on. MEASurement1 is the left-most measurement on the display.

<N> {1 | 2 | 3 | 4 | 5}

Example

This example assigns measurement 2 to the jitter measurement analysis.

```
10 OUTPUT 707;":MEASURE:JITTER:MEASUREMENT MEASUREMENT2"
20 END
```

Query

```
:MEASure:JITTer:MEASurement?
```

The :MEASure :JITTer:MEASurement? query returns the measurement number you are performing the jitter analysis on. If no measurements are being displayed on the oscilloscope, the query will return a null string.

Returned format

```
[:MEASure:JITTer:MEASurement MEASurement<N>]
```

Example

This example places the current measurement number that you are performing jitter analysis on in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:MEASUREMENT?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:SPECtrum

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:SPECtrum command turns the jitter spectrum display on or off when a jitter measurement is displayed.

Example

This example turns the jitter measurement spectrum display on.

```
10 OUTPUT 707;":MEASURE:JITTER:SPECTRUM ON"
20 END
```

Query

```
:MEASure:JITTer:SPECtrum?
```

The :MEASure :JITTer:SPECtrum? query returns the state of jitter spectrum display.

Returned format

```
[:MEASure:JITTer:SPECtrum] {1 | 0}
```

Example

This example places the current setting of the jitter spectrum mode in the variable Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

JITTer:SPECtrum:HORizontal

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:HORizontal {AUTO | MANual}
```

The :MEASure:JITTer:SPECtrum:HORizontal command sets the jitter spectrum horizontal mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the horizontal scaling and center frequency. In manual mode, you can set your own horizontal scaling and center frequency values.

Example

This example sets the jitter spectrum horizontal mode to automatic.

```
10 OUTPUT 707;":MEASURE:JITTER:SPECTRUM:HORIZONTAL AUTO"
20 END
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal?
```

The :MEASure:JITTer:SPECtrum:HORizontal? query returns the current jitter spectrum horizontal mode setting.

Returned format

```
[:MEASure:JITTer:SPECtrum:HORizontal] {AUTO | MANual}
```

Example

This example places the current setting of the jitter trend horizontal mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:HORIZONTAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:SPECtrum:HORizontal:POSition

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:HORizontal:POSition
<position>
```

The :MEASure:JITTer:SPECtrum:HORizontal:POSition command sets the jitter spectrum horizontal center frequency position.

`<position>` A real number for the center frequency position in Hertz.

Example

This example sets the jitter spectrum horizontal center frequency position to 250 kHz.

```
10 OUTPUT 707;":MEASURE:JITTER:SPECTRUM:HORIZONTAL:POSITION
250E3"
20 END
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal:POSition?
```

The :MEASure:JITTer:SPECtrum:HORizontal:POSition? query returns the current jitter spectrum horizontal center frequency position setting.

Returned format

```
[:MEASure:JITTer:SPECtrum:HORizontal:POSition] <value><NL>
```

`<value>` The jitter spectrum horizontal center frequency setting.

Example

This example places the current setting of the jitter trend horizontal center frequency position in the variable Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:HORIZONTAL:POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

JITTer:SPECtrum:HORizontal:RANGe

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:HORizontal:RANGe <range>
```

The :MEASure:JITTer:SPECtrum:HORizontal:RANGe command sets the jitter spectrum horizontal range.

`<range>` A real number for the horizontal frequency range in Hertz.

Example

This example sets the jitter spectrum horizontal range to 10 GHz (1 GHz/div).

```
10 OUTPUT 707;":MEASURE:JITTER:SPECTRUM:HORIZONTAL:RANGE 10E9"
20 END
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal:RANGe?
```

The :MEASure:JITTer:SPECtrum:HORizontal:RANGe? query returns the current jitter spectrum horizontal range setting.

Returned format

```
[:MEASure:JITTer:SPECtrum:HORizontal:RANGe] <value><NL>
```

`<value>` The jitter spectrum horizontal range setting.

Example

This example places the current setting of the jitter trend horizontal range in the variable Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:HORIZONTAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

JITTer:SPECtrum:VERTical

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:VERTical {AUTO | MANual}
```

The :MEASure:JITTer:SPECtrum:VERTical command sets the jitter spectrum vertical mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own vertical scaling and offset values.

Example

This example sets the jitter spectrum vertical mode to automatic.

```
10 OUTPUT 707;":MEASURE:JITTER:SPECTRUM:VERTICAL AUTO"
20 END
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical?
```

The :MEASure:JITTer:SPECtrum:VERTical? query returns the current jitter spectrum vertical mode setting.

Returned format

```
[:MEASure:JITTer:SPECtrum:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the jitter spectrum vertical mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:SPECtrum:VERTical:OFFSet

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:VERTical:OFFSet <offset>
```

The :MEASure:JITTer:SPECtrum:VERTial:OFFSet command sets the jitter spectrum vertical offset.

`<offset>` A real number for the vertical offset of the jitter measurement spectrum.

Example

This example sets the jitter spectrum vertical offset to 2 ns.

```
10  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:VERTICAL:OFFSET 10E-9"
20  END
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:OFFSet?
```

The :MEASure:JITTer:SPECtrum:VERTIcal:OFFSet? query returns the jitter spectrum vertical offset time.

Returned format

```
[:MEASure:JITTer:SPECtrum:VERTical:OFFSet] <value>
[,<result_state>]<NL>
```

`<value>` The jitter vertical spectrum offset time setting.

Example

This example places the current value of jitter spectrum vertical offset in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

JITTer:SPECtrum:VERTical:RANGe

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:VERTical:RANGe <range>
```

The :MEASure:JITTer:SPECtrum:VERTial:RANGe command sets the jitter spectrum vertical range.

`<range>` A real number for the full-scale vertical range for the jitter measurement spectrum.

Example

This example sets the jitter spectrum vertical range to 4 ns (500 ps/div X 8 div).

```
10  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:VERTICAL:RANGE 4E-9"
20  END
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:RANGe?
```

The :MEASure:JITTer:SPECtrum:VERTIcal:RANGe? query returns the jitter spectrum range time setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:RANGe] <value>
[,<result_state>]<NL>
```

`<value>` The jitter spectrum vertical range setting.

Example

This example places the current value of jitter spectrum vertical range in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

JITTer:SPECtrum:WINDow

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:SPECtrum:WINDow {RECTangular | HANNing | FLATtop}
```

The :MEASure:JITTer:SPECtrum:WINDow command sets the jitter spectrum window mode to rectangular, Hanning, or flattop.

Example

This example sets the jitter spectrum window mode to Hanning.

```
10 OUTPUT 707;":MEASURE:JITTER:SPECTRUM:WINDOW HANNING"
20 END
```

Query

```
:MEASure:JITTer:SPECtrum:WINDow?
```

The :MEASure:JITTer:SPECtrum:WINDow? query returns the current jitter spectrum window mode setting.

Returned format

```
[:MEASure:JITTer:SPECtrum:WINDow] {RECTangular | HANNing | FLATtop}<NL>
```

Example

This example places the current setting of the jitter spectrum window mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:SPECTRUM:WINDOW?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:STATistics

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:STATistics {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:STATistics command enables or disables jitter mode and allows you to view: measurement histogram (:MEASure:JITTer:HISTogram), measurement trend (:MEASure:JITTer:TRENd), and jitter spectrum (:MEASure:JITTer:SPsECtrum) if they are enabled. It also turns on the ability to measure all edges in the waveform; not just the first edge on screen.

Example

This example turns the jitter measurement statistics on.

```
10 OUTPUT 707;":JITTer:STATISTICS ON"
20 END
```

Query

```
:MEASure:JITTer:STATistics?
```

The :MEASure :JITTer:STATistics? query returns the state of jitter statistics.

Returned format

```
[:MEASure:JITTer:STATistics] {1 | 0}
```

Example

This example places the current setting of the jitter statistics mode in the variable Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:STATISTICS?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

JITTer:TRENd

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:TRENd {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:TRENd command turns the jitter measurement trend display on or off. When on, trend plots measurement results time correlated to the waveform being measured.

Example

This example turns the jitter measurement trend display on.

```
10 OUTPUT 707;":MEASURE:JITTER:TREND ON"
20 END
```

Query

```
:MEASure:JITTer:TRENd?
```

The :MEASure :JITTer:TRENd? query returns the state of jitter trend display.

Returned format

```
[:MEASure:JITTer:TRENd] {1 | 0}
```

Example

This example places the current setting of the jitter trend mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:TREND?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:TRENd:SMOoth

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:TRENd:SMOoth {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:TRENd:SMOoth command sets jitter trend smoothing to on or off. When on, smoothing creates a running average smoothed by the number of points set by the :JITTer:TRENd:SMOoth:POINts command.

Example

This example sets the jitter trend smoothing mode to on.

```
10 OUTPUT 707;":MEASURE:JITTer:TREND:SMOOTH ON"
20 END
```

Query

```
:MEASure:JITTer:TRENd:SMOoth?
```

The :MEASure:JITTer:TRENd:SMOoth? query returns the current jitter trend smoothing mode setting.

Returned format

```
[:MEASure:JITTer:TRENd:SMOoth] {1 | 0}
```

Example

This example places the current setting of the jitter trend smoothing mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:TREND:SMOOTH?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:TRENd:SMOoth:POINts

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:TRENd:SMOoth:POINts <points>
```

The :MEASure:JITTer:TRENd:SMOoth:POINts command sets the number of points as a set size for the data smoothing feature.

`<points>` odd integers, 3 to 1001. If out of range, the number will be rounded to nearest lower odd integer.

Example

This example sets the jitter trend smoothing points to 7.

```
10  OUTPUT 707;":MEASURE:JITTER:TREND:SMOOTH:POINTS 7"
20  END
```

Query

```
:MEASure:JITTer:TRENd:SMOoth:POINts?
```

The :MEASure:JITTer:TRENd:SMOoth:POINts? query returns the current setting for jitter trend smoothing points.

Returned format

```
[:MEASure:JITTer:TRENd:SMOoth:POINts] <value><NL>
```

`<value>` The jitter offset smoothing points setting.

Example

This example places the current value of jitter trend smoothing points in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:JITTER:TREND:SMOOTH:POINTS?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

JITTer:TRENd:VERTical

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:TRENd:VERTical {AUTO | MANual}
```

The :MEASure:JITTer:TRENd:VERTIcal command sets the jitter trend vertical mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the jitter trend vertical mode to automatic.

```
10 OUTPUT 707;":MEASURE:JITTer:TRENd:VERTical AUTO"
20 END
```

Query

```
:MEASure:JITTer:TRENd:VERTical?
```

The :MEASure:JITTer:TRENd:VERTical? query returns the current jitter trend vertical mode setting.

Returned format

```
[:MEASure:JITTer:TRENd:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the jitter trend vertical mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:JITTER:TREND:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

JITTer:TRENd:VERTical:OFFSet

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:TRENd:VERTical:OFFSet <offset>
```

The :MEASure:JITTer:TRENd:VERTial:OFFSet command sets the jitter trend vertical offset.

`<offset>` A real number for the vertical offset for the jitter measurement trend.

Example

This example sets the jitter trend vertical offset to 100 ps.

```
10  OUTPUT 707;":MEASURE:JITTER:TREND:VERTICAL:OFFSET 100E-12"
20  END
```

Query

```
:MEASure:JITTer:TRENd:VERTical:OFFSet?
```

The :MEASure:JITTer:TRENd:VERTIcal:OFFSet? query returns the jitter trend vertical offset setting.

Returned format

```
[:MEASure:JITTer:TRENd:VERTical:OFFSet] <value><NL>
```

`<value>` The jitter vertical trend offset setting.

Example

This example places the current value of jitter trend vertical offset in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:JITTER:TREND:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

JITTer:TRENd:VERTical:RANGe

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:JITTer:TRENd:VERTical:RANGe <range>
```

The :MEASure:JITTer:TRENd:VERTial:RANGe command sets the jitter trend vertical range.

`<range>` A real number for the full-scale vertical range for the jitter measurement trend.

Example

This example sets the jitter trend vertical range to 4 ns (500 ps/div X 8 div).

```
10  OUTPUT 707;":MEASURE:JITTER:TREND:VERTICAL:RANGE 4E-9"
20  END
```

Query

```
:MEASure:JITTer:TRENd:VERTical:RANGe?
```

The :MEASure:JITTer:TRENd:VERTIcal:RANGe? query returns the jitter trend vertical range setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical:RANGe] <value><NL>
```

`<value>` The jitter trend vertical range setting.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the current value of jitter trend vertical range in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:JITTER:TREND:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

NAME

Command `:MEASure:NAME {MEAS1 | MEAS2 | MEAS3 | MEAS4}, <name>`

The :MEASure:NAME commands sets the name of the specified measurement to whatever string is given to <name>. This enables you to give specific names to measurements displayed on the oscilloscope's screen.

`<name>` a quoted string

Query `:MEASure:NAME? {MEAS1 | MEAS2 | MEAS3 | MEAS4}`

The :MEASure:NAME? query returns the name of the corresponding measurement.

NCJitter

This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

Command

```
:MEASure:NCJitter <source>,<direction>,<n>,<start>
```

The :MEASure:NCJitter command measures the N cycle jitter of the waveform.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing}, specifies direction of waveform edge to make measurement.

`<n>` An integer, 1 to 99, the number of cycles in a group.

`<start>` An integer, 1 to <n> - 1, typically 1, the cycle to start measuring.

Example

This example measures the N cycle jitter on channel 1, rising edge, 5 cycles in a group, starting on the first cycle of the waveform.

```
10  OUTPUT 707;":MEASURE:NCJITTER CHANNEL1,RISING,5,1"
20  END
```

Query `:MEASure:NCJitter? <source>,<direction>,<n>,<start>`

The :MEASure:NCJitter? query returns the measured N cycle jitter time of the waveform.

Returned Format `[:MEASure:NCJitter] <value>[,<result_state>]<NL>`

`<value>` The N cycle jitter time of the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of N cycle jitter in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:NCJITTER? CHANNEL1,RIS,5,1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

NPERiod

Command `:MEASure:NPERiod <source>, <slope>, <N>`

The :MEASure:NPERiod command measures the span of time of N consecutive periods. The measurement then moves over one period and measures the span of time of the next N consecutive periods.

`<source>` the source on which the measurement is made

`<slope>` rising or falling

`<N>` An integer greater than or equal to 1.

Example This example meaures the time span of 3 consecutive periods on channel 1 (rising edge).

```
10  OUTPUT 707;":MEASURE:NPERiod CHAN1, rising, 3"
20  END
```

Query `:MEASure:NPERiod?`

NPULses

Command `:MEASure:NPULses <source>`

The :MEASure:NPULses measures the number of negative pulses on the screen.

`<source>` the source on which the measurement is made

Example This example meaures the number of negative pulses on channel 1.

```
10  OUTPUT 707;":MEASURE:NPULses CHAN1"
20  END
```

Query :MEASure:NPULses?

This query returns the result for the NPULses measurement.

NUI

Command `:MEASure:NPERiod <source>, <N>`

The :MEASure:NPERiod command measures N consecutive unit intervals. The measurement then moves over one unit interval and measures the span of time of the next N consecutive unit intervals.

`<source>` the source on which the measurement is made

`<N>` An integer greater than or equal to 1.

Example This example meaures the time span of 3 consecutive unit intervals on channel 1.

```
10  OUTPUT 707;":MEASURE:NUI CHAN1, 3"
20  END
```

Query `:MEASure:NUI?`

NWIDth

Command `:MEASure:NWIDth [<source>]`

The :MEASure:NWIDth command measures the width of the first negative pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard threshold selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:NWIDth command.

The algorithm is:

If the first edge on the screen is rising,

then

nwidth = time at the second rising edge – time at the first falling edge

else

nwidth = time at the first rising edge – time at the first falling edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example measures the width of the first negative pulse on the screen.

```
10  OUTPUT 707;":MEASURE:NWIDTH CHANNEL1"
20  END
```

Query `:MEASure:NWIDth? [<source>]`

The :MEASure:NWIDth? query returns the measured width of the first negative pulse of the specified source.

Returned Format `[:MEASure:NWIDth] <value>[,<result_state>]<NL>`

`<value>` The width of the first negative pulse on the screen using the mid-threshold levels of the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current width of the first negative pulse on the screen in the numeric variable, Width, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:NWIDTH? CHANNEL1"
30  ENTER 707;Width
40  PRINT Width
50  END
```

OVERshoot

Command

```
:MEASure:OVERshoot [<source>]
```

The :MEASure:OVERshoot command measures the overshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:OVERshoot command.

The algorithm is:

If the first edge on the screen is rising,

then

overshoot = (Local Vmax – Vtop) / Vamplitude

else

overshoot = (Vbase – Local Vmin) / Vamplitude.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the overshoot of the first edge on the screen.

```
10  OUTPUT 707;":MEASURE:OVERSHOOT CHANNEL1"
20  END
```

Query `:MEASure:OVERshoot? [<source>]`

The :MEASure:OVERshoot? query returns the measured overshoot of the specified source.

Returned Format `[:MEASure:OVERshoot] <value>[,<result_state>]<NL>`

`<value>` Ratio of overshoot to amplitude, in percent.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of overshoot in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:OVERSHOOT? CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

PERiod

Command `:MEASure:PERiod [<source>],<direction>`

The <direction> parameter is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed. When <direction> is specified, the <source> parameter is required.

The :MEASure:PERiod command measures the period of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PERiod command.

The algorithm is:

If the first edge on the screen is rising,

then

period = time at the second rising edge – time at the first rising edge

else

period = time at the second falling edge – time at the first falling edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing}

Specifies direction of edge to start measurement.

Example This example measures the period of the waveform.

```
10  OUTPUT 707;":MEASURE:PERIOD CHANNEL1"
20  END
```

Query `:MEASure:PERiod? [<source>],<direction>`

The :MEASure:PERiod? query returns the measured period of the specified source.

Returned Format `[:MEASure:PERiod] <value>[,<result_state>]<NL>`

`<value>` Period of the first complete cycle on the screen.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current period of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PERIOD? CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

PHASe

Command `:MEASure:PHASe [<source>[,<source>[,<direction>]]]`

The <direction> parameter is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

The :MEASure:PHASe command measures the phase in degrees between two edges. If two sources are specified, the phase from the specified edge of the first source to the specified edge of the second source is measured. If one source is specified, the phase is always 0.0E0.00°.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1-4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing}

Specifies direction of edge to measure.

Example This example measures the phase between channel 1 and channel 2.

```
10  OUTPUT 707;":MEASURE:PHASE CHANNEL1,CHANNEL2"
20  END
```

Query `:MEASure:PHASe? [<source>[,<source>[,<direction>]]]`

The :MEASure:PHASe? query returns the measured phase angle value.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

Returned Format `[:MEASure:PHASe] <value>[,result_state]<NL>`

`<value>` Phase angle from the first edge on the first source to the first edge on the second source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current phase angle value between channel 1 and channel 2 in the variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PHASE? CHANNEL1,CHANNEL2"
30  ENTER 707;Value
40  PRINT Value
50  END
```

PPULses

Command `:MEASure:PPULses <source>`

The :MEASure:PPULses measures the number of positive pulses on the screen.

`<source>` the source on which the measurement is made

Example This example meaures the number of positive pulses on channel 1.

```
10  OUTPUT 707;":MEASURE:PPULses CHAN1"
20  END
```

Query :MEASure:PPULses?

This query returns the result for the PPULses measurement.

PREShoot

Command

`:MEASure:PREShoot [<source>]`

The :MEASure:PREShoot command measures the preshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PREShoot command.

The algorithm is:

If the first edge on the screen is rising,

then

preshoot = (Vbase – Local Vmin) / Vamplitude

else

preshoot = (Local Vmax – Vtop) / Vamplitude.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the preshoot of the waveform on the screen.

```
10  OUTPUT 707;":MEASURE:PRESHOOT CHANNEL1"
20  END
```

Query `:MEASure:PREShoot?[<source>]`

The :MEASure:PREShoot? query returns the measured preshoot of the specified source.

Returned Format `[:MEASure:PREShoot] <value>[,<result state>]<NL>`

`<value>` Ratio of preshoot to amplitude, in percent.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of preshoot in the numeric variable, Preshoot, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PRESHOOT? CHANNEL1"
30  ENTER 707;Preshoot
40  PRINT Preshoot
50  END
```

PWIDth

Command

```
:MEASure:PWIDth [<source>]
```

The :MEASure:PWIDth command measures the width of the first positive pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PWIDth command.

The algorithm is:

If the first edge on the screen is rising,

then

pwidth = time at the first falling edge – time at the first rising edge

else

pwidth = time at the second falling edge – time at the first rising edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the width of the first positive pulse on the screen.

```
10  OUTPUT 707;":MEASURE:PWIDTH CHANNEL1"
20  END
```

Query `:MEASure:PWIDth?[<source>]`

The :MEASure:PWIDth? query returns the measured width of the first positive pulse of the specified source.

Returned Format `[:MEASure:PWIDth] <value>[,<result_state>]<NL>`

`<value>` Width of the first positive pulse on the screen in seconds.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the width of the first positive pulse on the screen in the numeric variable, Width, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PWIDTH? CHANNEL1"
30  ENTER 707;Width
40  PRINT Width
50  END
```

QUALifier<M>:CONDition

Command

```
:MEASure:QUALifier<M>:CONDition {HIGH | LOW |
INSide | OUTSide}
```

The :MEASure:QUALifier<M>:CONDition

The :MEASure:QUALifier<M>:CONDition command sets the condition when valid timing measurements are made

- Above Middle Threshold (HIGH)
- Below Middle Threshold (LOW)
- Between Upper, Lower Thresholds (INSide)
- Not Between Thresholds (OUTSide)

<M> An integer, 1-3.

Example

This example sets the level qualifier 2 condition to HIGH.

```
10  OUTPUT 707;":MEASURE:QUALIFIER2:CONDITION HIGH"
20  END
```

Query

```
:MEASure:QUALifier<M>:CONDition?
```

The :MEASure:QUALifier<M>:CONDition? query returns the condition being used of the level qualifier.

Returned Format

```
[:MEASure:QUALifier<M>:CONDition] <source><NL>
```

Example

This example places the current condition of level qualifier for timing measurements in the source variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:QUALIFIER2:CONDition?"
30  ENTER 707;Source
40  PRINT Source
50  END
```

QUALifier<M>:SOURce

The channel being selected must not be used to make a timing measurement and must be turned on.

Command

```
:MEASure:QUALifier<M>:SOURce <source>
```

The :MEASure:QUALifier<M>:SOURce command sets the source of the level qualify for timing measurements.

<source> CHANnel<N>

<N> An integer, 1- 4.

<M> An integer, 1-3.

Example

This example sets the level qualifier 2 source to the channel 1 waveform.

```
10  OUTPUT 707;":MEASURE:QUALIFIER2:SOURce CHANNEL1"
20  END
```

Query

```
:MEASure:QUALifier<M>:SOURce?
```

The :MEASure:QUALifier<M>:SOURce? query returns the source being used of the level qualifier for timing measurements.

Returned Format

```
[:MEASure:QUALifier<M>:SOURce] <source><NL>
```

Example

This example places the current source of level qualifier for timing measurements in the source variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:QUALIFIER2:SOURce?"
30  ENTER 707;Source
40  PRINT Source
50  END
```

QUALifier<M>:STATe

Command

```
:MEASure:QUALifier<M>:STATe {{ON | 1} | {OFF | 0}}
```

The :MEASure:QUALifier<M>:STATe command enables or disables level qualifying for timing measurements.

<M> An integer, 1-3.

Example

This example sets the level qualifier 2 state to ON.

```
10  OUTPUT 707;":MEASURE:QUALIFIER2:STATE ON"
20  END
```

Query

```
:MEASure:QUALifier<M>:STATe?
```

The :MEASure:QUALifier<M>:STATe? query returns the state of the level qualifier for timing measurements.

Returned Format

```
[:MEASure:QUALifier<M>:SOURce] {1 | 0}<NL>
```

Example

This example places the current state of the level qualifier for timing measurements in the state variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:QUALIFIER2:STATE?"
30  ENTER 707;state
40  PRINT state
50  END
```

RESults?

Query

```
:MEASure:RESults?
```

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values. If SENDvalid is ON, the results state is returnedt.

If more than one measurement is running continuously, the values in the :MEASure:RESults returned are duplicated for each continuous measurement from the first to last (left to right) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of five continuous measurements that can be continuously displayed at a time.

Returned Format

```
[:MEASure:RESults] <result_list><NL>
```

`<result_list>` A list of the measurement results separated with commas. The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measurement label	current	result state	min	max	mean	std dev	# of meas

Min, max, mean, std dev, and # of meas are only returned if the :MEASure:STATistics is ON. The result state is only returned if :MEASure:SENDvalid is ON. See Table 22-1 for the meaning of the result state codes.

If the :MEASure:STATistics is set to CURRENT, MAX, MEAN, MIN, or STDDEV only that particular statistic value is returned for each measurement that is on.

Example

This example places the current results of the measurements in the string variable, Result$, then prints the contents of the variable to the computer's screen.

```
10  DIM Result$[500]!Dimension variable
20  OUTPUT 707;":MEASURE:RESULTS?"
30  ENTER 707;Result$
40  PRINT Result$
50  END
```

Table 22-1 Result States

Code	Description
0	Result correct. No problem found.
1	Result questionable but could be measured.
2	Result less than or equal to value returned.
3	Result greater than or equal to value returned.
4	Result returned is invalid.
5	Result invalid. Required edge not found.
6	Result invalid. Max not found.
7	Result invalid. Min not found.
8	Result invalid. Requested time not found.
9	Result invalid. Requested voltage not found.
10	Result invalid. Top and base are equal.
11	Result invalid. Measurement zone too small.
12	Result invalid. Lower threshold not on waveform.
13	Result invalid. Upper threshold not on waveform.
14	Result invalid. Upper and lower thresholds are too close.
15	Result invalid. Top not on waveform.
16	Result invalid. Base not on waveform.
17	Result invalid. Completion criteria not reached.
18	Result invalid. Measurement invalid for this type of waveform.
19	Result invalid. waveform is not displayed.
20	Result invalid. Waveform is clipped high.
21	Result invalid. Waveform is clipped low.
22	Result invalid. Waveform is clipped high and low.
23	Result invalid. Data contains all holes.
24	Result invalid. No data on screen.
29	Result invalid. FFT peak not found.
30	Result invalid. Eye pattern not found.
31	Result invalid. No NRZ eye pattern found.
33	Result invalid. There is more than one source on creating the database.

35	**Signal may be too small to evaluate.**
36	**Result invalid. Awaiting completion of averaging.**
39	**Result invalid. Need jitter package to make this measurement or must be in jitter mode to make this measurement.**
40	**Current measurement is not on screen.**
41	**Not enough points available to recover the clock.**
42	**The loop bandwidth of the PLL is too high to recover the clock.**
43	**RJDJ pattern not found in data.**
45	**Clock recovery mode is not permitted.**
46	**Too much jitter to make a RJDJ separation.**

RISetime

Command

```
:MEASure:RISetime [<source>]
```

The :MEASure:RISetime command measures the rise time of the first displayed edge by measuring the time at the lower threshold of the rising edge, measuring the time at the upper threshold of the rising edge, then calculating the rise time with the following algorithm:

Rise time = time at upper threshold point – time at lower threshold point.

To make this measurement requires 4 or more sample points on the rising edge of the waveform.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the RISetime command. With standard thresholds selected, the lower threshold is at the 10% point and the upper threshold is at the 90% point on the rising edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the rise time of the channel 1 waveform.

```
10  OUTPUT 707;":MEASURE:RISETIME CHANNEL1"
20  END
```

Query `:MEASure:RISetime?[<source>]`

The :MEASure:RISetime? query returns the rise time of the specified source.

Returned Format `[:MEASure:RISetime] <value>[,<result_state>]<NL>`

`<value>` Rise time in seconds.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of rise time in the numeric variable, Rise, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RISETIME? CHANNEL1"
30  ENTER 707;Rise
40  PRINT Rise
50  END
```

RJDJ:ALL?

This command is only available when the N5400A/N5401A Software is installed.

Query `:MEASure:RJDJ:ALL?`

The :MEASure:RJDJ:ALL? query returns all of the RJDJ jitter measurements. These values are returned as comma separated values using the following format:

Returned Format

```
[:MEASure:RJDJ:ALL<space>]
TJ(<tj_format>),<tj_result>,<tj_state>,
RJ(<rj_format>),<rj_result>,<rj_state>,
DJ(<dj_format>),<dj_result>,<dj_state>,
PJ(<pj_format>),<pj_result>,<pj_state>,
PJ(<pj_format>),<pj_result>,<pj_state>,
DDJ(<ddj_format>),<ddj_result>,<ddj_state>,
DCD,<dcd_result>,<dcd_state>,
ISI(<isi_format>),<isi_result>,<isi_state>,
Transitions,<number_of_transitions><NL>
```

`<space>` White space (ASCII 32) character.

`<tj_format>` `<rj_format>` `<dj_format>` `<pj_format>` `<pj_format>` `<ddj_format>` `<isi_format>` The format value tells you something about how the measurement is made. For instance, TJ(1E-12) means that the TJ measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (d-d) means the measurement uses a dual-Dirac delta model to derive the measurement. A format of (p-p) means the measurement is a peak-to-peak measurement.

`<tj_result>` `<rj_result>` `<dj_result>` `<pj_result>` `<pj_result>` `<ddj_result>` `<isi_result>` The measured results for the RJDJ measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<tj_state>
<rj_state>
<dj_state>
<pj_state>
<pj_state>
<ddj_state>
<isi_state>

The measurement result state. See Table 22-1 on page 122 for a list of values and descriptions of the result state value.

<number_of_transitions>

The number of waveform transistions that have been measured.

Example

This example places the jitter measures in the Results variable and displays it on the computer's screen.

```
5   DIM Result$[500]!Dimension variable
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:ALL?"
30  ENTER 707;Results$
40  PRINT Results$
50  END
```

RJDJ:BANDwidth

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:BANDwidth {NARRow | WIDE}
```

The :MEASure:RJDJ:BANDwidth command sets the type of filtering used to separate the data dependent jitter form the random jitter and the periodic jitter.

Example

This example sets the RJ bandwidth to WIDE.

```
10  OUTPUT 707;":MEASURE:RJDJ:BANDWIDTH WIDE"
20  END
```

Query

```
:MEASure:RJDJ:BANDwidth?
```

The :MEASure:RJDJ:BANDwidth? query returns the RJ bandwidth filter setting.

Returned Format

```
[:MEASure:RJDJ:BANDwidth] {NARRow | WIDE}<NL>
```

Example

This example places the RJ filter setting the Filter variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:BANDWIDTH?"
30  ENTER 707;Filter
40  PRINT Filter
50  END
```

RJDJ:BER

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:BER {E6 | E7 | E8 | E9 | E10 | E11 |
E12 | E13 | E14 | E15 | E16 | E17 | E18}
```

The :MEASure:RJDJ:BER command sets the bit error rate for the Total Jitter (TJ) measurement. The E parameters have the following bit error rate meanings:

```
E6 = 1E-6
E7 = 1E-7
E8 = 1E-8
E9 = 1E-9
E10 = 1E-10
E11 = 1E-11
E12 = 1E-12
E13 = 1E-13
E14 = 1E-14
E15 = 1E-15
E16 = 1E-16
E17 = 1E-17
E18 = 1E-18
```

Example

This example sets the bit error rate to E16.

```
10  OUTPUT 707;":MEASURE:RJDJ:BER E16"
20  END
```

Query

```
:MEASure:RJDJ:BER?
```

The :MEASure:RJDJ:BER? query returns the bit error rate setting.

Returned Format

```
[:MEASure:RJDJ:BER] {E6 | E7 | E8 | E9 | E10 | E11 | E12 |
E13 | E14 | E15 | E16 | E17 | E18}<NL>
```

Example

This example places the bit error rate in the Rate variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:BER?"
30  ENTER 707;Rate
40  PRINT Rate
50  END
```

RJDJ:EDGE

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:EDGE {RISING | FALLING | BOTH}
```

The :MEASure:RJDJ:EDGE command sets the edge used for the RJDJ measurements.

Example

This example sets the RJDJ edge to use both edges.

```
10  OUTPUT 707;":MEASURE:RJDJ:EDGE BOTH"
20  END
```

Query

```
:MEASure:RJDJ:EDGE?
```

The :MEASure:RJDJ:EDGE? query returns the edge being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:EDGE] {RISING | FALLING | BOTH}<NL>
```

Example

This example places the current edge being used for RJDJ measurements in the edge variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:EDGE?"
30  ENTER 707;edge
40  PRINT edge
50  END
```

RJDJ:INTerpolate

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:INTerpolate {LINear | NONE}
```

The :MEASure:RJDJ:INTerpolate command sets the interpolation mode used for the RJDJ measurements.

Example

This example sets the RJDJ interpolation to use both linear.

```
10  OUTPUT 707;":MEASURE:RJDJ:INTERPOLATE LINEAR"
20  END
```

Query

```
:MEASure:RJDJ:INTerpolate?
```

The :MEASure:RJDJ:INTerpolate? query returns the edge being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:INTerpolate] {LINear | NONE}<NL>
```

Example

This example places the current interpolation mode being used for RJDJ measurements in the interpolate variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:INTERPOLATE?"
30  ENTER 707;interpolate
40  PRINT interpolate
50  END
```

RJDJ:PLENgth

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:PLENgth {AUTO |
{ARBitrary,<isi_filter_lead>,<isi_filter_lag>} |
<number_of_bits>}
```

The :MEASure:RJDJ:PLENgth command sets the number of bits used pattern length for the RJDJ measurements.

`<isi_filter_lead>` An integer number that is less than or equal to 0 that is the number of leading bits that are used to calculate the ISI filter.

`<isi_filger_lag}` An integer number that is greater than or equal to 0 that is the number of trailing bits used to calculate the ISI filter.

`<number_of_bits>` An integer number that is the length of pattern from 2 to 1024.

Example

This example sets the RJDJ bits to 5.

```
10  OUTPUT 707;":MEASURE:RJDJ:PLENgth 5"
20  END
```

Query

```
:MEASure:RJDJ:PLENgth?
```

The :MEASure:RJDJ:PLENgth? query returns the number of bits being used for the RJDJ measurements when Periodic pattern length is set. For Arbitrary pattern length, the ISI filter lead and filter lag numbers are returned.

Returned Format

```
[MEASure:RJDJ:PLENgth] {AUTO |
ARBitrary,<isi_filter_lead>,<isi_filter_lag> |
<number_of_bits>}<NL>
```

Example

This example places the current number of bits for RJDJ measurements in the bits variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:PLENgth?"
30  ENTER 707;bits
40  PRINT bits
50  END
```

RJDJ:SOURce

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:SOURce <source>
```

The :MEASure:RJDJ:SOURce command sets the source for the RJDJ measurements.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example sets the RJDJ source to the channel 1 waveform.

```
10  OUTPUT 707;":MEASURE:RJDJ:SOURce CHANNEL1"
20  END
```

Query

```
:MEASure:RJDJ:SOURce?
```

The :MEASure:RJDJ:SOURce? query returns the source being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:SOURce] <source><NL>
```

Example

This example places the current source for RJDJ measurements in the source variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
```

```
20  OUTPUT 707;":MEASURE:RJDJ:SOURce?"
30  ENTER 707;Source
40  PRINT Source
50  END
```

RJDJ:STATe

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:STATe {ON | OFF}
```

The :MEASure:RJDJ:STATe command enables or disables the RJDJ measurements.

Example

This example sets the RJDJ state to on.

```
10  OUTPUT 707;":MEASURE:RJDJ:STATE ON"
20  END
```

Query

```
:MEASure:RJDJ:STATe?
```

The :MEASure:RJDJ:STATe? query returns the state of the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:STATe] {1 | 0}<NL>
```

Example

This example places the current state of the RJDJ measurements in the state variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:STATE?"
30  ENTER 707;state
40  PRINT state
50  END
```

RJDJ:TJRJDJ?

This command is only available when the N5400A/N5401A Software is installed.

Query

```
:MEASure:RJDJ:TJRJDJ?
```

The :MEASure:RJDJ:TJRJDJ? query returns the Total Jitter (TJ), Random Jitter (RJ), and the Deterministic Jitter (DJ) measurements. These values are returned as comma separated values using the following format:

Returned Format

```
[:MEASure:RJDJ:TJRJDJ]
TJ(<tj_format>),<tj_result>,<tj_state>,
RJ(<rj_format>),<rj_result>,rj_state,
DJ(<dj_format>),<dj_result>,<dj_state><NL>
```

`<tj_format>` `<rj_format>` `<dj_format>` The format value tells you something about how the measurement is made. For instance, TJ(1E-12) means that the TJ measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (d-d) means the measurement uses from a dual-Dirac delta model used to derive the measurement. A format of (p-p) means the measurement is a peak-to-peak measurement.

`<tj_result>` `<rj_result>` `<dj_result>` The measured results for the RJDJ measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

`<tj_state>` `<rj_state>` `<dj_state>` The measurement result state. See Table 22-1 on page 122 for a list of values and descriptions of the result state value.

Example

This example places the current source for RJDJ measurements in the source variable and displays it on the computer's screen.

```
5   DIM Result$[500]!Dimension variable
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:TJRJDJ?"
30  ENTER 707;Result$
40  PRINT Result$
50  END
```

RJDJ:UNITs

This command is only available when the N5400A/N5401A Software is installed.

Command

```
:MEASure:RJDJ:UNITs {SECond | UNITinterval}
```

The :MEASure:RJDJ:UNITs command sets the unit of measure for RJDJ measurements to seconds or unit intervals.

Example

This example sets the RJDJ units to unit interval.

```
10  OUTPUT 707;":MEASURE:RJDJ:UNITS UNITINTERVAL"
20  END
```

Query

```
:MEASure:RJDJ:UNITs?
```

The :MEASure:RJDJ:UNITs? query returns the units of measure being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:UNITs] {SECond | UNITinterval}<NL>
```

Example

This example places the current units of measure for the RJDJ measurements in the units variable and displays it on the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RJDJ:UNITS?"
30  ENTER 707;units
40  PRINT units
50  END
```

SCRatch

Command

```
:MEASure:{SCRatch | CLEar}
```

The :MEASure:SCRatch command clears the measurement results from the screen. This command performs the same function as :MEASure:CLEar.

Example

This example clears the current measurement results from the screen.

```
10  OUTPUT 707;":MEASURE:SCRATCH"
20  END
```

SENDvalid

Command

```
:MEASure:SENDvalid {{OFF|0} | {ON|1}}
```

The :MEASure:SENDvalid command enables the result state code to be returned with the :MEASure:RESults? query and all other measurement queries.

Example

This example turns the send valid function on.

```
10  OUTPUT 707;":MEASURE:SENDVALID ON"
20  END
```

Query

```
:MEASure:SENDvalid?
```

The :MEASure:SENDvalid? query returns the state of the send valid control.

Returned Format

```
{:MEASure:SENDvalid] {0 | 1}<NL>
```

Example

This example places the current mode for SENDvalid in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:SENDVALID?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

See Also

Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

SETuptime

This command is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

Command

```
:MEASure:SETuptime
[<data_source>,<data_source_dir>,<clock_source>,
<clock_ source_dir>]
```

The :MEASure:SETuptime command measures the setup time between the specified clock and data source.

`<data_source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<clock_source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<data_source_dir>` {RISing | FALLing | BOTH}

Selects the direction of the data source edge. BOTH selects both edges to be measured.

`<clock_source_dir>` {RISing | FALLing}

Selects the direction of the clock source edge.

Example

This example measures the setup time from the rising edge of channel 1 to the rising edge of channel 2.

```
10  OUTPUT 707;":MEASURE:SETUPTIME CHAN1,RIS,CHAN2,RIS"
20  END
```

Query

```
:MEASure:SETuptime?
[<data_source>,<data_source_dir>,<clock_source>,
<clock_ source_dir>]
```

The :MEASure:SETuptime query returns the measured setup time between the specified clock and data source.

Returned Format

```
{:MEASure:SETuptime] <value><NL>
```

<value> Setup time in seconds.

Example

This example places the current value of setup time in the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:SETUPTIME? CHAN1,RIS,CHAN2,RIS"
30  ENTER 707;Time
40  PRINT Time
50  END
```

SLEWrate

This command is only available when the E2681A Jitter Analysis Software is installed.

Command `:MEASure:SLEWrate [<data_source>]`

The :MEASure:SLEWrate command measures the slew rate of the specified data source.

`<data_source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` is an integer, 1 - 4.

Example This example measures the slew rate of channel 1.

```
10  OUTPUT 707;":MEASURE:SLEWRATE CHAN1"
20  END
```

Query `:MEASure:SLEWrate? [<data_source>]`

The :MEASure:SLEWrate? query returns the measured slew rate for the specified source.

Returned Format `{:MEASure:SLEWrate] <value><NL>`

`<value>` Slew rate in volts per second.

Example This example places the channel 1 value of slew rate in the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:SLEWRATE? CHAN1"
```

```
30  ENTER 707;Time
40  PRINT Time
50  END
```

SOURce

Command `:MEASure:SOURce {<source>[,<source>]}`

The :MEASure:SOURce command selects the source for measurements. You can specify one or two sources with this command. All measurements except :MEASure:HOLDtime, :MEASure:SETUPtime, and :MEASure:DELTatime are made on the first specified source. The delta time measurement uses two sources if two are specified.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example selects channel 1 as the source for measurements.

```
10  OUTPUT 707;":MEASURE:SOURCE CHANNEL1"
20  END
```

Query `:MEASure:SOURce?`

The :MEASure:SOURce? query returns the current source selection.

Returned Format `[:MEASure:SOURce] <source>[,<source>]<NL>`

Example This example places the currently specified sources in the string variable, Source$, then prints the contents of the variable to the computer's screen.

```
10  DIM Source$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:SOURCE?"
30  ENTER 707;Source$
```

```
40  PRINT Source$
50  END
```

STATistics

Command

```
:MEASure:STATistics {{ON | 1} | CURRent | MAXimum | MEAN | MINimum | STDDev}
```

The :MEASure:STATistics command determines the type of information returned by the :MEASure:RESults? query. ON means all the statistics are on.

Example

This example turns all the statistics function on.

```
10  OUTPUT 707;":MEASURE:STATISTICS ON"
20  END
```

Query

```
:MEASure:STATistics?
```

The :MEASure:STATistics? query returns the current statistics mode.

Returned Format

```
[:MEASure:STATistics] {ON | CURRent | MAXimum | MEAN | MINimum | STDDev}<NL>
```

Example

This example places the current mode for statistics in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:STATISTICS?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

See Also

Refer to the :MEASure:RESults? query for information on the result returned and how it is affected by the STATistics command.

TEDGe

Command

```
:MEASure:TEDGe <meas_thres_txt>,
[<slope>]<occurrence>[,<source>]
```

The :MEASure:TEDGe command measures the time interval between the trigger event and the specified edge (threshold level, slope, and transition). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TEDGe command.

`<meas_thres_txt>` UPPer, MIDDle, or LOWer to identify the threshold.

`<slope>` { - (minus) for falling | + (plus) for rising | <none> (the slope is optional; if no slope is specified, + (plus) is assumed) }

`<occurrence>` An integer value representing the edge of the occurrence. The desired edge must be present on the display. Edges are counted with 1 being the first edge from the left on the display, and a maximum value of 65534.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query

```
:MEASure:TEDGe? <meas_thres_txt>,
<slope><occurrence> [,<source>]
```

The :MEASure:TEDGe? query returns the time interval between the trigger event and the specified edge (threshold level, slope, and transition).

Returned Format

```
[:MEASure:TEDGe] <time>[,<result_state>]<NL>
```

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example returns the time interval between the trigger event and the 90% threshold on the second rising edge of the source waveform to the numeric variable, Time. The contents of the variable are then printed to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TEDGE? UPPER,+2,CHANNEL1"
30  ENTER 707;Time
40  PRINT Time
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

THResholds:ABSolute

Command

```
:MEASure:THResholds:ABSolute <source>,
<upper_volts>,<middle_volts>,<lower_volts>
```

The :MEASure:THResholds:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

`<source>` {ALL | CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

`<N>` is an integer, 1 - 4.

`<upper_volts>`
`<middle_volts>`
`<lower_volts>` A real number specifying voltage thresholds.

Example

This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:ABSOLUTE
CHANNEL2,0.9,0.5,0.1"
30  END
```

Query

```
:MEASure:THResholds:ABSolute? <source>
```

The :MEASure:THResholds:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:ABSolute]
<upper_volts>,<middle_volts>,<lower_volts><NL>
```

Example

This example returns the upper level, middle level, and lower level voltages used to calculate the measurements on channel 1.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:ABSOLUTE? CHANNEL1"
30  ENTER 707;Upper,Middle,Lower
40  PRINT Upper,Middle,Lower
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

THResholds:HYSTeresis

Command

```
:MEASure:THResholds:HYSTeresis <source>,
<range>,<level>
```

The :MEASure:THResholds:HYSTeresis command sets the range and level voltages that are used to calculate the measurements that use them. The range is added to the level to determine the upper level voltage for measurements that use it. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

`<source>` {ALL | CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

`<N>` is an integer, 1 - 4.

`<range>` A real number specifying voltage range for the hysteresis around the level value.

`<level>` A real number specifying voltage level.

Example

This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:HYSTERESIS
CHANNEL2,0.9,0.5,0.1"
30  END
```

Query

```
:MEASure:THResholds:HYSTeresis? <source>
```

The :MEASure:THResholds:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:HYSTeresis]<range>,<level><NL>
```

Example

This example returns the range and level voltages used to calculate the measurements on channel 1.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:HYSTERESIS? CHANNEL1"
30  ENTER 707;Range,Level
40  PRINT Range,Level
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

THResholds:METHod

Command

```
:MEASure:THResholds:METHod <source>,{ABSolute | PERCent | HYSTeresis}
```

The :MEASure:THResholds:METHold command determines the way that the top and base of a waveform are calculated for all of the measurements that use them.

<source> {ALL | CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> is an integer, 1 - 4.

Example

This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:METHOD CHANNEL1,HYSTERESIS"
30  END
```

Query

```
:MEASure:THResholds:METHod?
```

The :MEASure:THResholds:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:METHod <source>,] {ABSolute | PERCent | HYSTeresis}
```

Example

This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:METHOD?"
30  ENTER 707;Method
40  PRINT Method
50  END
```

THResholds:PERCent

Command

```
:MEASure:THResholds:PERCent <source>,
<upper_pct>,<middle_pct>,<lower_pct
```

The :MEASure:THResholds:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

`<source>` {ALL | CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

`<N>` is an integer, 1 - 4.

`<upper_pct>` A real number specifying upper percentage from -24.8 to 125.0
`<middle_pct>` A real number specifying the middle percentage from -24.9 to 124.9.
`<lower_pct>` A real number specifying the lower percentage from -25.0 to 125.8

Example

This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:PERCENT CHANNEL2,100,50,0"
30  END
```

Query

```
:MEASure:THResholds:PERCent? <source>
```

The :MEASure:THResholds:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:PERCent]
<upper_pct>,<middle_pcts>,<lower_pct><NL>
```

Example

This example returns the upper level, middle level, and lower level percentages used to calculate the measurements on channel 1.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:PERCENT? CHANNEL1"
30  ENTER 707;Upper,Middle,Lower
40  PRINT Upper,Middle,Lower
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

THResholds:TOPBase:METHod

Command

```
:MEASure:THResholds:TOPBase:METHod
<source>,{ABSolute | HISTONLY | MINmax | STANdard}
```

The :MEASure:THResholds:TOPBase:METHold command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

<source> {ALL | CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> is an integer, 1 - 4.

Example

This example sets the method used to derive the top and base of a waveform to the histogram method.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:TOPBASE:METHOD
CHANNEL1,HISTONLY"
30  END
```

Query

```
:MEASure:THResholds:TOPBase:METHod?
```

The :MEASure:THResholds:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:TOPBase:METHod] {ABSolute |
HISTONLY | MINmax | STANdard}
```

Example

This example returns the method used to derive the top and base of a waveform for channel 1.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:TOPBASE:METHOD CHANNEL1"
30  ENTER 707;Method
40  PRINT Method
50  END
```

THResholds:TOPBase:ABSolute

Command

```
:MEASure:TOPBase:THResholds:ABSolute <source>,
<top_volts>,<base_volts>
```

The :MEASure:TOPBase:THResholds:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

`<source>` {ALL | CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

`<N>` is an integer, 1 - 4.

`<top_volts>`
`<base_volts>` A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example

This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:TOPBASE:ABSOLUTE
CHANNEL2,0.9,0.1"
30  END
```

Query

```
:MEASure:THResholds:TOPBase:ABSolute? <source>
```

The :MEASure:THResholds:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format

```
[:MEASure:THResholds:TOPBase:ABSolute]
<top_volts>,<base_volts><NL>
```

Example

This example returns the top level and base level voltages used to calculate the measurements on channel 1.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:THRESHOLDS:TOPBASE:ABSOLUTE? CHANNEL1"
30  ENTER 707;Top,Base
40  PRINT Top,Base
50  END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

TIEClock2

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

This command is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

Command

```
:MEASure:TIEClock2 <source>,{SECond | UNITinterval},
<direction>,{AUTO | CUSTOM,<frequency>} |
{VARiable,<frequency>,<bandwidth>} | CLOCk}
```

The :MEASure:TIEClock2 command measures time interval error on a clock. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval. If AUTO is selected, the oscilloscope selects the ideal constant clock frequency. If CUSTom is selected, you can enter your own ideal clock frequency. If VARiable is selected, a first order PLL clock recovery is used at the give clock frequency and loop bandwidth. If CLOCk is given, clock recovery is specified with the :MEASure:CLOCk:METHod command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is:

An integer, 1 - 2, for two channel Infiniium Oscilloscope.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<direction>` {RISing | FALLing | BOTH}

Specifies direction of clock edge. BOTH selects the first edge from the left-hand side of the waveform viewing area.

<frequency> A real number for the ideal clock frequency for clock recovery.

<bandwidth> A real number for the loop bandwidth of the PLL clock recovery method.

Example

This example measures the clock time interval error on the rising edge of channel 1, ideal clock frequency set to automatic, units set to seconds.

```
10  OUTPUT 707;":MEASURE:TIECLOCK2 CHANNEL1,SECOND,RISING,AUTO"
20  END
```

Query

```
:MEASure:TIEClock2? <source>,{SECond |
UNITinterval},<direction>,{AUTO |
CUSTOM,<frequency> |
{VARiable,<frequency>,<bandwidth>} | CLOCk}
```

The :MEASure:TIEClock2? query returns the current value of the clock time interval error.

Returned format

```
[:MEASure:TIEClock2] <value>[,<result_state>]<NL>
```

<value> The clock time interval error value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the current value of the clock time interval error in the variable Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;":MEASURE:TIECLOCK2?
CHANNEL1,SECOND,FALLING,CUSTOM,2.5E9"
30 ENTER 707;Value$
40 PRINT Value$
50 END
```

TIEData

This command is only available when the E2681A Jitter Analysis Software, Serial Data Analysis, or the N5400A/5401A Software is installed.

Command

```
:MEASure:TIEData <source>,{SECond | UNITinterval},
{AUTO | CUSTOM,<data_rate> |
VARiable,<data_rate>,<bandwidth> | CLOCk}
```

The :MEASure:TIEData command measures data time interval error. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval. If AUTO is selected, the oscilloscope selects the ideal data rate. If CUSTom is selected, you can enter your own ideal constant data rate. If VARiable is selected, a first order PLL clock recovery is used at a given data rate and loop bandwidth. If CLOCk is given, clock recovery as specified with the :MEASure:CLOCk:METHod is used.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is:

An integer, 1 - 2, for two channel Infiniium Oscilloscope.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<data_rate>` A real number for the ideal data rate for clock recovery.

`<bandwidth>` A real number for the loop bandwidth of the PLL clock recovery method.

Example

This example measures the data time interval error on channel 1, ideal data rate set to automatic, units set to seconds.

```
10  OUTPUT 707;":MEASURE:TIEDATA CHANNEL1,SECOND,AUTO"
20  END
```

Query

```
:MEASure:TIEData? <source>,(SECond | UNITinterval},
{AUTO | CUSTom,<frequency> |
VARiable,<frequency>,<bandwidth> | CLOCk}
```

The :MEASure:TIEData? query returns the current value of the data time interval error.

Returned format

```
[:MEASure:TIEData] <value>[,<result_state>]<NL>
```

`<value>` The data time interval error value.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the current value of the data time interval error in the variable Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":MEASURE:TIEDATA? CHANNEL1,SECOND,CUSTOM,1E9"
30  ENTER 707;Value$
40  PRINT Value$
50  END
```

TIEFilter:STARt

Command `:MEASure:TIEFilter:STARt <start_frequency>`

The :MEASure:TIEFilter:STARt command sets the starting frequency for the TIE filter.

`<start_frequency>` A real number.

Query `:MEASure:TIEFilter:STARt?`

The :MEASure:TIEFilter:STARt? query returns the current value of the starting frequency of the TIE filter.

Returned Format `[:MEASure:TIEFilter:STARt] <value><NL>`

`<value>` The start frequency for the TIE filter.

Example This example returns the current value of the starting frequency for the TIE filter then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TIEFILTER:START?"
30  ENTER 707;Start
40  PRINT Start
50  END
```

TIEFilter:STATe

Command

```
:MEASure:TIEFilter:STATe {{ON | 1} | {OFF | 0}}
```

The :MEASure:TIEFilter:STATe command enables the TIE filter for TIE data measurements.

Query

```
:MEASure:TIEFilter:STATe?
```

The :MEASure:TIEFilter:STATe? query returns the current state of the TIE data filter.

Returned Format

```
[:MEASure:TIEFilter:STATe] {0 | 1}<NL>
```

Example

This example returns the current state of the TIE data filter then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TIEFILTER:STATE?"
30  ENTER 707;State
40  PRINT State
50  END
```

TIEFilter:STOP

Command `:MEASure:TIEFilter:STOP <stop_frequency>`

The :MEASure:TIEFilter:STOP command sets the stopping frequency for the TIE filter.

`<stop_frequency>` A real number.

Query `:MEASure:TIEFilter:STOP?`

The :MEASure:TIEFilter:STOP? query returns the current value of the stopping frequency of the TIE filter.

Returned Format [:MEASure:TIEFilter:STOP] <value><NL>

`<value>` The stop frequency for the TIE filter.

Example This example returns the current value of the stopping frequency for the TIE filter then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TIEFILTER:START?"
30  ENTER 707;Stop
40  PRINT Stop
50  END
```

TIEFilter:TYPE

Command

```
:MEASure:TIEFilter:TYPE {BANDpass | LOWPass | HIGHpass}
```

The :MEASure:TIEFilter:TYPE command sets the type of TIE filter to be used.

Example

This example sets the TIE filter to highpass.

```
10  OUTPUT 707;":MEASURE:TIEFilter:TYPE HIGHpass"
20  END
```

Query

```
:MEASure:TIEFilter:TYPE?,
```

The :MEASure:TIEFilter:TYPE? query returns the current type of TIE filter being used.

Returned Format

```
[:MEASure:TIEFilter:TYPE] {BANDpass | LOWPass | HIGHpass}<NL>
```

Example

This example places the current mode for TIEFilter:TYPE in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:TIEFilter:TYPE?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

TMAX

Command `:MEASure:TMAX [<source>]`

The :MEASure:TMAX command measures the first time at which the maximum voltage of the source waveform occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMAX command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query `:MEASure:TMAX? [<source>]`

The :MEASure:TMAX? query returns the time at which the first maximum voltage occurred.

Returned Format `[:MEASure:TMAX] <time>[,<result_state>]<NL>`

`<time>` Time at which the first maximum voltage occurred or frequency where the maximum FFT amplitude occurred.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time at which the first maximum voltage occurred to the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TMAX? CHANNEL1"
30  ENTER 707;Time
```

```
40  PRINT Time
50  END
```

TMIN

Command `:MEASure:TMIN [<source>]`

The :MEASure:TMIN command measures the time at which the first minimum voltage occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMIN command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query `:MEASure:TMIN? [<source>]`

The :MEASure:TMIN? query returns the time at which the first minimum voltage occurred or the frequency where the minimum FFT amplitude occurred.

Returned Format `[:MEASure:TMIN] <time>[,<result_state>]<NL>`

`<time>` Time at which the first minimum voltage occurred.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time at which the first minimum voltage occurred to the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TMIN? CHANNEL1"
30  ENTER 707;Time
40  PRINT Time
50  END
```

TVOLt

Command

```
:MEASure:TVOLt <voltage>,[<slope>]<occurrence>
[,<source>]
```

The :MEASure:TVOLt command measures the time interval between the trigger event and the defined voltage level and transition. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TVOLt command.

The TEDGe command can be used to get the time of edges.

Query

```
:MEASure:TVOLt? <voltage>,<slope><occurrence>
[,<source>]
```

The :MEASure:TVOLt? query returns the time interval between the trigger event and the specified voltage level and transition.

`<voltage>` Voltage level at which time will be measured.

`<slope>` The direction of the waveform change when the specified voltage is crossed - rising (+) or falling (–). If no +/- sign is present, + is assumed.

`<occurrence>` The number of the crossing to be reported (if one, the first crossing is reported; if two, the second crossing is reported, etc.). The desired crossing must be present on the display. Occurrences are counted with 1 being the first occurrence from the left of the display, and a maximum value of 65534.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Returned Format

```
[:MEASure:TVOLt] <time>[,<result_state>]<NL>
```

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example returns the time interval between the trigger event and the transition through –.250 Volts on the third rising occurrence of the source waveform to the numeric variable, Time. The contents of the variable are then printed to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TVOLT? -.250,+3,CHANNEL1"
30  ENTER 707;Time
40  PRINT Time
50  END
```

UITouijitter

Command `:MEASure:UITouijitter <source>, <N>`

The :MEASure:UITouijitter command measures the difference between two consecutive N-UI measurements. The measurement then moves over one unit interval and makes another measurement. When N=1, this is analogous to cycle-cycle jitter, but measures unit intervals instead of periods. When N>1, this is analogous to N-Cycle jitter but measures unit intervals instead of periods.

`<source>` the source on which the measurement is made

`<N>` An integer greater than or equal to 1.

Example This example meaures the UI-UI jitter for 3 consecutive unit intervals on channel 1.

```
10  OUTPUT 707;":MEASURE:UITouijitter CHAN1, 3"
20  END
```

Query `:MEASure:UITouijitter?`

UNITinterval

This command is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

Command

```
:MEASure:UNITinterval <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:UNITinterval command measures the unit interval value of the selected source. Use the :MEASure:DATarate command/query to measure the data rate of the source

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<data_rate>` A real number representing the data rate.

Example

This example measures the unit interval of channel 1.

```
10 OUTPUT 707;"MEASURE:UNITINTERVAL CHANNEL1"
20 END
```

Query

```
:MEASure:UNITinterval? <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:UNITinterval? query returns the measured unit interval.

Returned Format

```
[:MEASure:UNITinterval] <value>[,<result_state>]<NL>
```

`<value>` Unit interval of the source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the current unit interval of the channel 1 waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:UNITINTERVAL? CHANNEL1”
30  ENTER 707;Value
40  PRINT Value
50  END
```

VAMPlitude

Command `:MEASure:VAMPlitude [<source>]`

The :MEASure:VAMPlitude command calculates the difference between the top and base voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAMPlitude command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example This example calculates the difference between the top and base voltage of the specified source.

```
10  OUTPUT 707;":MEASURE:VAMPLITUDE CHANNEL1"
20  END
```

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

The :MEASure:VAMPlitude? query returns the calculated difference between the top and base voltage of the specified source.

Returned Format `[:MEASure:VAMPlitude] <value>[,<result_state>]<NL>`

`<value>` Calculated difference between the top and base voltage.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the current Vamplitude value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VAMPLITUDE? CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

VAVerage

Command

```
:MEASure:VAVerage {CYCLe | DISPlay}[,<source>]
```

The :MEASure:VAVerage command calculates the average voltage over the displayed waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

`CYCLe` The CYCLe parameter instructs the average measurement to measure the average voltage across the first period on the display.

`DISPlay` The DISPlay parameter instructs the average measurement to measure all the data on the display.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example calculates the average voltage over the displayed waveform.

```
10  OUTPUT 707;":MEASURE:VAVERAGE DISPLAY,CHANNEL1"
20  END
```

Query `:MEASure:VAVerage? {CYCLe | DISPlay}[,<source>]`

The :MEASure:VAVerage? query returns the calculated average voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

Returned Format `[:MEASure:VAVerage] <value>[,<result_state>]<NL>`

`<value>` The calculated average voltage.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current average voltage in the numeric variable, Average, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VAVERAGE? DISPLAY,CHANNEL1 CHANNEL1"
30  ENTER 707;Average
40  PRINT Average
50  END
```

VBASe

Command

```
:MEASure:VBASe [<source>]
```

The :MEASure:VBASe command measures the statistical base of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VBASe command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the voltage at the base of the waveform.

```
10  OUTPUT 707;":MEASURE:VBASE CHANNEL1"
20  END
```

Query `:MEASure:VBASe? [<source>]`

The :MEASure:VBASe? query returns the measured voltage value at the base of the specified source.

Returned Format `[:MEASure:VBASe] <value>[,<result_state>]<NL>`

`<value>` Voltage at the base of the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the current voltage at the base of the waveform to the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VBASE? CHANNEL1"
30  ENTER 707;Voltage
40  PRINT Voltage
50  END
```

VLOWer

Command `:MEASure:VLOWer [<source>]`

The :MEASure:VLOWer command measures the voltage value at the lower threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VLOWer command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query `:MEASure:VLOWer?`

The :MEASure:VLOWer? query returns the measured lower threshold of the selected source.

Returned Format `[:MEASure:VLOWer] <value>[,<result_state>]<NL>`

`<value>` Voltage value at the lower threshold.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured voltage at the lower threshold of the waveform to the numeric variable, Vlower, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VLOW? CHANNEL1"
30  ENTER 707;Vlower
40  PRINT Vlower
50  END
```

VMAX

Command

```
:MEASure:VMAX [<source>]
```

The :MEASure:VMAX command measures the absolute maximum voltage present on the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMAX command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the absolute maximum voltage on the waveform.

```
10  OUTPUT 707;":MEASURE:VMAX CHANNEL1"
20  END
```

Query `:MEASure:VMAX? [<source>]`

The :MEASure:VMAX? query returns the measured absolute maximum voltage or maximum FFT amplitude present on the selected source waveform.

Returned Format `[:MEASure:VMAX] <value>[,<result_state>]<NL>`

`<value>` Absolute maximum voltage present on the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured absolute maximum voltage on the waveform to the numeric variable, Maximum, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VMAX? CHANNEL1"
30  ENTER 707;Maximum
40  PRINT Maximum
50  END
```

VMIDdle

Command `:MEASure:VMIDdle [<source>]`

The :MEASure:VMIDdle command measures the voltage level at the middle threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMIDdle command.

Query `:MEASure:VMIDdle? [<source>]`

The :MEASure:VMIDdle? query returns the voltage value at the middle threshold of the waveform.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Returned Format `[MEASure:VMIDdle] <value>[,<result_state>]<NL>`

`<value>` The middle voltage present on the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured middle voltage on the waveform to the numeric variable, Middle, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VMID? CHANNEL1"
30  ENTER 707;Middle
40  PRINT Middle
50  END
```

VMIN

Command

```
:MEASure:VMIN [<source>]
```

The :MEASure:VMIN command measures the absolute minimum voltage present on the selected source waveform. Sources are specified with :MEASure:SOURce or with the optional parameter following the :MEASure:VMIN command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the absolute minimum voltage on the waveform.

```
10  OUTPUT 707;":MEASURE:VMIN CHANNEL1"
20  END
```

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

The :MEASure:VMIN? query returns the measured absolute minimum voltage or minimum FFT amplitude present on the selected source waveform.

Returned Format `[:MEASure:VMIN] <value>[,<result_state>]<NL>`

`<value>` Absolute minimum voltage present on the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured absolute minimum voltage on the waveform to the numeric variable, Minimum, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VMIN? CHANNEL1"
30  ENTER 707;Minimum
40  PRINT Minimum
50  END
```

VPP

Command

```
:MEASure:VPP [<source>]
```

The :MEASure:VPP command measures the maximum and minimum voltages on the selected source, then calculates the peak-to-peak voltage as the difference between the two voltages. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VPP command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the peak-to-peak voltage or FFT amplitude range of the previously selected source.

```
10  OUTPUT 707;":MEASURE:VPP CHANNEL1"
20  END
```

Query `:MEASure:VPP? [<source>]`

The :MEASure:VPP? query returns the specified source peak-to-peak voltage.

Returned Format `[:MEASure:VPP] <value>[,<result_state>]<NL>`

`<value>` Peak-to-peak voltage of the selected source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current peak-to-peak voltage in the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VPP? CHANNEL1"
30  ENTER 707;Voltage
40  PRINT Voltage
50  END
```

VRMS

Command

```
:MEASure:VRMS {CYCLe | DISPlay},{AC | DC} [,<source>]
```

The :MEASure:VRMS command measures the RMS voltage of the selected waveform by subtracting the average value of the waveform from each data point on the display. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VRMS command.

`CYCLe` The CYCLe parameter instructs the RMS measurement to measure the RMS voltage across the first period of the display.

`DISPlay` The DISPLay parameter instructs the RMS measurement to measure all the data on the display. Generally, RMS voltage is measured across one waveform or cycle, however, measuring multiple cycles may be accomplished with the DISPLay option. The DISPlay parameter is also useful when measuring noise.

`AC` The AC parameter is used to measure the RMS voltage subtracting the DC component.

`DC` The DC parameter is used to measure RMS voltage including the DC component.

The AC RMS, DC RMS, and VAVG parameters are related as in this formula:

$DCVRMS^2 = ACVRMS^2 + VAVG^2$

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the RMS voltage of the previously selected waveform.

```
10  OUTPUT 707;":MEASURE:VRMS CYCLE,AC,CHANNEL1"
20  END
```

Query `:MEASure:VRMS? {CYCLe | DISPlay},{AC | DC} [,<source>]`

The :MEASure:VRMS? query returns the RMS voltage of the specified source.

Returned Format `[:MEASure:VRMS] <value>[,<result_state>]<NL>`

`<value>` RMS voltage of the selected waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current AC RMS voltage over one period of the waveform in the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VRMS? CYCLE,AC,CHANNEL1"
30  ENTER 707;Voltage
40  PRINT Voltage
50  END
```

VTIMe

Command

```
:MEASure:VTIMe <time>[,<source>]
```

The :MEASure:VTIMe command measures the voltage at the specified time. The time is referenced to the trigger event and must be on the screen. When an FFT function is the specified source, the amplitude at the specified frequency is measured. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTIMe command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<time>` A real number for time from trigger in seconds, or frequency in Hertz for an FFT (when a function is set to FFT or a waveform memory contains an FFT).

Query

```
:MEASure:VTIMe? <time>[,<source>]
```

The :MEASure:VTIMe? query returns the measured voltage or amplitude.

Returned Format

```
[:MEASure:VTIMe] <value>[,<result_state>]<NL>
```

`<value>` Voltage at the specified time. When the source is an FFT function, the returned value is the vertical value at the horizontal setting passed in the VTIMe <time> parameter. The time parameter is in Hertz when an FFT function is the source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example

This example places the voltage at 500 ms in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VTIME? 500E-3,CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

VTOP

Command

```
:MEASure:VTOP [<source>]
```

The :MEASure:VTOP command measures the statistical top of the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTOP command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the voltage at the top of the waveform.

```
10  OUTPUT 707;":MEASURE:VTOP CHANNEL1"
20  END
```

Query `:MEASure:VTOP? [<source>]`

The :MEASure:VTOP? query returns the measured voltage at the top of the specified source.

Returned Format `[:MEASure:VTOP] <value>[,<result_state>]<NL>`

`<value>` Voltage at the top of the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the voltage at the top of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VTOP? CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

VUPPer

Command

```
:MEASure:VUPPer [<source>]
```

The :MEASure:VUPPer command measures the voltage value at the upper threshold of the waveform. Sources are specified with the MEASure:SOURce command or with the optional parameter following the :MEASure:VUPPer command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

`<N>` CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Example

This example measures the voltage at the upper threshold of the waveform.

```
10  OUTPUT 707;":MEASURE:VUPPer CHANNEL1"
20  END
```

Query `:MEASure:VUPPer? [<source>]`

The :MEASure:VUPPer? query returns the measured upper threshold value of the selected source.

Returned Format `[:MEASure:VUPPer] <value>[,<result_state>]<NL>`

`<value>` Voltage at the upper threshold.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the voltage at the upper threshold of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VUPPER? CHANNEL1"
30  ENTER 707;Value
40  PRINT Value
50  END
```

WINdow

Command

```
:MEASure:WINdow {ZOOM | All}, {MEASN}
```

The :MEASure:WINdow command enables you to only make measurements in the zoom window (measurement gating).

Example

This example gates Measurement 1 to the zoom window.

```
10 OUTPUT 707;"MEASURE:WINdow ZOOM, MEAS1"
20 END
```

Query

:MEASure:WINdow? {MEAS*N*}

This query returns whether the measurement is being performed on the zoomed portion of the waveform (ZOOM) or the entire acquisition (ALL).

23

Root Level Commands

Root Level Commands

Root level commands control many of the basic operations of the oscilloscope that you can select by pressing the labeled keys on the front panel. These commands are always recognized by the parser if they are prefixed with a colon, regardless of the current tree position. After executing a root level command, the parser is positioned at the root of the command tree.

These root level commands and queries are implemented in the Infiniium Oscilloscopes:

- ADER? (Acquisition Done Event Register)
- AER? (Arm Event Register)
- ATER? (Auto Trigger Event Register)
- AUToscale
- AUToscale:CHANnels
- AUToscale:PLACement
- AUToscale:VERTical
- BEEP
- BLANk
- CDISplay
- DIGitize
- MTEE (Mask Test Enable Register)
- MTER? (Mask Test Event Register)
- MODel?
- OPEE (Operation Status Enable)
- OPER? (Operation Status Register)
- OVLRegister
- PDER?
- PRINt
- RECall:SETup
- RUN
- SERial (Serial Number)
- SINGle
- STATus?
- STOP
- STORe:JITTer

- STORe:SETup
- STORe:WAVeform
- TER? (Trigger Event Register)
- VIEW

ADER? (Acquisition Done Event Register)

Query

```
:ADER?
```

The :ADER? query reads the Acquisition Done Event Register and returns 1 or 0. After the Acquisition Done Event Register is read, the register is cleared. The returned value 1 indicates an acquisition completed event has occurred and 0 indicates an acquisition completed event has not occurred.

Once the Done bit is set, it is cleared only by doing :ADER? or by sending a *CLS command.

Returned Format

```
{1 | 0}<NL>
```

AER? (Arm Event Register)

Query `:AER?`

The :AER? query reads the Arm Event Register and returns 1 or 0. After the Arm Event Register is read, the register is cleared. The returned value 1 indicates a trigger armed event has occurred and 0 indicates a trigger armed has not occurred.

Arm Event Returns

:AER? will allow the Arm Event to return either immediately (if you have armed but not triggered) or on the next arm (if you have already triggered). However, *CLS is always required to get an SRQ again.

Once the AER bit is set, it is cleared only by doing :AER? or by sending a *CLS command.

Returned Format `{1 | 0}<NL>`

ATER? (Auto Trigger Event Register)

Query `:ATER?`

The :ATER? query reads the Auto Trigger Event Register and returns 1 or 0. After the Auto Trigger Event Register is read, the register is cleared. The returned value 1 indicates an auto trigger event has occurred and 0 indicates an auto trigger event has not occurred.

Returned Format `{1 | 0}<NL>`

AUToscale

Command

```
:AUToscale
```

The :AUToscale command causes the oscilloscope to evaluate all input waveforms and find the optimum conditions for displaying the waveform. It searches each of the channels for input waveforms and shuts off channels where no waveform is found. It adjusts the vertical gain and offset for each channel that has a waveform, and sets the time base on the lowest numbered input channel that has a waveform.

The trigger is found by searching each channel, starting with channel 4, then channel 3, channel 2, and channel 1, until a trigger waveform is detected. If waveforms cannot be found on any vertical input, the oscilloscope is returned to its former state.

Autoscale sets the following:

- Channel Display, Scale, and Offset
- Trigger Sweep, Mode, Edge, Source, Level, Slope, Hysteresis, and Holdoff
- Acquisition Sampling Rate and Memory Depth
- Time Base Scale and Position
- Marker Mode Set to Measurement
- Resets Acquisition Completion Criteria to 90%

Autoscale turns off the following:

- Measurements on sources that are turned off
- Functions
- Windows
- Memories

No other controls are affected by Autoscale.

Example

This example automatically scales the oscilloscope for the input waveform.

```
10  OUTPUT 707;":AUTOSCALE"
20  END
```

AUToscale:CHANnels {ALL | DISPlayed}

Command

```
:AUToscale:CHANNels {ALL | DISPlayed}
```

The :AUToscale:CHANnels command selects whether to apply autoscale to all of the input channels or just the input channels that are currently displayed.

Example

This example automatically scales only the displayed channels.

```
10  OUTPUT 707;":AUTOSCALE:CHANnels DISPlayed"
20  END
```

AUToscale:PLACement {STACk | SEParate | OVERlay}

Command

```
:AUToscale:PLACement {STACk | SEParate | OVERlay}
```

The :AUToscale:PLACement command controls how the waveforms are displayed on the oscilloscope when the autoscale command is used. If Stack is chosen then each waveform's amplitude is decreased and then the waveforms are offset so each takes up a different vertical portion of the screen. This makes it easier to view them, but decreases the accuracy of any measurements performed on the waveforms because they no longer take up the full dynamic range of th ADC (analog to digital converter). If Separate is chosen then the screen is divided into the same number of grids that there are waveforms (for example, if three waveforms are displayed then the screen will be divided into three grids). Each grid represents the full dynamic range of the ADC so this choice maximizes measurement accuracy while still separating the waveforms so they are easy to see. If the Overlay option is chosen then the waveforms are displayed on top of each other. This maximizes measurement accuracy, but can making viewing difficult.

Example

This example automatically overlays the waveforms after an autoscale.

```
10  OUTPUT 707;":AUTOSCALE:OVERlay ON"
20  END
```

Query

```
:AUToscale:PLACement?
```

AUToscale:VERTical

Command

```
:AUToscale:VERTical {CHANnel<N>}
```

The :AUToscale:VERTical command autoscales the vertical position and scaling for the corresponding channel without changing anything else (for example, trigger or timebase settings).

Example

This example automatically autoscales the vertical position and scale for the waveform on Channel 1.

```
10  OUTPUT 707;":AUTOSCALE:VERTical CHAN1"
20  END
```

NOTE

If you are using software 2.10 or earlier, the command syntax is (lower-case “t” in “vertical”):

AUToscale:VERtical <CHANnel 1 | CHANnel 2 | CHANnel 3 | CHANnel 4>

BEEP

Command

```
:BEEP <frequency>,<duration>
```

The :BEEP command makes the oscilloscope beep at a defined frequency and duration.

`<frequency>` A real number representing frequency of beep in Hertz.

`<duration>` A real number representing duration of beep in milliseconds.

Example

This example will create a beep at 1000 Hz for 500 ms.

```
10  OUTPUT 707;":BEEP 1000,500"
20  END
```

BLANk

Command

```
:BLANk {CHANnel<N> | FUNCtion<N> | HISTogram |
WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalize
| ALL}
```

The :BLANk command turns off an active channel, function, histogram, waveform memory, measurement trend, measurement spectrum, or Feed-Forward Equalized waveform. The :VIEW command turns them on.

<N> An integer, 1 - 4.

Example

This example turns off channel 1.

```
10  OUTPUT 707;":BLANK CHANNEL1"
20  END
```

CDISplay

Command

```
:CDISplay
```

The :CDISplay command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all of the data in active channels and functions is erased; however, new data is displayed on the next acquisition. Waveform memories are not erased.

Example

This example clears the oscilloscope display.

```
10  OUTPUT 707;":CDISPLAY"
20  END
```

DIGitize

Command

```
:DIGitize [CHANnel<N>][,...]
```

<N> An integer, 1 - 4.

The :DIGitize command invokes a special mode of data acquisition that is more efficient than using the :RUN command. This command initializes the selected channels, then acquires them according to the current oscilloscope settings. When all waveforms are completely acquired, the oscilloscope is stopped. The waveform completion criteria is set with the ":ACQuire:COMPlete" command.

If you specify channel or function parameters, then these are the only waveforms acquired and the display waveforms of the specified channels and functions are turned off.

Full Range of Measurement and Math Operators are Available

Even though digitized waveforms are not displayed, you may perform the full range of measurement and math operators on them.

If you use the :DIGitize command with no parameters, the digitize operation is performed on the channels or functions that are being displayed in the Infiniium waveform viewing area. In this case, the display state of the acquired waveforms is not changed after the :DIGitize command is completed. Because the command executes more quickly without parameters, this form of the command is useful for repetitive measurement sequences. You can also use this mode if you want to view the digitize results because the display state of the digitized waveforms is not affected.

See the Sample Programs in chapter 6 for examples of how to use :DIGitize and its related commands.

Example

This example acquires data on channel 1.

```
10  OUTPUT 707;":DIGITIZE CHANNEL1"
20  END
```

The ACQuire subsystem commands set up conditions such as COUNt for the next :DIGitize command. The WAVeform subsystem commands determine how the data is transferred out of the oscilloscope, and how to interpret the data.

MTEE

Command `:MTEE <enable_mask>`

The :MTEE command is used to set bits in the Mask Test Enable Register. This register enables the following bits of the Mask Test Event Register:

`<enable_mask>` Bit 0 - Mask Test Complete
Bit 1 - Mask Test Fail
Bit 2 - Mask Low Amplitude
Bit 3 - Mask High Amplitude
Bit 4 - Mask Align Complete
Bit 5 - Mask Align Fail
Bit 6-7 are not used and are set to zero (0).

Query `:MTEE?`

The :MTEE? query returns the value stored in the Mask Test Enable Register.

Returned Format `[:MTEE] <enable_mask>`

Example

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

```
OUTPUT 707;"MTEE 2"
```

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

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

MTER?

Query `:MTER?`

The :MTER? query returns the value stored in the Mask Test Event Register. The bits stored in the register have the following meanings:

Bit 0 Mask Test Complete bit is set whenever the mask test is complete.

Bit 1 Mask Test Fail bit is set whenever the mask test failed.

Bit 2 Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.

Bit 3 Mask High Amplitude bit is set whenever the signal is above the mask amplitude.

Bit 4 Mask Align Complete bit is set whenever the mask align is complete.

Bit 5 Mask Align Fail bit is set whenever the mask align failed.

The Mask Test Event Register is read and cleared by the MTER? query. The register output is enabled or disabled using the mask value supplied with the MTEE command.

`Returned Format` `0-63 decimal value.`

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

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

MODel?

Query

```
:MODel?
```

The :MODel? query returns the model number for the oscilloscope.

Returned Format

A six-character alphanumeric model number in quotation marks. Output is determined by header and longform status as in Table 23-1.

Table 23-1 MODel? Returned Format

HEADER		LONGFORM		RESPONSE
ON	OFF	ON	OFF	
	X		X	5485xA
	X	X		5485xA
X			X	:MOD 5485xA
X		X		:MODEL 5485xA

Where "x" in the Response 5485xA = 3, 4, or 5

Example

This example places the model number in a string variable, Model$, then prints the contents of the variable on the computer's screen.

```
10 Dim Model$[13]!Dimension variable
20 OUTPUT 707;":MODEL?"
30 ENTER 707; Model$
40 PRINT MODEL$
50 END
```

OPEE

Command `:OPEE <mask>`

`<mask>` The decimal weight of the enabled bits.

The :OPEE command sets a mask in the Operation Status Enable register. Each bit that is set to a "1" enables that bit to set bit 7 in the status byte register, and potentially causes an SRQ to be generated. Bit 5, Wait for Trig is used. Other bits are reserved.

Query `:OPEE?`

The query returns the current value contained in the Operation Status Enable register as a decimal number.

Returned Format `[OPEE] <value><NL>`

OPER?

Query `:OPER?`

The :OPER? query returns the value contained in the Operation Status Register as a decimal number. This register contains the WAIT TRIG bit (bit 5) and the OVLR bit (bit 11).

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

Returned Format `<value><NL>`

OVLRegister?

Query `:OVLRegister?`

The :OVLRegister? query returns the value stored in the Overload Event Register. The integer value returned by this query represents the channels as follows:

Bit 0 - Channel 1
Bit 1 - Channel 2
Bit 2 - Channel 3
Bit 3 - Channel 4
Bits 7-4 are not used and are set to zero (0).

Returned Format `<value><NL>`

PDER? (Processing Done Event Register)

Query `:PDER?`

The :PDER? query reads the Processing Done Event Register and returns 1 or 0. After the Processing Done Event Register is read, the register is cleared. The returned value 1 indicates indicates that all math and measurements are complete and 0 indicates they are not complete. :PDER? is non-blocking.

:PDER? can be used in place of :ADER?.

Returned Format `{1 | 0}<NL>`

PRINt

Command

`:PRINt`

The :PRINt command outputs a copy of the screen to a printer or other device destination specified in the HARDcopy subsystem. You can specify the selection of the output and the printer using the HARDcopy subsystem commands.

Example

This example outputs a copy of the screen to a printer or a disk file.

```
10  OUTPUT 707;":PRINT"
20  END
```

RECall:SETup

Command

```
:RECall:SETup <setup_memory_num>
```

<setup_memory_num> Setup memory number, an integer, 0 through 9.

The :RECall:SETup command recalls a setup that was saved in one of the oscilloscope's setup memories. You can save setups using either the :STORe:SETup command or the front panel.

Examples

This command recalls a setup from setup memory 2.

```
10 OUTPUT 707;":RECall:SETup 2"
20 END
```

RUN

Command

```
:RUN
```

The :RUN command starts the oscilloscope running. When the oscilloscope is running, it acquires waveform data according to its current settings. Acquisition runs repetitively until the oscilloscope receives a :STOP command, or until there is only one acquisition if Trigger Sweep is set to Single. However, the :TRIGger:SWEep SINGle should not be used in new programs. The :SINGle command should be used instead to acquire a single acquisition.

Example

This example causes the oscilloscope to acquire data repetitively.

```
10  OUTPUT 707;":RUN"
20  END
```

SERial (Serial Number)

Command `:SERial {<serial_number>}`

The :SERial command sets the serial number of the oscilloscope. A serial number was entered in your oscilloscope by Agilent Technologies before it was shipped to you. Therefore, setting the serial number is not normally required unless the oscilloscope is serialized for a different application.

The oscilloscope's serial number is part of the string returned for the *IDN? query described in the Common Commands chapter.

`<serial_number>` A ten-character alphanumeric serial number enclosed with quotation marks.

Example This example sets the serial number for the oscilloscope to "US12345678".

```
10  OUTPUT 707;":SERIAL ""US12345678"""
20  END
```

Query `:SERial?`

The query returns the current serial number string for the oscilloscope.

Returned Format `[:SERial] US12345678`

Example This example places the serial number for the oscilloscope in the string variable Serial?, then prints the contents of the variable to the computer's screen.

```
10 Dim Serial$[50]!Dimension variable
20 OUTPUT 707;":SERIAL?"
30 ENTER 707; Serial$
40 PRINT SERIAL$
50 END
```

SINGle

Command

`:SINGle`

The :SINGle command causes the oscilloscope to make a single acquisition when the next trigger event occurs. However, this command does not set the :TRIGger:SWEep to SINGle.

Example

This example sets up the oscilloscope to make a single acquisition when the next trigger event occurs.

```
10  OUTPUT 707;":SINGLE"
20  END
```

See Also

:TRIGger:SWEep AUTO|TRIGgered|SINGle for how to turn the single sweep off.

STATus?

Query

```
:STATus? {CHANnel<N> | FUNCtion<N> | HISTogram |
WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}
```

The :STATus? query shows whether the specified channel, function, wmemory, histogram, measurement trend, measurement spectrum, or equalized waveform is on or off. A return value of 1 means on and a return value of 0 means off.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Returned Format

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

Example

This example returns and prints the current status of channel 1.

```
10  OUTPUT 707;":STATUS? CHANNEL1"
30  ENTER 707;Current$
40  PRINT Current$
50  END
```

STOP

Command

`:STOP`

The :STOP command causes the oscilloscope to stop acquiring data. To restart the acquisition, use the :RUN or :SINGle command.

Example

This example stops the current data acquisition.

```
10  OUTPUT 707;":STOP"
20  END
```

STORe:JITTer

Command `:STORe:JITTer <file_name>`

The :STORe:JITTer command saves all of the RJ/DJ jitter measurement data to the specified file name. The file that is created has a header section followed by the RJ/DJ measurement results section. After the RJ/DJ measurement results section is the data for each of the measurements. Each data section has a header showing what the measurement data is that follows.

`<file_name>` A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example stores the RJ/DJ jitter measurements to a file.

```
10  OUTPUT 707;":STORE:JITTer "c:\Document and Settings\All
Users\Shared Documents\Infiniium\Data\jitter""
20  END
```

STORe:SETup

Command

```
:STORe:SETup <setup_memory_num>
```

`<setup_memory_num>` Setup memory number, an integer, 0 through 9.

The :STORe:SETup command saves the current oscilloscope setup in one of the setup memories.

Example

This example stores the current oscilloscope setup to setup memory 0.

```
10  OUTPUT 707;":STORE:SETUP 0"
20  END
```

STORe:WAVeform

Command

```
:STORe:WAVeform {{CHANnel<N> | FUNCtion<N> |
WMEMory<N> | MTRend | MSPectrum},{WMEMory<N>}}
```

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

<N> An integer, 1 - 4.

The :STORe:WAVeform command copies a channel, function, stored waveform, measurement trend, or measurement spectrum to a waveform memory. The parameter preceding the comma specifies the source and can be any channel, function, or waveform memory. The parameter following the comma is the destination, and can be any waveform memory.

The :WAVeform:VIEW command determines the view of the data being stored.

Example

This example copies channel 1 to waveform memory 3.

```
10  OUTPUT 707;":STORE:WAVEFORM CHANNEL1,WMEMORY3"
20  END
```

TER? (Trigger Event Register)

Query

```
:TER?
```

The :TER? query reads the Trigger Event Register. A “1” is returned if a trigger has occurred. A “0” is returned if a trigger has not occurred. The autotrigger does not set this register. The register is set to a value of 1 only when the waveform meets the trigger criteria.

Returned Format

```
{1 | 0}<NL>
```

Example

This example checks the current status of the Trigger Event Register, places the status in the string variable, Current$, then prints the contents of the variable to the computer's screen.

```
10  DIM Current$[50]!Dimension variable
20  OUTPUT 707;":TER?"
30  ENTER 707;Current$
40  PRINT Current$
50  END
```

Once this bit is set, you can clear it only by reading the register with the :TER? query, or by sending a *CLS common command. After the Trigger Event Register is read, it is cleared.

VIEW

Command

```
:VIEW {CHANnel<N> | FUNCtion<N> | HISTogram |
WMEMory<N> | MSTrend | MSPectrum}
```

The :VIEW command turns on a channel, function, histogram, or waveform memory.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

<N> An integer, 1 - 4.

Example

This example turns on channel 1.

```
10  OUTPUT 707;":VIEW CHANNEL1"
20  END
```

See Also

The :BLANk command turns off a channel, function, histogram, or waveform memory.

Self-Test Commands

Self-Test Commands

The SELFtest subsystem commands set up the self-test dialog and run the Infiniium-Series Oscilloscopes Self-Tests.

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

These SELFtest commands and queries are implemented in the Infiniium Oscilloscopes:

- CANCel
- SCOPETEST

CANCel

Command

```
:SELFtest:CANCel
```

The :SELFtest:CANCel command stops the currently running selftest.

Example

This example stops the currently running selftest.

```
10 OUTPUT 707;":SELF:CANC"
20 END
```

SCOPETEST

Command

```
:SELFtest:SCOPETEST
```

The :SELFtest:SCOPETEST command brings up the self-test dialog in customer self-test mode (Service Extensions Off) and runs the test, “Scope Self Tests.” Use the :SELFtest:SCOPETEST? query to determine the status of the test.

Example

This example brings up the self-test dialog and runs the oscilloscope self-tests.

```
10 OUTPUT 707;":SELF:SCOPETEST"
20 END
```

Query

```
:SELFtest:SCOPETEST?
```

Returned Format

```
[:SELFtest:SCOPETEST] <test_name>,<test_status>,
<time_stamp><NL>
```

<test_status>	Status Description
FAILED	Test completed and failed.
PASSED	Test completed and passed.
WARNING	Test passed but warning message was issued.
CANCELLED	Test was cancelled by user.
NODATA	Self-tests have not been executed on this instrument.
INPROGRESS	Test is in progress.

<test_name> A string as follows: “Scope Self Tests”.

<time_stamp> The time stamp follows the test name and test status, and is the part of the returned string that includes the date and time, in the format:
“29 NOV 2002 10:13:35”.

Example

This example places the current status of the self-test in the string variable, Txt$, then prints the contents of the variable to the computer's screen.

```
10 DIM Txt$[64]
20 OUTPUT 707;":SELF:SCOPETEST?"
30 ENTER 707;Txt$
40 PRINT Txt$
50 END
```

25

System Commands

System Commands

SYSTem subsystem commands control the way query responses are formatted, send and receive setup strings, and enable reading and writing to the advisory line of the oscilloscope. You can also set and read the date and time in the oscilloscope using the SYSTem subsystem commands.

These SYSTem commands and queries are implemented in the Infiniium Oscilloscopes:

- DATE
- DEBug
- DSP
- ERRor?
- HEADer
- LOCK
- LONGform
- PRESet
- SETup
- TIME

DATE

Command `:SYSTem:DATE <day>,<month>,<year>`

The :SYSTem:DATE command sets the date in the oscilloscope, and is not affected by the *RST common command.

`<year>` Specifies the year in the format <yyyy> | <yy>. The values range from 1992 to 2035.

`<month>` Specifies the month in the format <1, 2, . . . 12> | <JAN, FEB, MAR . . .>.

`<day>` Specifies the day in the format <1 . . . 31>.

Example This example sets the date to December 1, 2002.

```
10  OUTPUT 707;":SYSTEM:DATE 1,12,02"
20  END
```

Query `:SYSTem:DATE?`

The :SYSTem:DATE? query returns the current date in the oscilloscope.

Returned Format `[:SYSTem:DATE] <day> <month> <year><NL>`

Example This example queries the date.

```
10 DIM Date$ [50]
20 OUTPUT 707;":SYSTEM:DATE?"
30 ENTER 707; Date$
40 PRINT Date$
```

DEBug

Command

```
:SYSTem:DEBug {{ON|1}[,<output_mode>[,"<file_name>"
[,<create_mode>]]] | {OFF|0}}
```

The :SYSTem:DEBug command turns the debug m ode on and off. This mode enables the tracing of incoming GPIB commands. If you select CREate mode, a new file is created, and/or an existing file is overwritten. If you select APPend mode, the information is appended to an existing file. The :SYSTem:DEBug command shows any header and/or parameter errors.

The default create mode is CREate, the default output mode is FileSCReen, and the default file name is c:\Document and Settings\All Users\Shared Documents\Infiniium\Data\debug.txt. In debug mode, the File View button lets you view the current debug file, or any other debug file. This is a read-only mode.

`<output_mode>` {FILE | SCReen | FileSCReen}

`<file_name>` An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The file name assumes the present working directory if a path does not precede the file name.

`<create_mode>` {CREate | APPend}

Examples

This example turns on the debug/trace mode and creates a debug file.

```
10  OUTPUT 707;":SYSTEM:DEBUG ON,FILE,
    "C:\Document and Settings\All Users\Shared
Documents\Infiniium\Data\pacq8xx.txt",CREATE"
20  END
```

The created file resembles:

```
Debug information file C:\Document and Settings\All Users\Shared
Documents\Infiniium\Data\pacq8xx.txt
Date:  1 DEC 2002
Time:  09:59:35
Model: 54853A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>
?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
```

This example appends information to the debug file.

```
10  OUTPUT 707;":SYSTEM:DEBUG ON,FILE,
    "C:\Document and Settings\All Users\Shared
Documents\Infiniium\Data\pacq8xx.txt",APPEND"
20  END
```

After appending information, the file resembles:

```
Debug information file C:\Document and Settings\All Users\Shared
Documents\Infiniium\Data\pacq8xx.txt
Date:  1 DEC 2002
Time:  09:59:35
Model: 54853A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>
?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$

Debug information file C:\Document and Settings\All Users\Shared
Documents\Infiniium\Data\pacq8xx.txt appended
Date:  1 DEC 2002
Time:  10:10:35
Model: 54853A
Serial#:   sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>
?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
```

Query

`:SYSTem:DEBug?`

The :SYSTem:DEBug? query returns the current debug mode settings.

Returned Format

```
[:SYSTem:DEBug]  {{1,<output_mode>,"<file_name>",
<create_mode>} | 0} <NL>
```

DSP

Command

```
:SYSTem:DSP "<string>"
```

The :SYSTem:DSP command writes a quoted string, excluding quotation marks, to the advisory line of the instrument display. If you want to clear a message on the advisory line, send a null (empty) string.

`<string>` An alphanumeric character array up to 86 bytes long.

Example

This example writes the message, “Test 1” to the advisory line of the oscilloscope.

```
10  OUTPUT 707;":SYSTEM:DSP ""Test 1"""
20  END
```

Query

```
:SYSTem:DSP?
```

The :SYSTem:DSP? query returns the last string written to the advisory line. This may be a string written with a :SYSTem:DSP command, or an internally generated advisory.

The string is actually read from the message queue. The message queue is cleared when it is read. Therefore, the displayed message can only be read once over the bus.

Returned Format

```
[:SYSTem:DSP] <string><NL>
```

Example

This example places the last string written to the advisory line of the oscilloscope in the string variable, Advisory$. Then, it prints the contents of the variable to the computer's screen.

```
10  DIM Advisory$[89]!Dimension variable
20  OUTPUT 707;":SYSTEM:DSP?"
30  ENTER 707;Advisory$
40  PRINT Advisory$
50  END
```

ERRor?

Query `:SYSTem:ERRor? [{NUMBer | STRing}]`

The :SYSTem:ERRor? query outputs the next error number in the error queue over the GPIB. When either NUMBer or no parameter is specified in the query, only the numeric error code is output. When STRing is specified, the error number is output followed by a comma and a quoted string describing the error. Table 31-1 lists the error numbers and their corresponding error messages.

Returned Format `[:SYSTem:ERRor] <error_number>[,<quoted_string>]<NL>`

`<error_number>` A numeric error code.

`<quoted_string>` A quoted string describing the error.

Example This example reads the oldest error number and message in the error queue into the string variable, Condition$, then prints the contents of the variable to the computer's screen.

```
10  DIM Condition$[64]!Dimension variable
20  OUTPUT 707;":SYSTEM:ERROR? STRING"
30  ENTER 707;Condition$
40  PRINT Condition$
50  END
```

Infiniium Oscilloscopes have an error queue that is 30 errors deep and operates on a first-in, first-out (FIFO) basis. Successively sending the :SYSTem:ERRor? query returns the error numbers in the order that they occurred until the queue is empty. When the queue is empty, this query returns headers of 0, “No error.” Any further queries return zeros until another error occurs. Note that front-panel generated errors are also inserted in the error queue and the Event Status Register.

Send *CLS Before Other Commands or Queries

Send the *CLS common command to clear the error queue and Event Status Register before you send any other commands or queries.

See Also The “Error Messages” chapter for more information on error messages and their possible causes.

HEADer

Command

```
:SYSTem:HEADer {{ON|1} | {OFF|0}}
```

The :SYSTem:HEADer command specifies whether the instrument will output a header for query responses. When :SYSTem:HEADer is set to ON, the query responses include the command header.

Example

This example sets up the oscilloscope to output command headers with query responses.

```
10  OUTPUT 707;":SYSTEM:HEADER ON"
20  END
```

Query

```
:SYSTem:HEADer?
```

The :SYSTem:HEADer? query returns the state of the :SYSTem:HEADer command.

Returned Format

```
[:SYSTem:HEADer] {1|0}<NL>
```

Example

This example examines the header to determine the size of the learn string. Memory is then allocated to hold the learn string before reading it. To output the learn string, the header is sent, then the learn string and the EOF.

```
10  DIM Header$[64]
20  OUTPUT 707;"syst:head on"
30  OUTPUT 707;":syst:set?"
40  More_chars:  !
50  ENTER 707 USING "#,A";This_char$
60  Header$=Header$&This_char$
70  IF This_char$<>"#" THEN More_chars
80  !
90  ENTER 707 USING "#,D";Num_of_digits
100 ENTER 707 USING "#,"&VAL$(Num_of_digits)&"D";Set_size
110 Header$=Header$&"#"&VAL$(Num_of_digits)&VAL$(Set_size)
120 !
130 ALLOCATE INTEGER Setup(1:Set_size)
140 ENTER 707 USING "#,B";Setup(*)
150 ENTER 707 USING "#,A";Eof$
160 !
170 OUTPUT 707 USING "#,-K";Header$
180 OUTPUT 707 USING "#,B";Setup(*)
190 OUTPUT 707 USING "#,A";Eof$
200 END
```

Turn Headers Off when Returning Values to Numeric Variables

Turn headers off when returning values to numeric variables. Headers are always off for all common command queries because headers are not defined in the IEEE 488.2 standard.

LOCK

Command

```
:SYSTem:LOCK {{ON | 1} | {OFF | 0}}
```

The :SYSTem:LOCK ON command disables the front panel. The front panel can be re-enabled by sending the :SYSTem:LOCK OFF command or by using the mouse to click on the Minimize button in the upper right-hand corner of the oscilloscope screen.

Example

This example disables the oscilloscope's front panel.

```
10  OUTPUT 707;":SYSTEM:LOCK ON"
20  END
```

Query

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the state of the :SYSTem:LOCK command.

Returned Format

```
[:SYSTem:LOCK] {1 | 0}<NL>
```

LONGform

Command

```
:SYSTem:LONGform {{ON | 1} | {OFF | 0}}
```

The :SYSTem:LONGform command specifies the format for query responses. If the LONGform is set to OFF, command headers and alpha arguments are sent from the oscilloscope in the short form (abbreviated spelling). If LONGform is set to ON, the whole word is output.

Example

This example sets the format for query responses from the oscilloscope to the short form (abbreviated spelling).

```
10  OUTPUT 707;":SYSTEM:LONGFORM OFF"
20  END
```

Query

```
:SYSTem:LONGform?
```

The :SYSTem:LONGform? query returns the current state of the :SYSTem:LONGform command.

Returned Format

```
[:SYSTem:LONGform] {1 | 0}<NL>
```

Example

This example checks the current format for query responses from the oscilloscope, and places the result in the string variable, Result$. Then, it prints the contents of the variable to the computer's screen.

```
10  DIM Result$[50]!Dimension variable
20  OUTPUT 707;":SYSTEM:LONGFORM?"
30  ENTER 707;Result$
40  PRINT Result$
50  END
```

LONGform Does Not Affect Input Headers and Arguments

LONGform has no effect on input headers and arguments sent to the instrument. You may send headers and arguments to the oscilloscope in either the long form or short form, regardless of the current state of the :SYSTem:LONGform command.

PRESet

Command

```
:SYSTem:PRESet [ {DEFault|FACtory} ]
```

The :SYSTem:PRESet command performs a Default Setup just like the oscilloscope's Default Setup key. Using this command does not change any of the control settings found in the User Preferences dialog box, display color settings, screen options, probe skew, probe external adapter settings for differential probes, or probe internal attenuation and gain settings for differential probes. The parameters are optional. A default reset will occur if no parameters are used or the DEFault parameter is used. A factory default occurs with the FACtory parameter.

Example

This example performs an oscilloscope default setup.

```
10  OUTPUT 707;":SYSTEM:PRESet"
20  END
```

SETup

Command

```
:SYSTem:SETup <binary_block_data>
```

The :SYSTem:SETup command sets up the oscilloscope as defined by the data in the binary block of data from the computer.

`<binary_block_data>` A binary block of data, consisting of bytes of setup information. The number of bytes is a dynamic number that is read and allocated by oscilloscope's software.

Example

This example sets up the instrument as defined by the setup data stored in the variable, Set$.

```
10  OUTPUT 707 USING "#,-K";":SYSTEM:SETUP ";Set$
20  END
```

Query

```
:SYSTem:SETup?
```

The :SYSTem:SETup? query outputs the oscilloscope's current setup to the computer in binary block data format as defined in the IEEE 488.2 standard.

Returned Format

```
[:SYSTem:SETup] #NX...X<setup_data_string><NL>
```

The first character in the setup data block is a number added for disk operations.

Example

This example stores the current oscilloscope setup in the string variable, Set$.

```
10  DIM Set$[15000]!Dimension variable
20  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
30  OUTPUT 707;":SYSTEM:SETUP?"
40  ENTER 707 USING "-K";Set$
50  END
```

HP BASIC Image Specifiers

–K is an HP BASIC image specifier which places the block data in a string, including carriage returns and line feeds, until EOI is true, or the dimensioned length of the string is reached.

:SYSTem:SETup Can Operate Just Like *LRN?

When headers and LONGform are on, the :SYSTem:SETup? query operates the same as the *LRN? query in the common commands. Otherwise, *LRN? and :SYSTem:SETup are not interchangeable.

TIME

Command `:SYSTem:TIME <hour>,<minute>,<second>`

The :SYSTem:TIME command sets the time in the oscilloscope and is not affected by the *RST common command.

`<hour>` 0...23

`<minute>` 0...59

`<second>` 0...59

Example This example sets the oscilloscope time to 10:30:45 a.m.

```
10  OUTPUT 707;":SYSTEM:TIME 10,30,45"
20  END
```

Query `:SYSTem:TIME?`

The :SYSTem:TIME? query returns the current time in the oscilloscope.

Returned Format `[:SYSTem:TIME] <hour>,<minute>,<second>`

26

Time Base Commands

Time Base Commands

The TIMebase subsystem commands control the horizontal (X axis) oscilloscope functions. These TIMebase commands and queries are implemented in the Infiniium Oscilloscopes:

- POSition
- RANGe
- REFClock
- REFerence
- SCALe
- VIEW
- WINDow:DELay
- WINDow:POSition
- WINDow:RANGe
- WINDow:SCALe

POSition

Command `:TIMebase:POSition <position_value>`

The :TIMebase:POSition command sets the time interval between the trigger event and the delay reference point. The delay reference point is set with the :TIMebase:REFerence command.

`<position_value>` A real number for the time in seconds from trigger to the delay reference point.

Example This example sets the delay position to 2 ms.

```
10  OUTPUT 707;":TIMEBASE:POSITION 2E-3"
20  END
```

Query `:TIMebase:POSition?`

The :TIMebase:POSition? query returns the current delay value in seconds.

Returned Format `[:TIMebase:POSition] <position_value><NL>`

Example This example places the current delay value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

RANGe

Command

```
:TIMebase:RANGe <full_scale_range>
```

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds. The range value is ten times the time-per-division value.

`<full_scale_range>` A real number for the horizontal time, in seconds. The timebase range is 50 ps (5 ps/div) to 200 s (20 s/div).

Example

This example sets the full-scale horizontal range to 10 ms.

```
10  OUTPUT 707;":TIMEBASE:RANGE 10E-3"
20  END
```

Query

```
:TIMebase:RANGe?
```

The :TIMebase:RANGe? query returns the current full-scale horizontal time.

Returned Format

```
[:TIMebase:RANGe] <full_scale_range><NL>
```

Example

This example places the current full-scale horizontal range value in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:RANGE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

REFClock

Command

```
:TIMebase:REFClock {{ON | 1} | {OFF | 0}}
```

The :TIMebase:REFClock command enables or disables the 10 MHz REF IN BNC input located on the rear panel of the oscilloscope. The 10 MHz reference input is used as a reference clock for the oscilloscope for the horizontal scale section instead of the internal 10 MHz reference when this feature is enabled.

Example

This example turns on the 10 MHz reference mode.

```
10  OUTPUT 707;":TIMEBASE:REFCLOCK ON"
20  END
```

Query

```
:TIMebase:REFClock?
```

The :TIMebase:REFClock? query returns the current state of the 10 MHz reference mode control.

Returned Format

```
[TIMEBASE:REFCLOCK] {1 | 0}<NL>
```

Example

This example places the current value of the 10 MHz reference mode control in the variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:REFCLOCK?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

REFerence

Command

```
:TIMebase:REFerence {LEFT | CENTer | RIGHt}
```

The :TIMebase:REFerence command sets the delay reference to the left, center, or right side of the display.

Example

This example sets the delay reference to the center of the display.

```
10  OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
20  END
```

Query

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current delay reference position.

Returned Format

```
[:TIMebase:REFerence] {LEFT | CENTer | RIGHt}<NL>
```

Example

This example places the current delay reference position in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":TIMEBASE:REFERENCE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

SCALe

Command `:TIMebase:SCALe <time>`

The :TIMebase:SCALe command sets the time base scale. This corresponds to the horizontal scale value displayed as time/div on the oscilloscope screen.

`<time>` A real number for the time value, in seconds per division.
The timebase scale is 5 ps/div to 20 s/div.

Example This example sets the scale to 10 ms/div.

```
10  OUTPUT 707;":TIMEBASE:SCALE 10E-3"
20  END
```

Query `:TIMebase:SCALe?`

The :TIMebase:SCALe? query returns the current scale time setting.

Returned Format `[:TIMebase:SCALe] <time><NL>`

Example This example places the current scale value in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:SCALE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

VIEW

Command

```
:TIMebase:VIEW {MAIN | WINDow}
```

The :TIMebase:VIEW command turns the delayed displayed view on and off. This is the same as using the front panel Delayed button.

Example

This example turns the delayed view on.

```
10  OUTPUT 707;":TIMEBASE:VIEW WINDOW"
20  END
```

Query

```
:TIMebase:VIEW?
```

The :TIMebase:VIEW? query returns Infiniium's current view.

Returned Format

```
[:TIMebase:VIEW] {MAIN | WINDow}<NL>
```

Example

This example places the current view in the string variable, State$, then prints the contents of the variable to the computer's screen.

```
10  DIM State$[50]!Dimension variable
20  OUTPUT 707;":TIMEBASE:VIEW?"
30  ENTER 707;State$
40  PRINT State$
50  END
```

WINDow:DELay

Command

```
:TIMebase:WINDow:DELay <delay_value>
```

The :TIMebase:WINDow:DELay sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

This Command is Provided for Compatibility

This command is the same as the :TIMebase:WINDow:POSition command, and is provided for compatibility with programs written for previous oscilloscopes. The preferred command for compatibility with Infiniium Oscilloscopes is :TIMebase:WINDow:POSition.

`<delay_value>` A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

Example

This example sets the time base window delay position to 20 ns.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:DELAY 20E-9"
20  END
```

Query

```
:TIMebase:WINDow:DELay?
```

The :TIMebase:WINDow:DELay? query returns the current horizontal position in the delayed view.

Returned Format

```
[:TIMebase:WINDow:DELay] <delay_position><NL>
```

Example

This example places the current horizontal position in the delayed view in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:WINDOW:DELAY?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

See Also

The :TIMebase:WINDow:POSition command performs the same function as this command and should be used in new programs.

WINDow:POSition

Command

```
:TIMebase:WINDow:POSition <position_value>
```

The :TIMebase:WINDow:POSition sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

<position_value> A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

Example

This example sets the time base window delay position to 20 ns.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:POSITION 20E-9"
20  END
```

Query

```
:TIMebase:WINDow:POSition?
```

The :TIMebase:WINDow:POSition? query returns the current horizontal position in the delayed view.

Returned Format

```
[:TIMebase:WINDow:POSition] <position_value><NL>
```

Example

This example places the current horizontal position in the delayed view in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:WINDOW:POSITION?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

WINDow:RANGe

Command

```
:TIMebase:WINDow:RANGe <full_scale_range>
```

The :TIMebase:WINDow:RANGe command sets the full-scale range of the delayed view. The range value is ten times the time per division of the delayed view. The maximum range of the delayed view is the current main range. The minimum delayed view range is 10 ps (1 ps/div).

`<full_scale_range>` A real number for the full-scale range of the time base window, in seconds.

Example

This example sets the full-scale range of the delayed view to 100 ns.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:RANGE 100E-9"
20  END
```

Query

```
:TIMebase:WINDow:RANGe?
```

The :TIMebase:WINDow:RANGe? query returns the current full-scale range of the delayed view.

Returned Format

```
[:TIMebase:WINDow:RANGe] <full_scale_range><NL>
```

Example

This example reads the current full-scale range of the delayed view into the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:WINDOW:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

WINDow:SCALe

Command `:TIMebase:WINDow:SCALe <time>`

The :TIMebase:WINDow:SCALe command sets the time/div in the delayed view. This command rescales the horizontal components of displayed waveforms.

`<time>` A real number for the delayed windows scale.

Example This example sets the scale of the time base window to 2 milliseconds/div.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:SCALE 2E-3"
20  END
```

Query `:TIMebase:WINDow:SCALe?`

The :TIMebase:WINDow:SCALe? query returns the scaled window time, in seconds/div.

Returned Format `[:TIMebase:WINDow:SCALe] <time><NL>`

Trigger Commands

Trigger Commands

The oscilloscope trigger circuitry helps you locate the waveform you want to view. There are several different types of triggering, but the one that is used most often is edge triggering. Edge triggering identifies a trigger condition by looking for the slope (rising or falling) and voltage level (trigger level) on the source you select. Any input channel, auxiliary input trigger, or line can be used as the trigger source.

The commands in the TRIGger subsystem define the conditions for triggering. Many of the commands in the TRIGger subsystem are used in more than one of the trigger modes. The command set has been defined to closely represent the front-panel trigger menus. As a trade-off, there may be less compatibility between Infiniium Oscilloscopes and command sets for previous oscilloscopes. Infiniium Oscilloscopes still accept some commands for compatibility with previous instruments. An alternative command that is accepted by the oscilloscope is noted for a particular command.

These TRIGger commands and queries are implemented in the Infiniium Oscilloscopes:

- :AND:{ENABle | SOURce}
- :COMM:{BWIDth | ENCode | PATTern | POLarity | SOURce}
- :DELay:{ARM | EDELay | MODE | TDELay | TRIGger}
- :EDGe:{SLOPe | SOURce_EITHer}
- :GLITch:{POLarity | SOURce | WIDTh}
- :HOLDoff
- :HTHReshold
- :HYSTeresis
- :LEVel
- :LTHReshold
- :MODE
- :PATTern:{CONDition | LOGic}
- :PWIDth:{DIRection | POLarity | SOURce | TPOint | WIDTh}
- :RUNT:{POLarity | QUALified | SOURce | TIME}
- SEQuence:{RESet | TERM1 | TERM2 | WAIT | EXTend}
- :SHOLd:{CSOurce | DSOurce | HoldTIMe | MODE | SetupTIMe}
- :STATE:{CLOCk | LOGic | LTYPe | SLOPe}

- :SWEep
- :TIMeout:{CONDition | SOURce | TIME}
- :TRANsition:{DIRection | SOURce | TIME | TYPE}
- :TV:{MODE | SOURce | STANdard | UDTV}
- :WINDow:{CONDition | SOURce | TIME | TPOint}

The trigger modes are summarized in the next section. In addition, each mode is described before its set of commands in the following sections.

These general trigger commands are described first.

- AND
- HOLDoff
- HTHReshold
- HYSTeresis
- LTHReshold
- LEVel
- MODE
- SWEep

The following sections in this chapter describe the individual trigger modes and commands, and are organized in this order:

- COMM
- DELay
- EDGE
- GLITch
- PATTern
- PWIDth
- RUNT
- SEQuence
- SHOLd
- STATe
- TIMeout
- TRANsition
- TV
- WINDow

Make sure the oscilloscope is in the proper trigger mode for the command you want to send. One method of ensuring that the oscilloscope is in the proper trigger mode is to send the :TRIGger:MODE command in the same program message as the parameter to be set.

For example, these commands place the instrument in the proper triggering mode you select:

```
:TRIGger:MODE <Trigger_mode>
```

`<Trigger_mode>` The trigger modes include COMM, DELay, EDGE, GLITch, PATTern, PWIDth, RUNT, SEQuence, SHOLd, STATe, TIMeout, TRANsition, TV, and WINDow. Each mode is described with its command set in this chapter.

AND:ENABle

Command

```
:TRIGger:AND[{1 | 2}]:ENABle {{ON | 1} | {OFF | 0}}
```

The :TRIGger:AND:ENABle command enables the ability to further qualify the trigger using other channels.

The optional [{1|2}] parameter sets whether the AND qualifier goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:AND:ENABle?
```

The query returns the current state of the AND qualifier.

Returned Format

```
[:TRIGger:AND:ENABle] {1 | 0}<NL>
```

AND:SOURce

Command

```
:TRIGger:AND[{1 | 2}]:SOURce CHANnel<N>,{HIGH | LOW | DONTcare}
```

The :TRIGger:AND:SOURce command sets the logic value used to qualify the trigger for the specified channel. The TRIGger:LEVel command determines what voltage level is considered a HIGH or a LOW logic value. If you set more than one channel to a HIGH or a LOW, then the multiple channels are used to qualify the trigger.

The optional [{1|2}] parameter sets whether the AND qualifier goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query

```
:TRIGger:AND:SOURce? CHANnel<N>
```

The query returns the logic value for the designated channel.

Returned Format

```
[:TRIGger:AND:SOURce CHANnel<N>] {HIGH | LOW | DONTcare}<NL>
```

COMM:BWIDth

Command `:TRIGger:COMM:BWIDth <bwidth_value>`

The :TRIGger:COMM:BWIDth command is used to set the width of a bit for your waveform. The bit width is usually defined in the mask standard for your waveform.

`<bwidth_value>` A real number that represents the width of a bit.

Query `:TRIGger:COMM:BWIDth?`

The query returns the current bit width.

Returned Format `[:TRIGger:COMM:BWIDth] <bwidth_value><NL>`

COMM:ENCode

Command `:TRIGger:COMM:ENCode {RZ | NRZ}`

This :TRIGger:COMM:ENCode command sets the type of waveform encoding for your waveform. You should use NRZ for CMI type waveforms and RZ for all other type of waveforms.

Query `:TRIGger:COMM:ENCode?`

The :TRIGger:COMM:ENCode? query returns the current value of encoding

Returned Format `[:TRIGger:COMM:ENCode] {RZ | NRZ}<NL>`

COMM:PATTern

Command `:TRIGger:COMM:PATTern <bit>[,<bit[,<bit[,<bit[,<bit[,<bit]]]]]`

The :TRIGger:COMM:PATTern command sets the pattern used for triggering the oscilloscope when in communication trigger mode. The pattern can be up to 6 bits long. For NRZ type waveforms with positive polarity, there must be at least one logic 0 to logic 1 transition in the pattern. For NRZ waveforms with negative polarity there must be at least one logic 1 to logic 0 transition in the pattern. For RZ type waveforms the pattern must have at least one logic 1 bit for positive polarity. For RZ type waveforms the pattern must have at least one logic -1 bit for negative polarity.

`<bit>` A 1, -1, or 0.

Query `:TRIGger:COMM:PATTern?`

The `:TRIGger:COMM:PATTern?` query returns the current communication trigger pattern.

Returned Format `[:TRIGger:COMM:PATTern] <pattern><NL>`

`<pattern>` A string of up to 6 characters.

COMM:POLarity

Command

```
:TRIGger:COMM:POLarity {POSitive | NEGative}
```

The `:TRIGger:COMM:POLarity` command directly controls the trigger slope used for communication trigger. When set to a positive value, the rising edge of a pulse or waveform is used to trigger the oscilloscope. When set to a negative value, the falling edge of a pulse or waveform is used.

The polarity setting is also used to check for valid patterns. If you are trying to trigger on an isolated 1 pattern, you should set the polarity to positive. If you are trying to trigger on an isolated -1 pattern, you should set the polarity to negative.

Query

```
:TRIGger:COMM:POLarity?
```

The `:TRIGger:COMM:POLarity?` query returns the current setting for polarity.

Returned Format

```
[:TRIGger:COMM:POLarity} {POSitive | NEGative}<NL>
```

COMM:SOURce

Command `:TRIGger:COMM:SOURce CHANnel<N>`

The :TRIGger:COMM:SOURce command selects the channel used for the communication trigger.

`<N>` An integer, 1-4.

Query `:TRIGger:COMM:SOURce?`

The `:TRIGger:COMM:SOURce?` query returns the currently selected communication trigger source.

Returned Format `[:TRIGger:COMM:SOURce] CHANnel<N><NL>`

DELay:ARM:SOURce

Command `:TRIGger:DELay:ARM:SOURce CHANnel<N>`

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay trigger mode.

`<N>` An integer, 1 - 4.

Query `:TRIGger:DELay:ARM:SOURce?`

The query returns the currently defined Arm On source for the Delay trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:ARM:SOURce] CHANnel<N><NL>`

DELay:ARM:SLOPe

Command

```
:TRIGger:DELay:ARM:SLOPe {NEGative | POSitive}
```

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay trigger mode.

Query

```
:TRIGger:DELay:ARM:SLOPe?
```

The query returns the currently defined slope for the Delay trigger mode.

Returned Format

```
[:TRIGger:DELay:ARM:SLOPe] {NEGative | POSitive}<NL>
```

DELay:EDELay:COUNt

Command `:TRIGger:DELay:EDELay:COUNt <edge_number>`

This command sets the event count for a Delay By Event trigger event.

`<edge_num>` An integer from 0 to 16,000,000 specifying the number of edges to delay.

Query `:TRIGger:DELay:EDELay:COUNt?`

The query returns the currently defined number of events to delay before triggering on the next Trigger On condition in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:COUNt] <edge_number><NL>`

DELay:EDELay:SOURce

Command `:TRIGger:DELay:EDELay:SOURce CHANnel<N>`

This command sets the Event source for a Delay By Event trigger event.

`<N>` An integer, 1 - 4.

Query `:TRIGger:DELay:EDELay:SOURce?`

The query returns the currently defined Event source in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:SOURce] CHANnel<N><NL>`

DELay:EDELay:SLOPe

Command `:TRIGger:DELay:EDELay:SLOPe {NEGative | POSitive}`

This command sets the trigger slope for the Delay By Event trigger event.

Query `:TRIGger:DELay:EDELay:SLOPe?`

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:SLOPe] {NEGative | POSitive}<NL>`

DELay:MODE

Command

```
:TRIGger:DELay:MODE {EDELay | TDELay}
```

The :TRIGger:DELay:MODE command selects the type of delay trigger mode to either events or to time.

Query

```
:TRIGger:DELay:MODE?
```

The query returns the currently selected delay trigger mode.

Returned Format

```
[:TRIGger:DELay:MODE] {EDELay | TDELay}<NL>
```

DELay:TDELay:TIME

Command `:TRIGger:DELay:TDELay:TIME <delay>`

This command sets the delay for a Delay By Time trigger event.

`<delay>` Time, in seconds, set for the delay trigger, from 10 ns to 10 s.

Query `:TRIGger:DELay:TDELay:TIME?`

The query returns the currently defined time delay before triggering on the next Trigger On condition in the Delay By Time trigger mode.

Returned Format `[:TRIGger:DELay:TDELay:TIME] <delay><NL>`

DELay:TRIGger:SOURce

Command `:TRIGger:DELay:TRIGger:SOURce CHANnel<N>`

This command sets the Trigger On source for a Delay trigger event.

`<N>` An integer, 1 - 4.

Query `:TRIGger:DELay:TRIGger:SOURce?`

The query returns the currently defined Trigger On source in the Delay trigger mode.

Returned Format `[:TRIGger:DELay:TRIGger:SOURce] CHANnel<N><NL>`

DELay:TRIGger:SLOPe

Command `:TRIGger:DELay:TRIGger:SLOPe {NEGative | POSitive}`

This command sets the trigger slope for the Delay trigger event.

Query `:TRIGger:DELay:TRIGger:SLOPe?`

The query returns the currently defined slope for an event in the Delay trigger mode.

Returned Format `[:TRIGger:DELay:TRIGger:SLOPe] {NEGative | POSitive}<NL>`

EDGE:SLOPe

Command

```
:TRIGger:EDGE[{1 | 2}]:SLOPe {POSitive | NEGative | EITHer}
```

The :TRIGger:EDGE:SLOPe command sets the slope of the trigger source previously selected by the :TRIGger:EDGE:SOURce command. The LINE source has no slope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:EDGE:SLOPe?
```

The query returns the currently selected slope for the specified edge trigger source.

Returned Format

```
[:TRIGger:EDGE:SLOPe] {POSitive | NEGative | EITHer}<NL>
```

EDGE:SOURce

Command

```
:TRIGger:EDGE[{1 | 2}]:SOURce {CHANnel<N> | AUX | LINE}
```

The :TRIGger:EDGE:SOURce command selects the source for edge mode triggering. This is the source that will be used for subsequent :TRIGger:EDGE:SLOPe commands or queries.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query

```
:TRIGger:EDGE:SOURce?
```

The query returns the currently selected edge mode trigger source.

Returned Format

```
[:TRIGger:EDGE:SOURce] {CHANnel<N> | AUX | LINE}<NL>
```

GLITch:POLarity

Command

```
:TRIGger:GLITch[{1 | 2}]:POLarity {POSitive | NEGative}
```

This command defines the polarity of the glitch as positive or negative. The trigger source must be set using the :TRIGger:GLITch:SOURce command.
The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:GLITch:POLarity?
```

The query returns the currently selected glitch polarity.

Returned Format

```
[:TRIGger:GLITch:POLarity] {POSitive | NEGative}<NL>
```

GLITch:SOURce

Command `:TRIGger:GLITch[{1 | 2}]:SOURce CHANnel<N>`

This command sets the source for the glitch trigger mode.

The optional [{1|2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query `:TRIGger:GLITch:SOURce?`

The query returns the currently selected source for the glitch trigger mode.

Returned Format `[:TRIGger:GLITch:SOURce] CHANnel<N><NL>`

GLITch:WIDTh

Command `:TRIGger:GLITch[{1 | 2}]:WIDTh <width>`

This command sets the glitch width. The oscilloscope will trigger on a pulse that has a width less than the specified width

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<width>` A real number for the glitch width, ranging from 250 ps to 10 s.

Query `:TRIGger:GLITch:WIDTh?`

The query returns the currently specified glitch width.

Returned Format `[:TRIGger:GLITch:WIDTh] <width><NL>`

HOLDoff

Command `:TRIGger:HOLDoff <holdoff_time>`

The :TRIGger:HOLDoff command specifies the amount of time the oscilloscope should wait after receiving a trigger before enabling the trigger again.

`<holdoff_time>` A real number for the holdoff time, ranging from 80 ns to 10 s.

Query `:TRIGger:HOLDoff?`

The query returns the current holdoff value for the current mode.

Returned Format `[:TRIGger:HOLDoff] <holdoff><NL>`

HTHReshold

Command `:TRIGger:HTHReshold CHANnel<N>,<level>`

This command specifies the high threshold voltage level for the selected trigger source. Set the high threshold level to a value considered to be a high level for your logic family; your data book gives two values, V_{IH} and V_{OH}.

`<N>` An integer, 1 - 4.

`<level>` A real number for the voltage level for the trigger source.

Query `:TRIGger:HTHReshold? CHANnel<N>`

The query returns the currently defined high threshold voltage level for the trigger source.

Returned Format `[:TRIGger:HTHReshold CHANnel<N>,] <level><NL>`

HYSTeresis

Command

```
:TRIGger:HYSTeresis {NORMal | HSENsitivity}
```

The :TRIGger:HYSTeresis command specifies the trigger hysteresis (noise reject) as either normal or high sensitivity. NORMal sensitivity adds hysteresis to the trigger circuitry for rejecting noise and should be used for waveforms of 4 GHz or below. HSENsitivity lowers the hysteresis of the trigger circuitry and should be used for waveforms of 4 GHz and above.

Query

```
:TRIGger:HYSTeresis?
```

The query returns the current hysteresis setting.

Returned Format

```
[:TRIGger:HYSTeresis] {NORMal | HSENsitivity}<NL>
```

LEVel

Command `:TRIGger:LEVel {{CHANnel<N> | AUX},<level>}}`

The :TRIGger:LEVel command specifies the trigger level on the specified channel for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. This level applies to the channel throughout the trigger dialog.

`<N>` An integer, 1 - 4.

`<level>` A real number for the trigger level on the specified channel or Auxiliary Trigger Input.

Query `:TRIGger:LEVel? {CHANnel<N> | AUX}`

The query returns the specified channel's trigger level.

Returned Format `[:TRIGger:LEVel {CHANnel<N> | AUX},] <level><NL>`

LTHReshold

Command `:TRIGger:LTHReshold CHANnel<N>,<level>`

This command specifies the low threshold voltage level for the selected trigger source. This command specifies the low threshold voltage level for the selected trigger source. Set the low threshold level to a value considered to be a low level for your logic family; your data book gives two values, V_{IL} and V_{OL}.

`<N>` An integer, 1 - 4.

`<level>` A real number for the voltage level for the trigger source.

Query `:TRIGger:LTHReshold? CHANnel<N>`

The query returns the currently defined low threshold for the trigger source.

Returned Format `[:TRIGger:LTHReshold CHANnel<N>,] <level><NL>`

MODE

Command

```
:TRIGger:MODE {COMM | DELay | EDGE | GLITch |
PATTern | PWIDth | RUNT | SEQuence | SHOLd | STATe
| TIMeout | TRANsition | TV | WINDow}
```

The :TRIGger:MODE command selects the trigger mode.

Table 27-1 Trigger Mode Settings

Mode	Definition
COMM	COMM mode lets you trigger on a serial pattern of bits in a waveform.
DELay	Delay by Events mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. Delay by Time mode lets you view pulses in your waveform that occur a long time after a specified waveform edge.
EDGE	Edge trigger mode.
GLITch	Trigger on a pulse that has a width less than a specified amount of time.
PATTern	Pattern triggering lets you trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width.
PWIDth	PWIDth or pulse width trigger is used to find pulse that are either narrower or wider than pulse width and conditions that you specify.
RUNT	Runt trigger allows you to find pulses in your waveform that are shorter in amplitude than other pulses in your waveform.
SEQuence	Sequential triggering allows you to use multiple events or time/pattern qualifications to define your trigger.
SHOLd	SHOLd or setup and hold time trigger lets you find violations of setup and hold times in your waveform.
STATe	State triggering lets you set the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform.
TIMeout	Timeout trigger lets you find pulses that are high, low, or idle for too long a period of time.
TRANsition	Transition trigger lets you find pulses that have edges that are too slow or too fast in your waveform.
TV	TV trigger mode lets you trigger the oscilloscope on one of the standard television waveforms. You can also use this mode to trigger on a custom television waveform that you define.

WINDow	**Window trigger lets you set a range of voltages (window) which can be used to determine when your waveform is inside or out side of the window.**

Query `:TRIGger:MODE?`

The query returns the currently selected trigger mode.

Returned Format `[:TRIGger:MODE] {COMM | DELay | EDGE | GLITch | PATTern | PWIDth | RUNT | SEQuence | SHOLd | STATe | TIMeout | TRANsition | TV | WINDow}<NL>`

PATTern:CONDition

Command

```
:TRIGger:PATTern[{1 | 2}]:CONDition {ENTered |
EXITed | {GT,<time>[,PEXits|TIMeout]} |
{LT,<time>} | {RANGe,<gt_time>,<lt_time>}}
```

This command describes the condition applied to the trigger pattern to actually generate a trigger.

The optional [{1|2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<gt_time>` The minimum time (greater than time) for the trigger pattern, from 10 ns to 9.999 s.

`<lt_time>` The maximum time (less than time) for the trigger pattern, from 10.5 ps to 10 s.

`<time>` The time condition, in seconds, for the pattern trigger, from 100 ps to 10 s.

When using the GT (Present >) parameter, the PEXits (Pattern Exits) or the TIMeout parameter controls when the trigger is generated.

Query

```
:TRIGger:PATTern:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:PATTern:CONDition] {ENTered|EXITed |
{GT,<time>[,PEXits|TIMeout]} | {LT,<time>} |
{RANGe,<gt_time>, <lt_time>}}<NL>
```

PATTern:LOGic

Command

```
:TRIGger:PATTern[{1 | 2}]:LOGic CHANnel<N>},{HIGH | LOW | DONTcare | RISing | FALLing}
```

This command defines the logic criteria for a selected channel.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

<Query

```
:TRIGger:PATTern:LOGic? CHANnel<N>}
```

The query returns the current logic criteria for a selected channel.

Returned Format

```
[:TRIGger:PATTern:LOGic CHANnel<N>,] {HIGH | LOW | DONTcare | RISing | FALLing}<NL>
```

PWIDth:DIRection

Command

```
:TRIGger:PWIDth[{1 | 2}]:DIRection {GTHan | LTHan}
```

This command specifies whether a pulse must be wider or narrower than the width value to trigger the oscilloscope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:PWIDth:DIRection?
```

The query returns the currently defined direction for the pulse width trigger.

Returned Format

```
[:TRIGger:PWIDth:DIRection] {GTHan | LTHan}<NL>
```

PWIDth:POLarity

Command

```
:TRIGger:PWIDth[{1 | 2}]:POLarity {NEGative | POSitive}
```

This command specifies the pulse polarity that the oscilloscope uses to determine a pulse width violation. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:PWIDth:POLarity?
```

The query returns the currently defined polarity for the pulse width trigger.

Returned Format

```
[:TRIGger:PWIDth:POLarity] {NEGative | POSitive}<NL>
```

PWIDth:SOURce

Command `:TRIGger:PWIDth[{1 | 2}]:SOURce CHANnel<N>`

This command specifies the channel source used to trigger the oscilloscope with the pulse width trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<N>` An integer, 1 - 4.

`<level>` A real number for the voltage through which the pulse must pass before the oscilloscope will trigger.

Query `:TRIGger:PWIDth:SOURce?`

The query returns the currently defined channel source for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:SOURce] CHANnel<N><NL>`

PWIDth:TPOint

Command

```
:TRIGger:PWIDth[{1 | 2}]:TPOint {EPULse | TIMeout}
```

This command specifies whether the pulse width trigger should occur at the end of the pulse or at a specified timeout period. This command is only available if the pulse direction is set to GTHan.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:PWIDth:TPOint?
```

The query returns the currently defined trigger on point for the pulse width trigger.

Returned Format

```
[:TRIGger:PWIDth:TPOint] {EPULse | TIMeout}<NL>
```

PWIDth:WIDTh

Command `:TRIGger:PWIDth[{1 | 2}]:WIDTh <width>`

This command specifies how wide a pulse must be to trigger the oscilloscope.

The optional [{1|2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<width>` Pulse width, which can range from 250 ps to 10 s.

Query `:TRIGger:PWIDth:WIDTh?`

The query returns the currently defined width for the pulse.

Returned Format `[:TRIGger:PWIDth:WIDTh] <width><NL>`

RUNT:POLarity

Command

```
:TRIGger:RUNT[{1 | 2}]:POLarity {POSitive | NEGative}
```

This command defines the polarity of the runt pulse as positive or negative. The trigger source must be set using the :TRIGger:RUNT:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:RUNT:POLarity?
```

The query returns the currently selected runt pulse polarity.

Returned Format

```
[:TRIGger:RUNT:POLarity] {POSitive | NEGative}<NL>
```

RUNT:QUALified

Command

```
:TRIGger:RUNT[{1 | 2}]:QUALified {{ON | 1} | {OFF | 0}}
```

This command enables the time qualified runt pulse feature the polarity of the runt pulse as positive or negative. The trigger source must be set using the :TRIGger:RUNT:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:RUNT:QUALified?
```

The query returns the current state of the time qualified runt pulse feature.

Returned Format

```
[:TRIGger:RUNT:QUALified] {1 | 0}<NL>
```

RUNT:SOURce

Command `:TRIGger:RUNT[{1 | 2}]:SOURce CHANnel<N>`

This command sets the source for the runt trigger mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query `:TRIGger:RUNT:SOURce?`

The query returns the currently selected source for the runt trigger mode.

Returned Format `[:TRIGger:RUNT:SOURce] CHANnel<N><NL>`

RUNT:TIME

Command `:TRIGger:RUNT[{1 | 2}]:TIME <time>`

This command sets the time qualifier. The oscilloscope will trigger on a runt pulse that has a width greater than the specified time.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<time>` A real number for the time greater than qualifier, ranging from 250 ps to 30 ns.

Query `:TRIGger:RUNT:TIME?`

The query returns the currently specified glitch width.

Returned Format `[:TRIGger:RUNT:TIME] <time><NL>`

SEQuence:TERM1

Command

```
:TRIGger:SEQuence:TERM1 { EDGE1 | GLITch1 | PWIDth1
| RUNT1 | SHOLD1 | STATe1 | TIMeout1 | TRANsition1
| WINDow1 }
```

This command specifies the trigger mode for the TERM1 state in the sequential trigger (the Find (A) state in the Trigger Setup dialog box on the oscilloscope).

There are five limitations associated with sequential triggering:

1) The Edge followed by Edge and Video trigger modes cannot be used in sequential triggering.

2) The AND qualifier cannot be used when the Reset condition is based upon a logical pattern.

3) The Pattern/State trigger mode that uses range as the When Pattern selection can only be used for either the Term1 state or the Term2 state, but not both.

4) You can only use one long timer (>30 ns). Therefore, trigger modes that use timers greater than 30 ns can only be used for either the Term1 state or the Term2 state, but not both. Some examples of trigger modes where you might use a long timer include Pulse Width, Glitch, Window, Edge Transition, and Timeout.

5) The alternating edge trigger mode cannot be used in sequential triggering.

Limitations (3) and (4) deal with extended resources. Extended resources refer to trigger modes or conditions that are only available to either the Term1 state or the Term2 state, but not both at the same time. The oscilloscope will figure out which state has access to these extended resources based upon the conditions you setup in each of these states. If you want Term2 to have a timer longer than 30 ns, you must first change the timer associated with Term1 to be less than 30 ns.

Query

```
:TRIGger:SEQuence:TERM1?
```

The query returns the currently defined trigger mode for the TERM1 state.

SEQuence:TERM2

Command

```
:TRIGger:SEQuence:TERM2 { EDGE2 | GLITch2 | PWIDth2
| RUNT2 | SHOLD2 | STATe2 | TIMeout2 | TRANsition2
| WINDow2 }
```

This command specifies the trigger mode for the TERM2 state in the sequential trigger (the Trigger (B) state in the Trigger Setup dialog box on the oscilloscope).

There are five limitations associated with sequential triggering:

1) The Edge followed by Edge and Video trigger modes cannot be used in sequential triggering.

2) The AND qualifier cannot be used when the Reset condition is based upon a logical pattern.

3) The Pattern/State trigger mode that uses range as the When Pattern selection can only be used for either the Term1 state or the Term2 state, but not both.

4) You can only use one long timer (>30 ns). Therefore, trigger modes that use timers greater than 30 ns can only be used for either the Term1 state or the Term2 state, but not both. Some examples of trigger modes where you might use a long timer include Pulse Width, Glitch, Window, Edge Transition, and Timeout.

5) The alternating edge trigger mode cannot be used in sequential triggering.

Limitations (3) and (4) deal with extended resources. Extended resources refer to trigger modes or conditions that are only available to either the Term1 state or the Term2 state, but not both at the same time. The oscilloscope will figure out which state has access to these extended resources based upon the conditions you setup in each of these states. If you want Term2 to have a timer longer than 30 ns, you must first change the timer associated with Term1 to be less than 30 ns.

Query

```
:TRIGger:SEQuence:TERM2?
```

The query returns the currently defined trigger mode for the TERM2 state.

SEQuence:RESet:ENABle

Command

```
:TRIGger:SEQuence:RESet:ENABle {{ON | 1} | {OFF | 0}}
```

This command turns the Reset feature on or off for the sequential trigger.

The Reset feature allows you to specify a length of time such that if this time is exceeded between when the TERM1 event occurs and when the TERM2 event occurs, the sequential trigger is reset and the oscilloscope returns to looking for the TERM1 event without triggering. If the Delay feature (remote command :WAIT) is used as well then the Reset timer does not start counting down until after the delay period is complete.

You can also base the Reset condition on a logical pattern. If the specified pattern is found between when the TERM1 occurs and the TERM2 event occurs, the sequential trigger resets and goes back to looking for the TERM1 event without triggering. The delay feature does not impact a logical pattern Reset as the pattern is searched for immediately after the TERM1 event occurs regardless of whether of not the Delay period is complete.

If the Reset feature is enabled, the AND qualifier cannot be used for the TERM1 state.

Query `:TRIGger:SEQuence:RESet:ENABle?`

The query returns whether or not the Reset feature is enabled.

SEQuence:RESet:TYPE

Command `:TRIGger:SEQuence:RESet:TYPE { TIME | EVENt }`

This command specifies whether the Reset condition is based upon a length of time or a logical pattern.

The Reset feature allows you to specify a length of time such that if this time is exceeded between when the TERM1 event occurs and when the TERM2 event occurs, the sequential trigger is reset and the oscilloscope returns to looking for the TERM1 event without triggering. If the Delay feature (remote command :WAIT) is used as well then the Reset timer does not start counting down until after the delay period is complete.

You can also base the Reset condition on a logical pattern. If the specified pattern is found between when the TERM1 occurs and the TERM2 event occurs, the sequential trigger resets and goes back to looking for the TERM1 event without triggering. The delay feature does not impact a logical pattern Reset as the pattern is searched for immediately after the TERM1 event occurs regardless of whether of not the Delay period is complete.

Query

```
:TRIGger:SEQuence:RESet:TYPE?
```

The query returns whether the Reset condition is based upon a length of time or an event.

SEQuence:RESet:EVENt

Command

```
:TRIGger:SEQuence:RESet:EVENt {CHANnel<N>}, { HIGH
| LOW | DONTcare }
```

This command defines the logical pattern used for an event Reset condition.

You can specify for each channel (1-4) whether you want the value to be high (1), low (0), or you don't care (X).

<N> An integer, 1 - 4.

SEQuence:RESet:TIME

Command `:TRIGger:SEQuence:RESet:TIME <time>`

This command defines the length of time to use for the time-based Reset condition.

`<time>` A length of time in seconds.

Query `:TRIGger:SEQuence:RESet:TIME?`

The query returns the length of time used for the Reset condition.

SEQuence:RESet:WAIT:ENABle

Command `:TRIGger:SEQuence:RESet:WAIT:ENABle { {ON|1} | {OFF|0}}`

This command turns the Delay feature on or off for the sequential trigger.

The Delay feature allows you to define a length of time for the sequential trigger system to wait after the TERM1 event occurs before it starts searching for the TERM2 event.

Query `:TRIGger:SEQuence:RESet:ENABle?`

The query returns whether or not the Delay feature is turned on.

SEQuence:RESet:WAIT:TIME

Command `:TRIGger:SEQuence:RESet:WAIT:TIME <time>`

This command defines the length of time to use for the Delay condition.

`<time>` A length of time in seconds.

Query `:TRIGger:SEQuence:WAIT:TIME?`

The query returns the length of time used for the Delay condition.

SHOLd:CSOurce

Command `:TRIGger:SHOLd[{1 | 2}]:CSOurce CHANnel<N>`

This command specifies the clock source for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query `:TRIGger:SHOLd:CSOurce?`

The query returns the currently defined clock source for the trigger setup and hold violation.

Returned Format `[:TRIGger:SHOLd:CSOurce] CHANnel<N><NL>`

SHOLd:CSOurce:EDGE

Command

```
:TRIGger:SHOLd[{1 | 2}]:CSOurce:EDGE {RISing | FALLing}
```

This command specifies the clock source trigger edge for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:SHOLd:CSOurce:EDGE?
```

The query returns the currently defined clock source edge for the trigger setup and hold violation level for the clock source.

Returned Format

```
[:TRIGger:SHOLd:CSOurce:EDGE] {RISing | FALLing}<NL>
```

SHOLd:DSOurce

Command `:TRIGger:SHOLd[{1 | 2}]:DSOurce CHANnel<N>`

The data source commands specify the data source for the trigger setup and hold violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query `:TRIGger:SHOLd:DSOurce?`

The query returns the currently defined data source for the trigger setup and hold violation.

Returned Format `[:TRIGger:SHOLd:DSOurce] CHANnel<N><NL>`

SHOLd:HoldTIMe (HTIMe)

Command `:TRIGger:SHOLd[{1 | 2}]:HoldTIMe <time>`

This command specifies the amount of hold time used to test for both a setup and hold trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<time>` Hold time, in seconds.

Query `:TRIGger:SHOLD:HoldTIMe?`

The query returns the currently defined hold time for the setup and hold trigger violation.

Returned Format `[:TRIGger:SHOLD:HoldTIMe] <time><NL>`

SHOLd:MODE

Command `:TRIGger:SHOLd[{1 | 2}]:MODE {SETup | HOLD | SHOLd}`

`SETup` When using the setup time mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the trigger level thresholds during this time window. If the waveform crosses a threshold during this time window, a violation event occurs and the oscilloscope triggers.

`HOLD` When using the hold time mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

`SHOLd` When using the setup and hold time mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers. The total time allowed for the sum of setup time plus hold time is 24 ns maximum.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:SHOLd:MODE?`

The query returns the currently selected trigger setup violation mode.

Returned Format `[:TRIGger:SHOLd:MODE] {SETup | HOLD | SHOLd}<NL>`

SHOLd:SetupTIMe

Command `:TRIGger:SHOLd[{1 | 2}]:SetupTIMe <time>`

This command specifies the amount of setup time used to test for both a setup and hold trigger violation. The setup time is the amount of time that the data must be stable and valid before a clock edge.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<time>` Setup time, in seconds.

Query `:TRIGger:SHOLd:SetupTIMe?`

The query returns the currently defined setup time for the setup and hold trigger violation.

Returned Format `[:TRIGger:SHOLd:SetupTIMe] <time><NL>`

STATe:CLOCk

Command `:TRIGger:STATe[{1 | 2}]:CLOCk {CHANnel<N>}`

This command selects the source for the clock waveform in the State Trigger Mode. The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4.

Query `:TRIGger:STATe:CLOCk?`

The query returns the currently selected clock source.

Returned Format `[:TRIGger:STATe:CLOCk] {CHANnel<N>}<NL>`

STATe:LOGic

Command

```
:TRIGger:STATe[{1 | 2}]:LOGic CHANnel<N>,{LOW | HIGH | DONTcare | RISing | FALLing}
```

This command defines the logic state of the specified source for the state pattern. The command produces a settings conflict on a channel that has been defined as the clock.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 - 4

Query

```
:TRIGger:STATe:LOGic? CHANnel<N>
```

The query returns the logic state definition for the specified source.

<N> N is the channel number, an integer in the range of 1 - 4.

Returned Format

```
[:TRIGger:STATe:LOGic CHANnel<N>,] {LOW | HIGH | DONTcare | RISing | FALLing}<NL>
```

STATe:LTYPe

Command

```
:TRIGger:STATe[{1 | 2}]:LTYPe {AND | NAND}
```

This command defines the state trigger logic type. If the logic type is set to AND, then a trigger is generated on the edge of the clock when the input waveforms match the pattern specified by the :TRIGger:STATe:LOGic command. If the logic type is set to NAND, then a trigger is generated on the edge of the clock when the input waveforms do not match the specified pattern.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:STATe:LTYPe?
```

The query returns the currently specified state trigger logic type.

Returned Format

```
[:TRIGger:STATe:LTYPe] {AND | NAND}<NL>
```

STATe:SLOPe

Command

```
:TRIGger:STATe[{1 | 2}]:SLOPe {RISing | FALLing | EITHer}
```

This command specifies the edge of the clock that is used to generate a trigger. The waveform source used for the clock is selected by using the :TRIGger:STATe:CLOCk command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:STATe:SLOPe?
```

The query returns the currently defined slope for the clock in State Trigger Mode.

Returned Format

```
[:TRIGger:STATe:SLOPe] {RISing | FALLing | EITHer}<NL>
```

SWEep

Command `:TRIGger:SWEep {AUTO | TRIGgered | SINGle}`

The :TRIGger:SWEep command selects the oscilloscope sweep mode. New programs should use :RUN and :SINGle for run control and this command for AUTO and TRIGgered for sweep control. The SINGle sweep control should not be used.

`AUTO` When you select AUTO, if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 50 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.

`TRIGgered` When you select TRIGgered, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen.

`SINGle` When you select SINGle, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen. Do not use in new programs.

Query `:TRIGger:SWEep?`

The query returns the specified channel's trigger level.

Returned Format `[:TRIGger:SWEep] {AUTO | TRIGgered}<NL>`

TIMeout:CONDition

Command `:TRIGger:TIMeout[{1 | 2}]:CONDition {HIGH | LOW | UNCHanged}`

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

This command sets the condition used for the timeout trigger.

`HIGH` Trigger when the waveform has been high for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

`LOW` Trigger when the waveform has been low for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

`UNCHanged` Trigger when the waveform has not changed state for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

Query `:TRIGger:TIMeout:CONDition?`

The query returns the currently defined trigger condition for the timeout trigger.

Returned Format `[:TRIGger:TIMeout:CONDition] {HIGH | LOW | UNCHanged}<NL>`

TIMeout:SOURce

Command `:TRIGger:TIMeout[{1 | 2}]:SOURce CHANnel<N>`

This command specifies the channel source used to trigger the oscilloscope with the timeout trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<N>` An integer, 1 - 4.

Query `:TRIGger:TIMeout:SOURce?`

The query returns the currently defined channel source for the timeout trigger.

Returned Format `[:TRIGger:TIMeout:SOURce] CHANnel<N><NL>`

TIMeout:TIME

Command `:TRIGger:TIMeout[{1 | 2}]:TIME <time>`

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1|2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<time>` The time for the timeout trigger, in seconds.

Query `:TRIGger:TIMeout:TIME?`

The query returns the currently defined time for the trigger trigger.

Returned Format `[:TRIGger:TIMeout:TIME] <time><NL>`

TRANsition:DIRection

Command

```
:TRIGger:TRANsition[{1 | 2}]:DIRection {GTHan | LTHan}
```

This command lets you look for transition violations that are greater than or less than the time specified by the :TRIGger:TRANsition:TIME command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:TRANsition:DIRection?
```

The query returns the currently defined direction for the trigger transition violation.

Returned Format

```
[:TRIGger:TRANsition:DIRection] {GTHan | LTHan}]<NL>
```

TRANsition:SOURce

Command `:TRIGger:TRANsition[{1 | 2}]:SOURce CHANnel<N>`

The transition source command lets you find any edge in your waveform that violates a rise time or fall time specification. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<N>` An integer, 1 - 4.

Query `:TRIGger:TRANsition:SOURce?`

The query returns the currently defined transition source for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:SOURce] CHANnel<N><NL>`

TRANsition:TIME

Command `:TRIGger:TRANsition[{1 | 2}]:TIME <time>`

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<time>` The time for the trigger violation transition, in seconds.

Query `:TRIGger:TRANsition:TIME?`

The query returns the currently defined time for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:TIME] <time><NL>`

TRANsition:TYPE

Command

```
:TRIGger:TRANsition[{1 | 2}]:TYPE {RISetime | FALLtime}
```

This command lets you select either a rise time or fall time transition violation trigger event.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:TRANsition:TYPE?
```

The query returns the currently defined transition type for the trigger transition violation.

Returned Format

```
[:TRIGger:TRANsition:TYPE] {RISetime | FALLtime}<NL>
```

TV:LINE

Command `:TRIGger:TV:LINE <line_number>`

The :TRIGger:TV:LINE command selects the horizontal line that you want to examine. The allowable line number entry depends on the :TRIGger:TV:MODE selected. Once the vertical sync pulse of the selected field is received, the trigger is delayed by the number of lines specified.

`<line_number>` Horizontal line number as shown below.

Video Standard	Field 1	Field 2	Alternate Field
NTSC/PAL-M	1 to 263	1 to 262	1 to 262
PAL/SECAM	1 to 313	314 to 625	1 to 312

EDTV/HDTV	Line numbers
EDTV 480p/60	1 to 525
EDTV 576p/50	1 to 625
HDTV 720p/60	1 to 750
HDTV 720p/50	1 to 750
HDTV 1080i/60	1 to 1125
HDTV 1080i/50	1 to 1125
HDTV1080p/60	1 to 1125
HDTV 1080p/50	1 to 1125
HDTV 1080p/30	1 to 1125
HDTV 1080p/25	1 to 1125
HDTV 1080p/24	1 to 1125

Query `:TRIGger:TV:LINE?`

The query returns the current line number.

Returned Format `[:TRIGger:TV:LINE] <line_number><NL>`

TV:MODE

Command

```
:TRIGger:TV:MODE {AFIelds | ALINes | ALTernate | FIElds1 | FIElds2 | LINE}
```

The :TRIGger:TV:MODE command determines which portion of the video waveform is used to trigger.

Query

```
:TRIGger:TV:MODE?
```

The query returns the current TV trigger mode.

Returned Format

```
[:TRIGger:TV:MODE] {AFIelds | ALINes | ALTernate | FIElds1 | FIElds2 | LINE}<NL>
```

TV:POLarity

Command

```
:TRIGger:TV:POLarity  {NEGative | POSitive}
```

The :TRIGger:TV:POLarity command specifies the vertical sync pulse polarity for the selected field used during TV mode triggering.

Query

```
:TRIGger:TV:POLarity?
```

The query returns the currently selected sync pulse polarity.

Returned Format

```
[:TRIGger:TV:POLarity] {NEGative | POSitive}<NL>
```

TV:SOURce

Command `:TRIGger:TV:SOURce CHANnel<N>`

The :TRIGger:TV:SOURce command selects the source for the TV mode triggering. This is the source that will be used for subsequent :TRIGger:TV commands and queries.

`<N>` An integer, 1 - 4.

Query `:TRIGger:TV:SOURce?`

The query returns the currently selected standard TV trigger mode source.

Returned Format `[:TRIGger:TV:SOURce] CHANnel<N>NL>`

TV:STANdard

Command

```
:TRIGger:TV:STANdard {I1080L60HZ | I1080L50HZ |
L525 | L625 | P480L60HZ | P576L50HZ | P720L60HZ |
P720L50HZ | P1080L60HZ | P1080L50HZ | P1080L30HZ |
P1080L25HZ | P1080L24HZ | UDTV}
```

The TRIGger:TV:STANdard command sets triggering to one of the standard video types. There is also a user defined TV type that can be used to set the triggering to one of the non-standard types of video.

Query

```
:TRIGger:TV:STANdard?
```

The query returns the currently selected video standard.

Returned Format

```
[:TRIGger:TV:STANdard] {I1080L60HZ | I1080L50HZ | L525 | L625 |
P480L60HZ | P576L50HZ | P720L60HZ | P720L50HZ | P1080L60HZ |
P1080L50HZ | P1080L40HZ | P1080L30HZ | P1080L25HZ |
P1080L24HZ | UDTV}<NL>
```

TV:UDTV:ENUMber

Command `:TRIGger:TV:UDTV:ENUMber <count>`

The :TRIGger:TV:UDTV:ENUMber command specifies the number of events (horizontal sync pulses) to delay after arming the trigger before looking for the trigger event. Specify conditions for arming the trigger using:

TRIGger:TV:UDTV:PGTHan, and

TRIGger:TV:UDTV:POLarity.

`<count>` An integer for the number of events to delay. Allowable values range from 1 to 16,000,000.

Query `:TRIGger:TV:UDTV:ENUMber?`

The query returns the currently programmed count value.

Returned Format `[:TRIGger:TV:UDTV:ENUMber] <count><NL>`

TV:UDTV:HSYNc

Command `:TRIGger:TV:UDTV:HSYNc {ON | 1} | {OFF | 0}}`

This command enables the horizontal sync mode of triggering.

Query `:TRIGger:TV:UDTV:HSYNc?`

The query returns the current state of the horizontal sync mode of triggering.

Returned Format `[:TRIGger:TV:UDTV:HSYNc] {1 | 0}<NL>`

TV:UDTV:HTIMe

Command `:TRIGger:TV:UDTV:HTIMe <time>`

The :TRIGger:TV:UDTV:HTIMe command sets the time that a sync pulse must be present to be considered a valid sync pulse.

`<time>` A real number that is the time width for the sync pulse.

Query `:TRIGger:TV:UDTV:HTIMe?`

The query returns the currently defined time for the sync pulse width.

Returned Format `[:TRIGger:TV:UDTV:HTIMe] <time><NL>`

TV:UDTV:PGTHan

Command `:TRIGger:TV:UDTV:PGTHan <lower_limit>`

The :TRIGger:TV:UDTV:PGTHan (Present Greater THan) command specifies the minimum pulse width of the waveform used to arm the trigger used during user-defined trigger mode.

`<lower_limit>` Minimum pulse width (time >), from 5 ns to 9.9999999 s.

Query `:TRIGger:TV:UDTV:PGTHan?`

The query returns the currently selected minimum pulse width.

Returned Format `[:TRIGger:TV:UDTV:PGTHan] <lower_limit><NL>`

TV:UDTV:POLarity

Command

```
:TRIGger:TV:UDTV:POLarity {NEGative | POSitive}
```

The :TRIGger:TV:UDTV:POLarity command specifies the polarity for the sync pulse used to arm the trigger in the user-defined trigger mode.

Query

```
:TRIGger:TV:UDTV:POLarity?
```

The query returns the currently selected UDTV sync pulse polarity.

Returned Format

```
[:TRIGger:TV:UDTV:POLarity] {NEGative | POSitive}<NL>
```

WINDow:CONDition

Command

```
:TRIGger:WINDow[{1 | 2}]:CONDition {ENTer | EXIT |
INSide [,{GTHan | LTHan}] |
OUTSide [,{GTHan | LTHan}]}
```

This command describes the condition applied to the trigger window to actually generate a trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:WINDow:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:WINDow:CONDition] {ENTer | EXIT |
INSide,{GTHan | LTHan} | OUTSide,{GTHan | LTHan}}<NL>
```

WINDow:SOURce

Command `:TRIGger:WINDow[{1 | 2}]:SOURce CHANnel<N>`

This command specifies the channel source used to trigger the oscilloscope with the window trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<N>` An integer, 1 - 4.

Query `:TRIGger:WINDow:SOURce?`

The query returns the currently defined channel source for the window trigger.

Returned Format `[:TRIGger:WINDow:SOURce] CHANnel<N><NL>`

WINDow:TIME

Command `:TRIGger:WINDow[{1 | 2}]:TIME <time>`

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

`<time>` The time for the trigger violation transition, in seconds.

Query `:TRIGger:WINDow:TIME?`

The query returns the currently defined time for the trigger window timeout.

Returned Format `[:TRIGger:WINDow:TIME] <time><NL>`

WINDow:TPOint

Command

```
:TRIGger:WINDow[{1 | 2}]:TPOint {BOUNdary | TIMeout}
```

This command specifies whether the window trigger should occur at the boundary of the window or at a specified timeout period.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:WINDow:TPOint?
```

The query returns the currently defined trigger on point for the pulse width trigger.

Returned Format

```
[:TRIGger:PWIDth:TPOint] {BOUNdary | TIMeout}<NL>
```

Waveform Commands

Waveform Commands

The WAVeform subsystem is used to transfer waveform data between a computer and the oscilloscope. It contains commands to set up the waveform transfer and to send or receive waveform records to or from the oscilloscope. These WAVeform commands and queries are implemented in the Infiniium Oscilloscopes:

- BANDpass?
- BYTeorder
- COMPlete?
- COUNt?
- COUPling?
- DATA?
- FORMat
- POINts?
- PREamble
- SEGMented:ALL
- SEGMented:COUNt?
- SEGMented:TTAG?
- SEGMented:XLIst?
- SOURce
- STReaming
- TYPE?
- VIEW
- XDISplay?
- XINCrement?
- XORigin?
- XRANge?
- XREFerence?
- XUNits?
- YDISplay?
- YINCrement?
- YORigin?

- YRANge?
- YREFerence?
- YUNits?

Data Acquisition

When data is acquired using the DIGitize command, the data is placed in the channel or function memory of the specified source. After the DIGitize command executes, the oscilloscope is stopped. If the oscilloscope is restarted by your program or from the front panel, the data acquired with the DIGitize command is overwritten.

You can query the preamble, elements of the preamble, or waveform data while the oscilloscope is running, but the data will reflect only the current acquisition, and subsequent queries will not reflect consistent data. For example, if the oscilloscope is running and you query the X origin, the data is queried in a separate command, it is likely that the first point in the data will have a different time than that of the X origin. This is due to data acquisitions that may have occurred between the queries. For this reason, Agilent Technologies does not recommend this mode of operation. Instead, you should use the DIGitize command to stop the oscilloscope so that all subsequent queries will be consistent.

Function and channel data are volatile and must be read following a DIGitize command or the data will be lost when the oscilloscope is turned off.

Waveform Data and Preamble

The waveform record consists of two parts: the preamble and the waveform data. The waveform data is the actual sampled data acquired for the specified source. The preamble contains the information for interpreting the waveform data, including the number of points acquired, the format of the acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data.

The values in the preamble are set when you execute the DIGitize command. The preamble values are based on the current settings of the oscilloscope's controls.

Data Conversion

Data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y origins and X and Y increments. These values can be read using the :WAVeform:XORigin?, WAVeform:YORigin?, WAVeform:XINCrement?, and WAVeform:YINCreament? queries.

Conversion from Data Values to Units

To convert the waveform data values (essentially A/D counts) to real-world units, such as volts, use the following scaling formulas:

Y-axis Units = data value x Yincrement + Yorigin (analog channels)
X-axis Units = data index x Xincrement + Xorigin,
where the data index starts at zero: 0, 1, 2, ..., n-1.

The first data point for the time (X-axis units) must be zero, so the time of the first data point is the X origin.

Data Format for Data Transfer

There are four types of data formats that you can select using the :WAVeform:FORMat command: ASCii, BYTE, WORD, and BINary. Refer to the FORMat command in this chapter for more information on data formats.

BANDpass?

Query `:WAVeform:BANDpass?`

The :WAVeform:BANDpass? query returns an estimate of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limits are computed as a function of the coupling and the selected filter mode. The cutoff frequencies are derived from the acquisition path and software filtering.

Returned Format `[:WAVeform:BANDpass]<lower_cutoff>,<upper_cutoff><NL>`

`<lower_cutoff>` Minimum frequency passed by the acquisition system.

`<upper_cutoff>` Maximum frequency passed by the acquisition system.

Example This example places the estimated maximum and minimum bandwidth limits of the source waveform in the string variable, Bandwidth$, then prints the contents of the variable to the computer's screen.

```
10  DIM Bandwidth$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:BANDPASS?"
30  ENTER 707;Bandwidth$
40  PRINT Bandwidth$
50  END
```

BYTeorder

Command

```
:WAVeform:BYTeorder {MSBFirst | LSBFirst}
```

The :WAVeform:BYTeorder command selects the order in which bytes are transferred to and from the oscilloscope using WORD and LONG formats. If MSBFirst is selected, the most significant byte is transferred first. Otherwise, the least significant byte is transferred first. The default setting is MSBFirst.

The data transfer rate is faster using the LSBFirst byte order.

MSBFirst is for microprocessors, where the most significant byte resides at the lower address. LSBFirst is for microprocessors, where the least significant byte resides at the lower address.

Example

This example sets up the oscilloscope to send the most significant byte first during data transmission.

```
10  OUTPUT 707;":WAVEFORM:BYTEORDER MSBFIRST"
20  END
```

Query

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder? query returns the current setting for the byte order.

Returned Format

```
[:WAVeform:BYTeorder] {MSBFirst | LSBFirst}<NL>
```

Example

This example places the current setting for the byte order in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[10]!Dimension variable
20  OUTPUT 707;":WAVEFORM:BYTEORDER?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

COMPlete?

Query

```
:WAVeform:COMPlete?
```

The :WAVeform:COMPlete? query returns the percent of time buckets that are complete for the currently selected waveform.

For the NORMal, RAW, and INTerpolate waveform types, the percent complete is the percent of the number of time buckets that have data in them, compared to the memory depth.

For the AVERage waveform type, the percent complete is the number of time buckets that have had the specified number of hits divided by the memory depth. The hits are specified by the :ACQuire:AVERage:COUNt command.

For the VERSus waveform type, percent complete is the least complete of the X-axis and Y-axis waveforms.

Returned Format

```
[:WAVeform:COMPlete] <criteria><NL>
```

`<criteria>` 0 to 100 percent, rounded down to the closest integer.

Example

This example places the current completion criteria in the string variable, Criteria$, then prints the contents of the variable to the computer's screen.

```
10  DIM Criteria$[10]!Dimension variable
20  OUTPUT 707;":WAVEFORM:COMPLETE?"
30  ENTER 707;Criteria$
40  PRINT Criteria$
50  END
```

COUNt?

Query

```
:WAVeform:COUNt?
```

The :WAVeform:COUNt? query returns the fewest number of hits in all of the time buckets for the currently selected waveform. For the AVERage waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value specified with the :ACQuire:AVERage:COUNt command.

For the NORMal, RAW, INTerpolate, and VERSus waveform types, the count value returned is one, unless the data contains holes (sample points where no data is acquired). If the data contains holes, zero is returned.

Returned Format

```
[:WAVeform:COUNt] <number><NL>
```

<number> An integer. Values range from 0 to 1 for NORMal, RAW, or INTerpolate types, and VERSus type. If averaging is on values range from 0 to 65536.

Example

This example places the current count field value in the string variable, Count$, then prints the contents of the variable to the computer's screen.

```
10  DIM Count$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:COUNT?"
30  ENTER 707;Count$
40  PRINT Count$
50  END
```

COUPling?

Query

```
:WAVeform:COUPling?
```

The :WAVeform:COUPling? query returns the input coupling of the currently selected source and always returns DC. This query is provided for compatibility to other Infiniium oscilloscopes.

Returned Format

```
[:WAVeform:COUPling] DC<NL>
```

Example

This example places the current input coupling of the selected waveform in the string variable, Setting$, then prints the contents of the variable.

```
10  DIM Setting$[10]!Dimension variable
20  OUTPUT 707;":WAVEFORM:COUPLING?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

See Also

The :CHANnel<N>:INPut command sets the coupling for a particular channel.

You can use the :WAVeform:SOURce command to set the source for the coupling query.

Source	Return Value
CGRade	**Coupling of the lowest numbered channel that is on.**
HISTogram	**The coupling of the selected channel. For functions, the coupling of the lowest numbered channel in the function.**
CHANnel	**The coupling of the channel number**
FUNCtion	**The coupling of the lowest numbered channel in the function**
WMEMory	**The coupling value of the source that was loaded into the waveform memory. If channel 1 was loaded, it would be the channel 1 coupling value.**

DATA?

Query `:WAVeform:DATA? [<start>[,<size>]]`

The :WAVeform:DATA? query outputs waveform data to the computer over the GPIB Interface. The data is copied from a waveform memory, function, channel, or digital channel previously specified with the :WAVeform:SOURce command.

The data is returned response depends upon the setting of the :WAVeform:STReaming command. See see "Streaming Off" on page 28-11 or see "Returned Format" on page 28-13 for more detail.

If the waveform data is ASCII formatted, then waveform data is separated by commas.

The preamble queries, such as :WAVeform:XINCrement, can be used to determine the vertical scaling, the horizontal scaling, and so on.

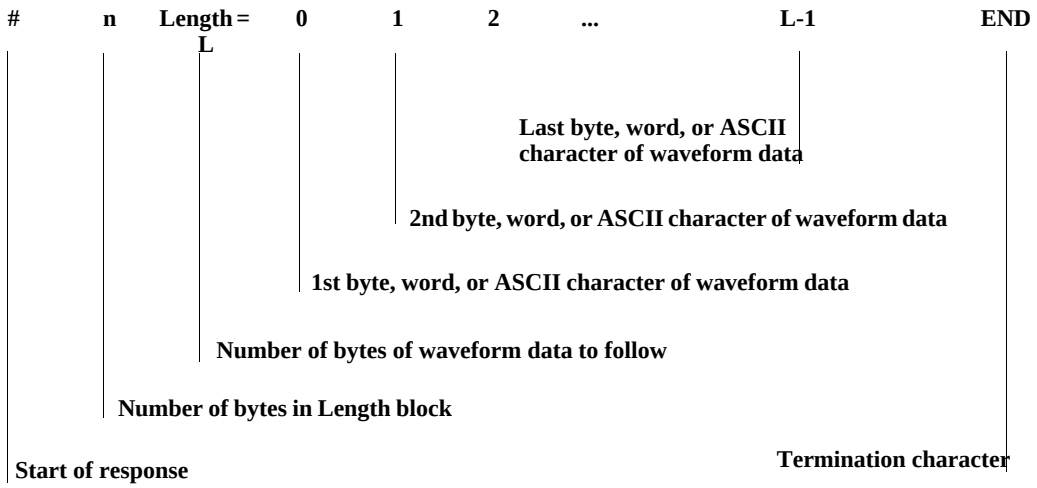
`<start>` An integer value which is the starting point in the source memory which is the first waveform point to transfer.

`<size>` An integer value which is the number of points in the source memory to transfer. If the size specified is greater than the amount of available data then the size is adjusted to be the maximum available memory depth minus the <start> value.

Streaming Off

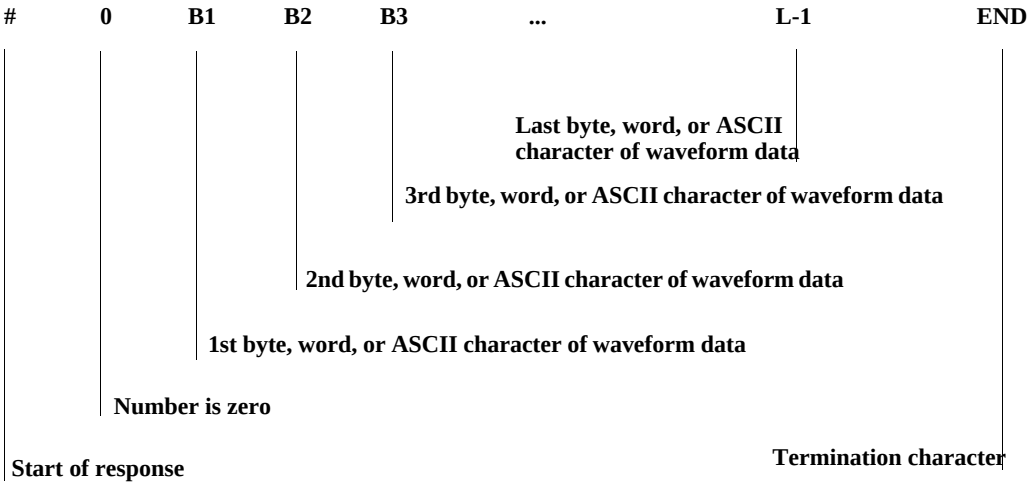
The returned waveform data response depends upon the setting of the :WAVeform:STReaming command. When the data format is BYTE and streaming is off, the number of waveform points must be less than 1,000,000,000 or an error occurs and only 999,999,999 bytes of data are sent. When the data format is WORD and streaming is off, the number of waveform points must be less than 500,000,000 or an error occurs and only 499,999,999 words of data are sent.

The returned waveform data in response to the :WAVeform:DATA? query is in the following order.



Streaming On

When streaming is on there is no limit on the number of waveform data points that are returned. It is recommended that any new programs use streaming on to send waveform data points. The waveform data response when streaming is on is as follows.



Returned Format `[:WAVeform:DATA] <block_data>[,<block_data>]<NL>`

BASIC Example

This example places the current waveform data from channel 1 of the array Wdata in the word format.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1!Select source
30  OUTPUT 707;":WAVEFORM:FORMAT WORD"!Select word format
40  OUTPUT 707;":WAVEFORM:DATA?"
50  ENTER 707 USING "#,1A";Pound_sign$
60  ENTER 707 USING "#,1D";Header_length
70  ENTER 707 USING "#,"&VAL$(Header_length)&"D";Length
80  Length = Length/2!Length in words
90  ALLOCATE INTEGER Wdata(1:Length)
100 ENTER 707 USING "#,W";Wdata(*)
110 ENTER 707 USING "-K,B";End$
120 END
```

HP BASIC Image Specifiers

 # is an HP BASIC image specifier that terminates the statement when the last ENTER item is terminated. EOI and line feed are the item terminators.

1A is an HP BASIC image specifier that places the next character received in a string variable.

1D is an HP BASIC image specifier that places the next character in a numeric variable.

W is an HP BASIC image specifier that places the data in the array in word format with the first byte entered as the most significant byte.

-K is an HP BASIC image specifier that places the block data in a string, including carriage returns and line feeds until EOI is true or when the dimensioned length of the string is reached.

B is an HP BASIC specifier that enters the next byte in a variable.

The format of the waveform data must match the format previously specified by the :WAVeform:FORMat, :WAVeform:BYTeorder, and :WAVeform:PREamble commands.

DATA? Example for Analog Channels

The following C example shows how to transfer WORD formatted waveform data for analog channels to a computer.

```
/* readdata. c */

/* Reading Word format example.  This program demonstrates the order
 * of commands suggested for operation of the Infiniium oscilloscope
 * via LAN.  This program initializes the oscilloscope, acquires data,
 * transfers data in WORD format, converts the data into time and
 * voltage values, and stores the data in a file as comma-separated
 * ascii values. This format is useful for spreadsheet and MATLAB
 * applications. It requires a waveform which is connected to Channel 1.
*/

#include <stdio.h>      /* location of: printf() */
#include <stdlib.h>     /* location of: atof(), atoi() */
#include <string.h>     /* location of: strlen() */
#include "sicl.h"

/* Prototypes */
int InitIO( void );
void WriteIO( char *buffer );
unsigned long ReadByte( char *buffer, unsigned long BytesToRead);
unsigned long ReadWord( char *buffer, int *reason,
                        unsigned long BytesToRead);
void ReadDouble( double *buffer );
void CloseIO( void );
void AcquireData( void );
void GetVoltageConversionFactors( double *yInc, double *yOrg );
void GetTimeConversionFactors( double *xInc, double *xOrg );
void WriteCsvToFile( unsigned long ByteToRead );
void SetupDataTransfer( void );

/* Defines */
#define MAX_LENGTH 10000000
#define INTERFACE "lan[130.29.70.247]:inst0"
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000
```

```
/* Globals */
INST bus;
INST scope;
char buffer[MAX_LENGTH];  /* Buffer for reading data */
double xOrg=0L, xInc=0L;  /* Values used to create time data */
double yOrg=0L, yInc=0L;  /* Values used to convert data to volts */


void main( void )
{
   unsigned long BytesToRead;

      if ( !InitIO() )
      {
            exit( 1 );
      }

   AcquireData();

   WriteIO( ":WAVeform:FORMat WORD" );    /* Setup transfer format */
   WriteIO( ":WAVeform:BYTeorder LSBFirst" ); /* Setup transfer of
                                                 LSB first */
   WriteIO( ":WAVeform:SOURce CHANnel1" ); /* Waveform data source
                                              channel 1 */
   WriteIO( ":WAVeform:STReaming 1" ); /* Turn on waveform
                                          streaming of data */

   GetVoltageConversionFactors( &yInc, &yOrg );
   GetTimeConversionFactors( &xInc, &xOrg );
   BytesToRead = MAX_LENGTH;
   SetupDataTransfer();
   WriteCsvToFile( BytesToRead );

   CloseIO( );

}
```

```
/*****************************************************************
 * Function name:  InitIO
 *   Parameters:   none
 *   Return value: TRUE if successfull otherwise FALSE
 *   Description:  This routine initializes the SICL environment.
 *                 It sets up error handling, opens both an interface
 *                 and device session, sets timeout values, clears
 *                 the LAN interface card, and clears the
 *                 oscilloscope's LAN interface by performing a
 *                 Selected Device Clear.
*****************************************************************/

int InitIO( void )
{

      ionerror( I_ERROR_EXIT );   /* set-up interface error handling */

      bus = iopen( INTERFACE );   /* open interface session */
      if ( bus == 0 )
      {
            printf( "Bus session invalid\n" );
            return FALSE;
      }

      itimeout( bus, IO_TIMEOUT );  /* set bus timeout */
      iclear( bus );                /* clear the interface */

      scope = bus;                  /* open the scope device session */

      return TRUE;
}
```

```
/*********************************************************************
 * Function name:  WriteIO
 *   Parameters:   char *buffer which is a pointer to the character
 *                    string to be output
 *   Return value: none
 *   Description:  This routine outputs strings to the oscilloscope
 *                 device session using SICL commands.
*********************************************************************/

void WriteIO( char *buffer )
{
      unsigned long actualcnt;
      unsigned long BytesToRead;
      int send_end = 1;

      BytesToRead = strlen( buffer );

      iwrite( scope, buffer, BytesToRead, send_end, &actualcnt );

}
```

```
/*****************************************************************
 * Function name:  ReadByte
 *   Parameters:   char *buffer which is a pointer to the array to
 *                   store the read bytes
 *                 unsigned long BytesToRead which indicates the
 *                   maximum number of bytes to read
 *   Return value: integer which indicates the actual number of bytes
 *                 read
 *   Description: This routine inputs strings from the scope device
 *                session using SICL commands.
*****************************************************************/

unsigned long ReadByte( char *buffer, unsigned long BytesToRead )
{
      unsigned long BytesRead;
      int reason;

      BytesRead = BytesToRead;

      iread( scope, buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}
```

```
/*****************************************************************
 * Function name:  ReadWord
 *   Parameters:   short *buffer which is a pointer to the word array
 *                    to store the bytes read
 *                 int reason which is the reason that the read
 *                    terminated
 *                 unsigned long BytesToRead which indicates the
 *                    maximum number of bytes to read
 *   Return value: integer which indicates the actual number of
 *                    bytes read
 *   Description: This routine inputs an array of short values from
 *                the oscilloscope device session using SICL commands.
*****************************************************************/

unsigned long ReadWord( char *buffer, int *reason,
                        unsigned long BytesToRead )
{
   long BytesRead;

   iread( scope, buffer, BytesToRead, reason, &BytesRead );

   return BytesRead;
}
```

```
/*****************************************************************
 * Function name:  ReadDouble
 *   Parameters:   double *buffer which is a pointer to the float
 *                    value to read
 *   Return value: none
 *   Description: This routine inputs a float value from the
 *                oscilloscope device session using SICL commands.
*****************************************************************/


void ReadDouble( double *buffer )
{
   iscanf( scope, "%lf", buffer );
}

/*****************************************************************
 * Function name: close_IO
 *   Parameters:   none
 *   Return value: none
 *   Description:  This routine closes device and interface sessions
 *                 for the SICL environment, and calls the routine
 *                 _siclcleanup which de-allocates resources
 *                 used by the SICL environment.
*****************************************************************/

void CloseIO( void )
{

   iclose( scope );   /* close device session */
   iclose( bus );     /* close interface session */

   _siclcleanup();    /* required for 16-bit applications */

}
```

```
/*****************************************************************
 * Function name:  AcquireData
 *   Parameters:   none
 *   Return value: none
 *   Description: This routine acquires data using the current
 *                oscilloscope settings.
*****************************************************************/

void AcquireData( void )
{
      /*
       * The root level :DIGitize command is recommended for
       * acquiring new waveform data. It initialize's the
       * oscilloscope's data buffers, acquires new data,
       * and ensures that acquisition criteria are met before the
       * acquisition is stopped.  Note that the display is
       * automatically turned off when you use this form of the
       * :DIGitize command and must be turned on to view the
       * captured data on screen.
      */

      WriteIO(":DIGitize CHANnel1");
      WriteIO(":CHANnel1:DISPlay ON");

}
```

```
/*****************************************************************
 * Function name: GetVoltageConversionFactors
 *   Parameters:  double yInc which is the voltage difference
 *                  represented by adjacent waveform data digital codes.
 *                double yOrg which is the voltage value of digital
 *                  code 0.
 *   Return value: none
 *   Description: This routine reads the conversion factors used to
 *                convert waveform data to volts.
*****************************************************************/

void GetVoltageConversionFactors( double *yInc, double *yOrg )
{

      /* Read values which are used to convert data to voltage values */

      WriteIO(":WAVeform:YINCrement?");
      ReadDouble( yInc );

      WriteIO(":WAVeform:YORigin?");
      ReadDouble( yOrg );

}
```

```
/*****************************************************************
 * Function name:  SetupDataTransfer
 *   Parameters:   none
 *   Return value: none
 *   Description: This routine sets up the waveform data transfer and
 *                removes the # and 0 characters.
*****************************************************************/

void SetupDataTransfer( void )
{
   char cData;

   WriteIO( ":WAVeform:DATA?" );   /* Request waveform data */

   /* Find the # character */

   do
   {
         ReadByte( &cData, 1L );
   } while ( cData != '#' );

   /* Find the 0 character */

   do
   {
         ReadByte( &cData, 1L );
   } while ( cData != '0' );
}
```

```
/*****************************************************************
 * Function name:  GetTimeConversionFactors
 *   Parameters:   double xInc which is the time between consecutive
 *                    sample points.
 *                 double xOrg which is the time value of the first
 *                    data point.
 *   Return value: none
 *   Description: This routine transfers the waveform conversion
 *                factors for the time values.
*****************************************************************/

void GetTimeConversionFactors( double *xInc, double *xOrg )
{

      /* Read values which are used to create time values */

   WriteIO(":WAVeform:XINCrement?");
   ReadDouble( xInc );

   WriteIO(":WAVeform:XORigin?");
   ReadDouble( xOrg );

}
```

```
/*****************************************************************
 * Function name:  WriteCsvToFile
 *   Parameters:   unsigned long BytesToRead which is the number of
 *                   data points to read
 *   Return value: none
 *   Description: This routine stores the time and voltage
 *                information about the waveform as time and
 *                voltage separated by commas to a file.
*****************************************************************/

void WriteCsvToFile( unsigned long BytesToRead )
{
  FILE *fp;
  int done = FALSE;
  int reason = 0;
  unsigned long i;
  unsigned long j = 0;
  unsigned long BytesRead = 0L;
  double Time;
  double Volts;
  short *buff;

  fp = fopen( "pairs.csv", "wb" ); /* Open file in binary mode - clear
                                      file if it already exists */

  if (fp != NULL)
  {
        while( !done )
        {
           BytesRead = ReadWord( buffer, &reason, BytesToRead );

        switch( reason )
        {
         case I_TERM_MAXCNT:
            done = FALSE;
            break;
         case I_TERM_END:
            done = TRUE;
            break;
```

```
            case I_TERM_CHR:
               done = TRUE;
               break;
            default:
               done = TRUE;
               break;
         };

         buff = (short *) buffer;

         for( i = 0; i < ((BytesRead - 1)/2); i++)
         {
            Time = (j * xInc) + xOrg; /* calculate time */
            j = j + 1;

            Volts = (buff[i] * yInc) + yOrg;/* calculate voltage */

            fprintf( fp, "%e,%f\n", Time, Volts );
         }
      }
     fclose( fp );

   }
   else
   {
      printf("Unable to open file 'pairs.csv'\n");
   }
}
```

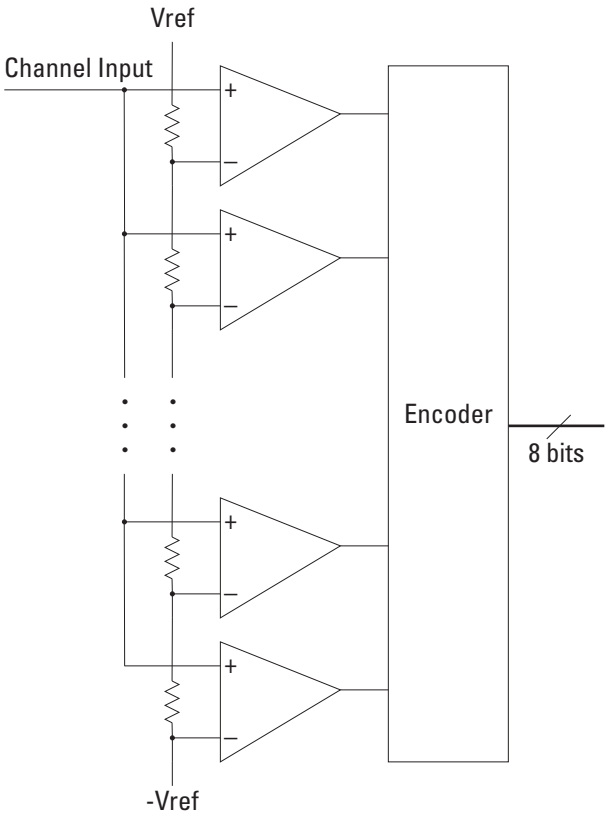
Understanding WORD and BYTE Formats

Before you can understand how the WORD and BYTE downloads work, it is necessary to understand how Infiniium creates waveform data.

Analog-to-digital Conversion Basics

The input channel of every digital sampling oscilloscope contains an analog-to-digital converter (ADC) as shown in figure 28-1. The 8-bit ADC in Infiniium consists of 256 voltage comparators. Each comparator has two inputs. One input is connected to a reference dc voltage level and the other input is connected to the channel input. When the voltage of the waveform on the channel input is greater than the dc level, then the comparator output is a 1 otherwise the output is a 0. Each of the comparators has a different reference dc voltage. The output of the comparators is converted into an 8-bit integer by the encoder.

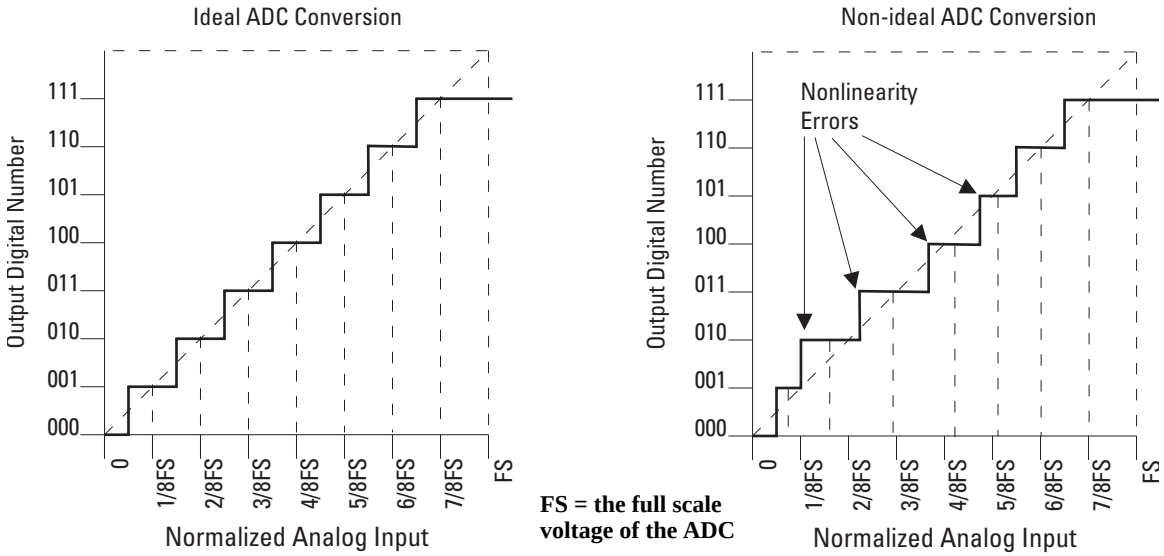
Figure 28-1



Block Diagram of an ADC

All ADCs have non-linearity errors which, if not corrected, can give less accurate vertical measurement results. For example, the non-linearity error for a 3-bit ADC is shown in the following figure.

Figure 28-2



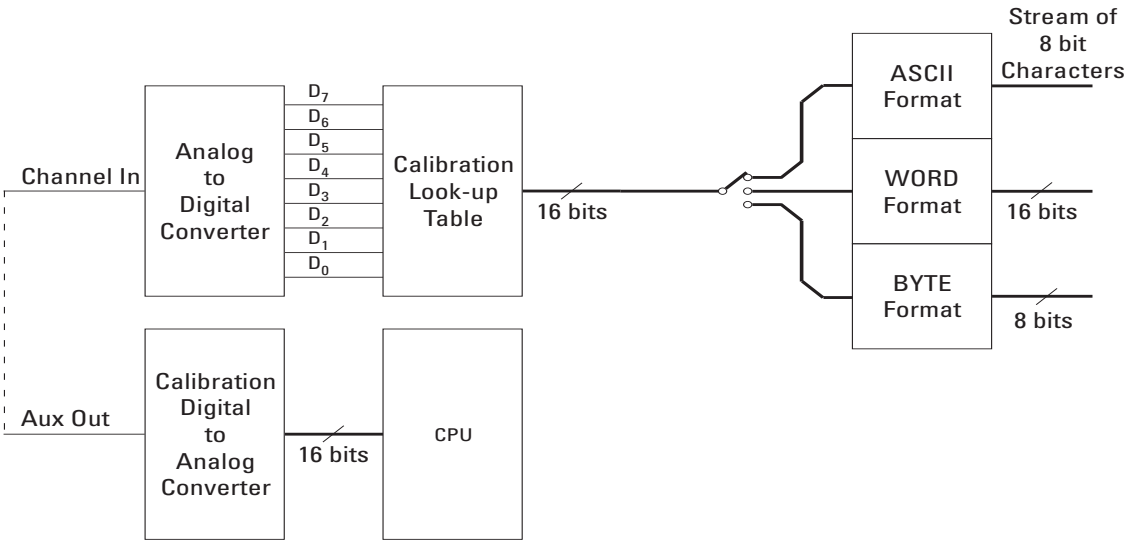
FS = the full scale voltage of the ADC

ADC Non-linearity Errors for a 3-bit ADC

The graph on the left shows an ADC which has no non-linearity errors. All of the voltage levels are evenly spaced producing output codes that represent evenly spaced voltages. In the graph on the right, the voltages are not evenly spaced with some being wider and some being narrower than the others.

When you calibrate your Infiniium, the input to each channel, in turn, is connected to the Aux Out connector. The Aux Out is connected to a 16-bit digital-to-analog convertor (DAC) whose input is controlled by Infiniium's CPU. There are 65,536 dc voltage levels that are produced by the 16-bit DAC at the Aux Out. At each dc voltage value, the output of the ADC is checked to see if a new digital code is produced. When this happens, a 16-bit correction factor is calculated for that digital code and this correction factor is stored in a Calibration Look-up Table.

Figure 28-3



Data Flow in Infiniium

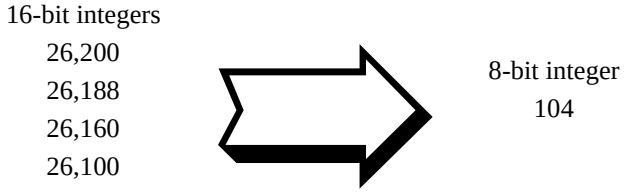
This process continues until all 256 digital codes are calibrated. The calibration process removes most of the non-linearity error of the ADC which yields more accurate vertical voltage values.

During normal operation of the oscilloscope, the output of the ADC is used as an address to the Calibration Look-up Table which produces 16-bit data for the oscilloscope to process and display. The output of the ADC is a signed 8-bit integer and the output of the Calibration Look-up Table is a signed 16-bit integer. If the amplitude of the input waveform is larger than the maximum dc reference level of the ADC, the ADC will output the maximum 8-bit value that it can (255). This condition is called ADC clipping. When the 255 digital code is applied to the Calibration Look-up Table, a 16-bit value, such as 26,188 could be produced which represents an ADC clipped value. This number will vary from one oscilloscope to the next.

WORD and BYTE Data Formats

When downloading the waveform data in WORD format, the 16-bit signed integer value for each data point is sent in two consecutive 8-bit bytes over GPIB. Whether the least significant byte (LSB) or the most significant byte (MSB) is sent first depends on the byte order determined by the BYTeorder command.

Before downloading the waveform data in BYTE format, each 16-bit signed integer is converted into an 8-bit signed integer. Because there are more possible 16-bit integers than there are 8-bit integers, a range of 16-bit integers is converted into single 8-bit numbers. For example, the following 16-bit numbers are all converted into one 8-bit number.



This conversion is what makes the BYTE download format less accurate than the WORD format.

FORMat

Command `:WAVeform:FORMat {ASCii | BINary | BYTE | WORD}`

The :WAVeform:FORMat command sets the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope, and pertains to all waveforms. The default format is ASCii.

Histograms can only be ACSii or BINary.

Digital busses and pod-all are not compatible with BYTE.

Selecting a Format

Type	Advantages	Disadvantages
ASCii	**Data is returned as voltage values and does not need to be converted and is as accurate as WORD format.**	**Very slow data download rate.**
BYTE	**Data download rate is twice as fast as the WORD format.**	**Data is less accurate than the WORD format for analog channels.**
WORD	**Data is the most accurate for analog channels.**	**Data download rate takes twice as long as the BYTE format.**
BINary	**This format can be used for analog channels and for HISTogram source.**	**Data download rate takes twice as long as the BYTE format for analog channels.**

`ASCii` ASCii-formatted data consists of waveform data values converted to the currently selected units, such as volts, and are output as a string of ASCII characters with each value separated from the next value by a comma. The values are formatted in floating point engineering notation. For example:

8.0836E+2,8.1090E+2,...,-3.1245E-3

The ASCii format does not send out the header information indicating the number of bytes being downloaded.

In ASCii format:

- The value “99.999E+36” represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

`BYTE` BYTE-formatted data is formatted as signed 8-bit integers. If you use BASIC, you need to create a function to convert these signed bits to signed integers. In BYTE format:

- The value 125 represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

The waveform data values are converted from 16-bit integers to 8-bit integers before being downloaded to the computer. For more information see "Understanding WORD and BYTE Formats" on page 28-28.

`WORD` WORD-formatted data is transferred as signed 16-bit integers in two bytes. If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each word is sent first. In WORD format:

- The value 31232 represents a hole level. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

For more information see "Understanding WORD and BYTE Formats" on page 28-28.

`BINary` BINary-formatted data can be used with any SOURce. When a source is any valid source except for histogram, the data is return in WORD format.

When the source is set to HISTogram, the data is transferred as signed 64-bit integers in 8 bytes. The are no hole values in the histogram data.

If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each long word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each long word is sent first.

Example

This example selects the WORD format for waveform data transmission.

```
10  OUTPUT 707;":WAVEFORM:FORMAT WORD"
20  END
```

Query

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat? query returns the current output format for transferring waveform data.

Returned Format

```
[:WAVeform:FORMat] {ASCii | BINary | BYTE | WORD}<NL>
```

Example

This example places the current output format for data transmission in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:FORMAT?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

POINts?

Query

```
:WAVeform:POINts?
```

The :WAVeform:POINts? query returns the points value in the current waveform preamble. The points value is the number of time buckets contained in the waveform selected with the :WAVeform:SOURce command. If the Sin(x)/x interpolation filter is enabled, the number of points can be larger than the oscilloscope's memory depth setting because the waveform includes the interpolated points.

Returned Format

```
[:WAVeform:POINts] <points><NL>
```

<points> An integer. See the :ACQuire:POINts command for a table of possible values.

Example

This example places the current acquisition length in the numeric variable, Length, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:POINTS?"
30  ENTER 707;Length
40  PRINT Length
50  END
```

Turn Headers Off

When you are receiving numeric data into numeric variables, you should turn the headers off. Otherwise, the headers may cause misinterpretation of returned data.

See Also

The :ACQuire:POINts command in the ACQuire Commands chapter.

PREamble

Query `:WAVeform:PREamble?`

The :WAVeform:PREamble? query outputs a waveform preamble to the computer from the waveform source, which can be a waveform memory or channel buffer.

Returned Format `[:WAVeform:PREamble] <preamble_data><NL>`

The preamble can be used to translate raw data into time and voltage values. The following lists the elements in the preamble.

`<preamble_data>` <format>, <type>, <points>, <count> ,
<X increment>, <X origin>, < X reference>,
<Y increment>, <Y origin>, <Y reference>,
<coupling>,
<X display range>, <X display origin>,
<Y display range>, <Y display origin>,
<date>, <time>,
<frame model #>,
<acquisition mode>, <completion>,
<X units>, <Y units>,
<max bandwidth limit>, <min bandwidth limit>

`<format>` 0 for ASCii format.
1 for BYTE format.
2 for WORD format.
3 for LONG format.
4 for LONGLONG

`<type>` 1 RAW type.
2 AVERage type.
3 VHIStogram.
4 HHIStogram.
5 not used.
6 INTERPOLATE type.
7 not used.
8 not used.
9 not used.
10 PDETect.

`<points>` The number of data points or data pairs contained in the waveform data. (See :ACQuire:POINts.)

<count> For the AVERAGE waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value requested with the :ACQuire:AVERage:COUNt command. For RAW and INTERPOLATE waveform types, this value is 0 or 1. The count value is ignored when it is sent to the oscilloscope in the preamble.
(See :WAVeform:TYPE and :ACQuire:COUNt.)

<X increment> The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.
(See the :WAVeform:XINCrement? query.)

<X origin> The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.
(See the :WAVeform:XORigin? query.)

<X reference> The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this oscilloscope, the value is always zero. (See the :WAVeform:XREFerence? query.)

<Y increment> The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. If the value is zero then no data has been acquired.
(See the :WAVeform:YINCrement? query.)

<Y origin> The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. If the value is zero then no data has been acquired.
(See the :WAVeform:YORigin? query.)

<Y reference> The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this oscilloscope, this value is always zero.
(See the :WAVeform:YREFerence? query.)

<coupling> 0 for AC coupling.
1 for DC coupling.
2 for DCFIFTY coupling.
3 for LFREJECT coupling.

<X display range> The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.
(See the :WAVeform:XRANge? query.)

<X display origin> The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.
(See the :WAVeform:XDISplay? query.)

<Y display range> The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. If the value is zero then no data has been acquired.
(See the :WAVeform:YRANge? query.)

<Y display origin> The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. If the value is zero then no data has been acquired.
(See the :WAVeform:YDISplay? query.)

<date> A string containing the date in the format DD MMM YYYY, where DD is the day, 1 to 31; MMM is the month; and YYYY is the year.

<time> A string containing the time in the format HH:MM:SS:TT, where HH is the hour, 0 to 23, MM is the minutes, 0 to 59, SS is the seconds, 0 to 59, and TT is the hundreds of seconds, 0 to 99.

<frame_model_#> A string containing the model number and serial number of the oscilloscope in the format of MODEL#:SERIAL#.

<acquisition_mode> 0 for RTIMe mode.
1 for ETIMe mode.
2 not used.
3 for PDETect.

<completion> The completion value is the percent of time buckets that are complete. The completion value is ignored when it is sent to the oscilloscope in the preamble. (See the :WAVeform:COMPlete? query.)

<x_units>
<y_units> 0 for UNKNOWN units.
1 for VOLT units.
2 for SECOND units.
3 for CONSTANT units.
4 for AMP units.
5 for DECIBEL units.

<max bandwidth limit>
<min bandwidth limit> The band pass consists of two values that are an estimation of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limit is computed as a function of the selected coupling and filter mode.
(See the :WAVeform:BANDpass? query.)

See Table 28-1 for descriptions of all the waveform preamble elements.

HP BASIC Image Specifiers

 # is an HP BASIC image specifier that suppresses the automatic output of the EOL sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

Example

This example outputs the current waveform preamble for the selected source to the string variable, Preamble$.

```
10  DIM Preamble$[250]!Dimension variable
20  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
30  OUTPUT 707;":WAVEFORM:PREAMBLE?"
40  ENTER 707 USING "-K";Preamble$
50  END
```

Placing the Block in a String

-K is an HP BASIC image specifier that places the block data in a string, including carriage returns and line feeds, until EOI is true, or when the dimensioned length of the string is reached.

Table 28-1 Waveform Preamble Elements

Element	Description
Format	The format value describes the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope. (See :WAVeform:FORMat.)
Type	This value describes how the waveform was acquired. (See also the :WAVeform:TYPE? query.)
Points	The number of data points or data pairs contained in the waveform data. (See :ACQuire:POINts.)
Count	For the AVERAGE waveform type, the count is the number of averages that have occurred. For RAW and INTERPOLATE waveform types, this value is 0 or 1. The count value is ignored when it is sent to the oscilloscope in the preamble. (See :WAVeform:TYPE and :ACQuire:COUNt.)
X Increment	The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. (See the :WAVeform:XINCrement? query.)
X Origin	The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. (See the :WAVeform:XORigin? query.)
X Reference	The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this oscilloscope, the value is always zero. (See the :WAVeform:XREFerence? query.)
Y Increment	The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. (See the :WAVeform:YINCrement? query.)
Y Origin	The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. (See the :WAVeform:YORigin? query.)
Y Reference	The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this oscilloscope, this value is always zero. (See the :WAVeform:YREFerence? query.)
Coupling	The input coupling of the waveform. The coupling value is ignored when sent to the oscilloscope in the preamble. (See the :WAVeform:COUPling? query.)
X Display Range	The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. (See the :WAVeform:XRANge? query.)
X Display Origin	The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. (See the :WAVeform:XDISplay? query.)

Element	Description
Y Display Range	The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. (See the :WAVeform:YRANge? query.)
Y Display Origin	The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. (See the :WAVeform:YDISplay? query.)
Date	The date that the waveform was acquired or created.
Time	The time that the waveform was acquired or created.
Frame Model #	The model number of the frame that acquired or created this waveform. The frame model number is ignored when it is sent to an oscilloscope in the preamble.
Acquisition Mode	The acquisition sampling mode of the waveform. (See :ACQuire:MODE.)
Complete	The complete value is the percent of time buckets that are complete. The complete value is ignored when it is sent to the oscilloscope in the preamble. (See the :WAVeform:COMPlete? query.)
X Units	The X-axis units of the waveform. (See the :WAVeform:XUNits? query.)
Y Units	The Y-axis units of the waveform. (See the :WAVeform:YUNits? query.)
Band Pass	The band pass consists of two values that are estimates of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limit is computed as a function of the selected coupling and filter mode. (See the :WAVeform:BANDpass? query.)

See Also :WAVeform:DATA?

SEGMented:ALL

Command

```
:WAVeform:SEGMented:ALL {{ON | 1} | {OFF | 0}}
```

The :WAVeform:SEGmented:ALL command configures the DATA query for rapidly downloading all segments in one query.

The <start> and <size> optional parameters for the DATA query are still supported and represent the start and size of the data for each segment.

Powering on the oscilloscope or performing a Default Setup sets this command to OFF.

There is complete backwards compatibility when this command is set to OFF.

The ON setting only supports channel and pod sources. If other sources such as functions are selected, a settings conflict message appears during the DATA query and no data is downloaded.

In segmented acquisition mode, with this command set to ON, the number of segments is appended to end of the waveform preamble.

Example

This example turns on this command.

```
10  OUTPUT 707;":WAVeform:SEGMented:ALL ON"
20  END
```

Query

```
:WAVeform:SEGMented:ALL?
```

This query returns the status of this command.

SEGMented:COUNt?

Query `:WAVeform:SEGMented:COUNt?`

The :WAVeform:SEGMented:COUNt? query returns the index number of the last captured segment. A return value of zero indicates that the :ACQuire:MODE is not set to SEGMented.

`<index_number>` An integer number representing the index value of the last segment.

Returned Format [:WAVeform:SEGMented:COUNt] <index_number><NL>

Example This example returns the number of the last segment that was captured in the parameter Index and prints it to the computer screen.

```
10  OUTPUT 707;":WAVEFORM:SEGMENTED:COUNT?"
20  ENTER 707;Index
30  PRINT Index
40  END
```

SEGMented:TTAG?

Query `:WAVeform:SEGMented:TTAG?`

The :WAVeform:SEGMented:TTAG? query returns the time difference between the first segment's trigger point and the trigger point of the currently displayed segment.

`<delta_time>` A real number in exponential format representing the time value difference between the first segment's trigger point and the currently displayed segment.

Returned Format [:WAVeform:SEGMented:TTAG] <delta_time><NL>

Example This example returns the time from the first segment's trigger point and the currently displayed segment's trigger point in the parameter dtime and prints it to the computer screen.

```
10  OUTPUT 707;":WAVEFORM:SEGMENTED:TTAG?"
20  ENTER 707;dtime
30  PRINT dtime
40  END
```

SEGMented:XLISt?

Query

```
:WAVeform:SEGMented:XLISt? {RELXorigin | ABSXorigin
| TTAG}
```

The :WAVeform:SEGMented:XLISt? query rapidly downloads x-parameter values for all segments.

RELXorigin = relative X origin for each segment.

ABSXorigin = relative origin + time tag for each segment

TTAG = time tag for each segment

This query uses the DATA query format for the returned data and supports all waveform command options including: BYTeorder, FORmat (only ASCii or BINary (float64 with 8 bytes per value), SOURce (only CHANnel<N>), STReaming, VIEW.

SOURce

Command

```
:WAVeform:SOURce {CHANnel<N> | FUNCtion<N> |
HISTogram | WMEMory<N> | CLOCk | MTRend | MSPectrum
| EQUalized}
```

The :WAVeform:SOURce command selects a channel, function, waveform memory, or histogram as the waveform source.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

Example

This example selects channel 1 as the waveform source.

```
10  OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"
20  END
```

Query

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected waveform source.

Returned Format

```
[:WAVeform:SOURce] {CHANnel<N> | FUNCtion<N> | HISTogram |
 WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}<NL>
```

Example

This example places the current selection for the waveform source in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:SOURCE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

STReaming

Command

```
:WAVeform:STReaming {{ON | 1} | {OFF | 0}}
```

When enabled, :WAVeform:STReaming allows more than 999,999,999 bytes of data to be transfered from the Infiniium oscilloscope to a PC when using the :WAVefrom:DATA? query. See the :WAVeform:DATA? query for information on receiving this much data.

Example

This example turns on the streaming feature.

```
10  OUTPUT 707;":WAVeform:STReaming: ON"
20  END
```

Query

```
:WAVeform:STReaming?
```

The :WAVeform:STReaming? query returns the status of the streaming feature.

Returned Format

```
[:WAVeform:STReaming] {1 | 0}<NL>
```

TYPE?

Query `:WAVeform:TYPE?`

The :WAVeform:TYPE? query returns the current acquisition data type for the currently selected source. The type returned describes how the waveform was acquired. The waveform type may be RAW, INTerpolate, AVERage, HHIStogram, PDETect, or VHISTogram.

`RAW` RAW data consists of one data point in each time bucket with no interpolation.

`INTerpolate` In the INTerpolate acquisition type, the last data point in each time bucket is stored, and additional data points between the acquired data points are filled by interpolation.

`AVERage` AVERage data consists of the average of the first *n* hits in a time bucket, where *n* is the value in the count portion of the preamble. Time buckets that have fewer than *n* hits return the average of the data they contain. If the :ACQuire:COMPlete parameter is set to 100%, then each time bucket must contain the number of data hits specified with the :ACQuire:AVERage:COUNt command.

`HHIStogram` The data is a horizontal histogram. Histograms are transferred using the LONGLONG format. They can be generated using the Histogram subsystem commands.

`PDETect` PDETect data consists of two data points in each time bucket: the minimum values and the maximum values.

`VHIStogram` The data is a vertical histogram. Histograms are transferred using the LONGLONG format. They can be generated using the Histogram subsystem commands.

Returned Format

```
[:WAVeform:TYPE] {RAW | INTerpolate | AVERage |
HHIStogram | PDETect | VHIStogram}<NL>
```

Example This example places the current acquisition data type in the string variable, Type$, then prints the contents of the variable to the computer's screen.

```
10  DIM Type$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:TYPE?"
30  ENTER 707;Type$
40  PRINT Type$
50  END
```

VIEW

Command `:WAVeform:VIEW {ALL | MAIN | WINDow}`

The :WAVeform:VIEW command selects which view of the waveform is selected for data and preamble queries. You can set the command to ALL, MAIN, or WINDow. The view has different meanings depending upon the waveform source selected. The default setting for this command is ALL.

`Channels` For channels, you may select ALL, MAIN, or WINDow views. If you select ALL, all of the data in the waveform record is referenced. If you select MAIN, only the data in the main time base range is referenced. The first value corresponds to the first time bucket in the main time base range, and the last value corresponds to the last time bucket in the main time base range. If WINDow is selected, only data in the delayed view is referenced. The first value corresponds to the first time bucket in the delayed view and the last value corresponds to the last time bucket in the delayed view.

`Memories` For memories, if you specify ALL, all the data in the waveform record is referenced. WINDow and MAIN refer to the data contained in the memory time base range for the particular memory. The first value corresponds to the first time bucket in the memory time base range, and the last value corresponds to the last time bucket in the memory time base range.

`Functions` For functions, ALL, MAIN, and WINDow refer to all of the data in the waveform record.

Table 28-2 summarizes the parameters for this command for each source.

Example This example sets up the oscilloscope to view all of the data.

```
10  OUTPUT 707;":WAVEFORM:VIEW ALL"
20  END
```

Table 28-2 **Waveform View Parameters**

Source/Parameter	ALL	MAIN	WINDow
CHANNEL	All data	Main time base	Zoom
MEMORY	All data	Memory time base	Memory time base
FUNCTION	All data	All data	All data

Query

```
:WAVeform:VIEW?
```

The :WAVeform:VIEW? query returns the currently selected view.

Returned Format

```
[:WAVeform:VIEW] {ALL | MAIN | WINDow}<NL>
```

Example

This example returns the current view setting to the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:VIEW?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

XDISplay?

Query

```
:WAVeform:XDISplay?
```

The :WAVeform:XDISplay? query returns the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. For VERSus type waveforms, it is the value at the center of the X-axis of the display. This value is treated as a double precision 64-bit floating point number.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:XDISplay] <value><NL>
```

<value> A real number representing the X-axis value at the left edge of the display.

Example

This example returns the X-axis value at the left edge of the display to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XDISPLAY?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

XINCrement?

Query

```
:WAVeform:XINCrement?
```

The :WAVeform:XINCrement? query returns the duration between consecutive data points for the currently specified waveform source. For time domain waveforms, this is the time difference between consecutive data points. For VERSus type waveforms, this is the duration between levels on the X axis. For voltage waveforms, this is the voltage corresponding to one level.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:XINCrement] <value><NL>
```

<value> A real number representing the duration between data points on the X axis.

Example

This example places the current X-increment value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XINCREMENT?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

See Also

You can obtain the X-increment value through the :WAVeform:PREamble? query.

XORigin?

Query

```
:WAVeform:XORigin?
```

The :WAVeform:XORigin? query returns the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. For VERSus type waveforms, it is the X-axis value at level zero. For voltage waveforms, it is the voltage at level zero. The value returned by this query is treated as a double precision 64-bit floating point number.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:XORigin] <value><NL>
```

<value> A real number representing the X-axis value of the first data point in the data record.

Example

This example places the current X-origin value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XORIGIN?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

See Also

You can obtain the X-origin value through the :WAVeform:PREamble? query.

XRANge?

Query

```
:WAVeform:XRANge?
```

The :WAVeform:XRANge? query returns the X-axis duration of the displayed waveform. For time domain waveforms, it is the duration of the time across the display. For VERSus type waveforms, it is the duration of the waveform that is displayed on the X axis.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:XRANge] <value><NL>
```

<value> A real number representing the X-axis duration of the displayed waveform.

Example

This example returns the X-axis duration of the displayed waveform to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XRANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

XREFerence?

Query

```
:WAVeform:XREFerence?
```

The :WAVeform:XREFerence? query returns the data point or level associated with the X-origin data value. It is at this data point or level that the X origin is defined. In this oscilloscope, the value is always zero.

Returned Format

```
[:WAVeform:XREFerence] 0<NL>
```

Example

This example places the current X-reference value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XREFERENCE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

See Also

You can obtain the X-reference value through the :WAVeform:PREamble? query.

XUNits?

Query

```
:WAVeform:XUNits?
```

The :WAVeform:XUNits? query returns the X-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format

```
[:WAVeform:XUNits] {UNKNown | VOLT | SECond | CONStant | AMP
| DECibels | HERTz | WATT}<NL>
```

Example

This example returns the X-axis units of the currently selected waveform source to the string variable, Unit$, then prints the contents of the variable to the computer's screen.

```
10  DIM Unit$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:XUNITS?"
30  ENTER 707;Unit$
40  PRINT Unit$
50  END
```

YDISplay?

Query

```
:WAVeform:YDISplay?
```

The :WAVeform:YDISplay? query returns the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:YDISplay] <value><NL>
```

<value> A real number representing the Y-axis value at the center of the display.

Example

This example returns the current Y-display value to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;"":WAVEFORM:YDISPLAY?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

YINCrement?

Query

```
:WAVeform:YINCrement?
```

The :WAVeform:YINCrement? query returns the y-increment voltage value for the currently specified source. This voltage value is the voltage difference between two adjacent waveform data digital codes. Adjacent digital codes are codes that differ by one least significant bit. For example, the digital codes 24680 and 24681 vary by one least significant bit.

- For BYTE and WORD data, and voltage waveforms, it is the voltage corresponding to one least significant bit change.
- For ASCii data format, the YINCrement is the full scale voltage range covered by the A/D converter.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:YINCrement] <real_value><NL>
```

`<real_value>` A real number in exponential format.

Example

This example places the current Y-increment value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YINCREMENT?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

See Also

For more information on BYTE and WORD formats see "Understanding WORD and BYTE Formats" on page 28-28.

You can also obtain the Y-increment value through the :WAVeform:PREamble? query.

YORigin?

Query

```
:WAVeform:YORigin?
```

The :WAVeform:YORigin? query returns the y-origin voltage value for the currently specified source. The voltage value returned is the voltage value represented by the waveform data digital code 00000.

- For BYTE and WORD data, and voltage waveforms, it is the voltage at digital code zero.
- For ASCii data format, the YORigin is the Y-axis value at the center of the data range. Data range is returned in the Y increment.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:YORigin] <real_value><NL>
```

`<real_value>` A real number in exponential format.

Example

This example places the current Y-origin value in the numeric variable, Center, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YORIGIN?"
30  ENTER 707;Center
40  PRINT Center
50  END
```

See Also

For more information on BYTE and WORD formats see "Understanding WORD and BYTE Formats" on page 28-28.

You can obtain the Y-origin value through the :WAVeform:PREamble? query.

YRANge?

Query

```
:WAVeform:YRANge?
```

The :WAVeform:YRANge? query returns the Y-axis duration of the displayed waveform. For voltage waveforms, it is the voltage across the entire display.

A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format

```
[:WAVeform:YRANge] <value><NL>
```

<value> A real number representing the Y-axis duration of the displayed waveform.

Example

This example returns the current Y-range value to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YRANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

YREFerence?

Query

```
:WAVeform:YREFerence?
```

The :WAVeform:YREFerence? query returns the y-reference voltage value for the currently specified source. It is at this level that the Y origin is defined. In this oscilloscope, the value is always zero.

Returned Format

```
[:WAVeform:YREFerence] 0<NL>>
```

Example

This example places the current Y-reference value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YREFERENCE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

See Also

For more information on BYTE and WORD formats see “Understanding WORD and BYTE Formats” on page 28-28.

You can obtain the Y-reference value through the :WAVeform:PREamble? query.

YUNits?

Query

```
:WAVeform:YUNits?
```

The :WAVeform:YUNits? query returns the Y-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format

```
[:WAVeform:YUNits] {UNKNown | VOLT | SECond | HITS | DECibels
| CONStant | AMP}<NL>
```

Example

This example returns the Y-axis units of the currently selected waveform source to the string variable, Unit$, then prints the contents of the variable to the computer's screen.

```
10  DIM Unit$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:YUNITS?"
30  ENTER 707;Unit$
40  PRINT Unit$
50  END
```

29

Waveform Memory Commands

Waveform Memory Commands

The Waveform Memory Subsystem commands let you save and display waveforms, memories, and functions. These Waveform Memory commands and queries are implemented in the Infiniium Oscilloscopes:

- CLEar
- DISPlay
- LOAD
- SAVE
- XOFFset
- XRANge
- YOFFset
- YRANge

<N> in WMEMory<N> Indicates the Waveform Memory Number

In Waveform Memory commands, the <N> in WMEMory<N> represents the waveform memory number (1-4).

CLEar

Command `:WMEMory<N>:CLEar`

The :WMEMory<N>:CLEar clears the associated wave memory.

<N> The memory number is an integer from 1 to 4.

Example This example clears the waveform memory 1.

```
10  OUTPUT 707;":WMEMORY1:CLEar"
20  END
```

DISPlay

Command

```
:WMEMory<N>:DISPlay {{ON | 1} | {OFF | 0}}
```

The :WMEMory<N>:DISPlay command enables or disables the viewing of the selected waveform memory.

<N> The memory number is an integer from 1 to 4.

Example

This example turns on the waveform memory 1 display.

```
10  OUTPUT 707;":WMEMORY1:DISPLAY ON"
20  END
```

Query

```
:WMEMory<N>:DISPlay?
```

The :WMEMory<N>:DISPlay? query returns the state of the selected waveform memory.

Returned Format

```
[:WMEMory<N>:DISPlay] {1 | 0}<NL>
```

LOAD

Command

```
:WMEMory<N>:LOAD <file_name>
```

The :WMEMory<N>:LOAD command loads an oscilloscope waveform memory location with a waveform from a file that has an internal waveform format (extension .wfm), comma separated xypairs, (extension .csv), tab separated xypairs (extension .tsv), and yvalues text (extension .txt). You can load the file from either the c: or a: drive, or any lan connected drive. See the examples below.

The oscilloscope assumes that the default path for waveforms is c:\Document and Settings\All Users\Shared Documents\Infiniium\Data. To use a different path, specify the path and file name completely.

`<N>` The memory number is an integer from 1 to 4.

`<file_name>` A quoted string which specifies the file to load, and has a .wfm, .csv, .tsv, or .txt extension.

Examples

This example loads waveform memory 4 with a file.

```
10  OUTPUT 707;":WMEMORY4:LOAD ""c:\Document and Settings\All
Users\Shared Documents\Infiniium\Data\waveform.wfm"""
20  END
```

This example loads waveform memory 3 with a file that has the internal waveform format and is stored on the floppy drive.

```
10  OUTPUT 707;":WMEMORY3:LOAD ""a:\waveform.wfm"""
20  END
```

Related Commands

```
:DISK:LOAD
:DISK:STORe
```

SAVE

Command

```
:WMEMory<N>:SAVE {CHANnel<N> |CLOCk | FUNCtion<N> |
MTRend | MSPectrum | WMEMory<N>}
```

The :WMEMory<N>:SAVE command stores the specified channel, waveform memory, or function to the waveform memory. You can save waveforms to waveform memories regardless of whether the waveform memory is displayed or not.

The :WAVeform:VIEW command determines the view of the data being saved.

<N> CHANnel<N> is an integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

Example

This example saves channel 1 to waveform memory 4.

```
10  OUTPUT 707;":WMEMORY4:SAVE CHANNEL1"
20  END
```

XOFFset

Command `:WMEMory<N>:XOFFset <offset_value>`

The :WMEMory<N>:XOFFset command sets the x-axis, horizontal position for the selected waveform memory's display scale. The position is referenced to center screen.

`<N>` The memory number is an integer from 1 to 4.

`<offset_value>` A real number for the horizontal offset (position) value.

Example This example sets the X-axis, horizontal position for waveform memory 3 to 0.1 seconds (100 ms).

```
10  OUTPUT 707;":WMEMORY3:XOFFSET 0.1"
20  END
```

Query `:WMEMory<N>:XOFFset?`

The :WMEMory<N>:XOFFset? query returns the current X-axis, horizontal position for the selected waveform memory.

Returned Format `[:WMEMory<N>:XOFFset] <offset_value><NL>`

XRANge

Command `:WMEMory<N>:XRANge <range_value>`

The :WMEMory<N>:XRANge command sets the X-axis, horizontal range for the selected waveform memory's display scale. The horizontal scale is the horizontal range divided by 10.

`<N>` The memory number is an integer from 1 to 4.

`<range_value>` A real number for the horizontal range value.

Example This example sets the X-axis, horizontal range of waveform memory 2 to 435 microseconds.

```
10  OUTPUT 707;":WMEMORY2:XRANGE 435E-6"
20  END
```

Query `:WMEMory<N>:XRANge?`

The :WMEMory<N>:XRANge? query returns the current X-axis, horizontal range for the selected waveform memory.

Returned Format `[:WMEMory<N>:XRANge] <range_value><NL>`

YOFFset

Command `:WMEMory<N>:YOFFset <offset_value>`

The :WMEMory<N>:YOFFset command sets the Y-axis (vertical axis) offset for the selected waveform memory.

`<N>` The memory number is an integer from 1 to 4.

`<offset_value>` A real number for the vertical offset value.

Example This example sets the Y-axis (vertical) offset of waveform memory 2 to 0.2V.

```
10  OUTPUT 707;":WMEMORY2:YOFFSET 0.2"
20  END
```

Query `:WMEMory<N>:YOFFset?`

The :WMEMory<N>:YOFFset? query returns the current Y-axis (vertical) offset for the selected waveform memory.

Returned Format `[:WMEMory<N>:YOFFset] <offset_value><NL>`

YRANge

Command `:WMEMory<N>:YRANge <range_value>`

The :WMEMory<N>:YRANge command sets the Y-axis, vertical range for the selected memory. The vertical scale is the vertical range divided by 8.

`<N>` The memory number is an integer from 1 to 4.

`<range_value>` A real number for the vertical range value.

Example This example sets the Y-axis (vertical) range of waveform memory 3 to 0.2 volts.

```
10  OUTPUT 707;":WMEMORY3:YRANGE 0.2"
20  END
```

Query `:WMEMory<N>:YRANge?`

The :WMEMory<N>:YRANge? query returns the Y-axis, vertical range for the selected memory.

Returned Format `[:WMEMory<N>:YRANge]<range_value><NL>`

30

Serial Data Equalization Commands

Serial Data Equalization Commands

The N5461A Serial Data Equalization application is used to re-open partially or completely closed real-time eye diagrams. For additional information on equalization, consult the N5461A Infiniium Serial Data Equalization User's Guide. These SPRocessing commands and queries are implemented in the Infiniium Oscilloscopes (you must have the N5461A Infiniium Serial Data Equalization application installed to access these commands):

- CTLequalizer:DISPlay
- CTLequalizer:SOURce
- CTLequalizer:DCGain
- CTLequalizer:P1
- CTLequalizer:P2
- CTLequalizer:RATe
- CTLequalizer:VERTical
- CTLequalizer:VERTical:OFFSet
- CTLequalizer:VERTical:RANGe
- CTLequalizer:ZERo
- FFEQualizer:DISPlay
- FFEQualizer:SOURce
- FFEQualizer:NTAPs
- FFEQualizer:NPRecursor
- FFEQualizer:RATe
- FFEQualizer:TAP
- FFEQualizer:TAP:PLENgth
- FFEQualizer:TAP:WIDTh
- FFEQualizer:TAP:DELay
- FFEQualizer:TAP:AUTomatic
- FFEQualizer:BANDwidth
- FFEQualizer:BWMode
- FFEQualizer:TDELay
- FFEQualizer:TDMode
- FFEQualizer:VERTical

- FFEQualizer:VERTical:OFFSet
- FFEQualizer:VERTical:RANGe
- DFEQualizer:STATe
- DFEQualizer:SOURce
- DFEQualizer:NTAPs
- DFEQualizer:TAP
- DFEQualizer:TAP:WIDTh
- DFEQualizer:TAP:DELay
- DFEQualizer:TAP:MAX
- DFEQualizer:TAP:MIN
- DFEQualizer:TAP:GAIN
- DFEQualizer:TAP:UTARget
- DFEQualizer::TAP:LTARget
- DFEQualizer:TAP:AUTomatic

CTLequalizer:DISPlay

Command

```
:SPRocessing:CTLequalizer:DISPlay {(OFF | 0) | (ON | 1)}
```

The :CTLequalizer:DISPlay command turns the display of a Continuous Time Linear Equalizer (CTLE) real-time eye diagram on or off. Turning CTLE on automatically turns FFE off (and vice versa).

Example

This example turns on the display of a CTLE real-time eye diagram.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:DISPlay ON"
20  END
```

Query

```
:SPRocessing:CTLequalizer:DISPlay?
```

The :SPRocessing:CTLequalizer:DISPlay? query returns whether or not the CTLE real-time eye is displayed.

CTLequalizer:SOURce

Command

```
:SPRocessing:CTLequalizer:SOURce {CHANnel<N> | FUNCtion<N> | WMEMory<N>}
```

The :CTLequalizer:SOURce command sets the source for the Continuous Time Linear Equalization.

<N> CHANnel<N> is an integer, 1- 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1- 4, representing the selected function or waveform memory

Example

This example sets the CTLE source to Channel 1.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:SOURce Channel1"
20  END
```

Query

```
:SPRocessing:CTLequalizer:SOURce?
```

The :SPRocessing:CTLequalizer:SOURce? query returns the CTLE source.

CTLequalizer:DCGain

Command

```
:SPRocessing:CTLequalizer:DCGain <dc_gain>
```

The :CTLequalizer:DCGain command sets the DC Gain parameter for the Continuous Time Linear Equalization.

`<dc_gain>` A real number

Example

This example sets the CTLE DC Gain parameter to 1.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:DCGain 1"
20  END
```

Query

```
:SPRocessing:CTLequalizer:DCGain?
```

The :SPRocessing:CTLequalizer:DCGain? query returns the CTLE's DC Gain parameter.

CTLequalizer:P1

Command

```
:SPRocessing:CTLequalizer:P1 <pole1_freq>
```

The :CTLequalizer:P1 command sets the Pole 1 frequency for the Continuous Time Linear Equalization.

`<pole1_freq>` A real number

Example

This example sets the CTLE Pole 1 frequency to 1GHz.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:P1 1e9"
20  END
```

Query

```
:SPRocessing:CTLequalizer:P1?
```

The :SPRocessing:CTLequalizer:P1? query returns the CTLE's Pole 1 frequency.

CTLequalizer:P2

Command `:SPRocessing:CTLequalizer:P2 <pole2_freq>`

The :CTLequalizer:P1 command sets the Pole 2 frequency for the Continuous Time Linear Equalization.

`<pole2_freq>` A real number

Example This example sets the CTLE Pole 2 frequency to 4 GHz.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:P2 4e9"
20  END
```

Query `:SPRocessing:CTLequalizer:P2?`

The :SPRocessing:CTLequalizer:P2? query returns the CTLE's Pole 2 frequency.

CTLequalizer:RATe

Command `:SPRocessing:CTLequalizer:RATe <data_rate>`

The :CTLequalizer:RATe command sets the data rate for the CTLE equalizer.

`<data_rate>` A real number

Example This example sets the CTLE data rate to 3e9.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:RATe 3e9"
20  END
```

Query `:SPRocessing:CTLequalizer:RATe?`

The :SPRocessing:CTLequalizer:Rate? query returns the CTLE's data rate.

CTLequalizer:VERTical

Command

```
:SPRocessing:CTLequalizer:VERTical {AUTO | MANual}
```

The :SPRocessing:CTLequalizer:VERTIcal command sets the CTLE signal's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the CTLE signal's vertical scale mode to automatic.

```
10 OUTPUT 707;":SPRocessing:CTLequalizer:VERTical AUTO"
20 END
```

Query

```
:SPRocessing:CTLequalizer:VERTical?
```

The :SPRocessing:CTLequalizer:VERTical? query returns the current CTLE signal's vertical scale mode setting.

Returned format

```
[:SPRocessing:CTLequalizer:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the CTLE signal's vertical scale mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":SPRocessing:CTLequalizer:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

CTLequalizer:VERTical:OFFSet

Command

::SPRocessing:CTLequalizer:VERTical:OFFSet <offset>

The ::SPRocessing:CTLequalizer:VERTial:OFFSet command sets the CTLE signal's vertical offset.

<offset> A real number for the CTLE signal's vertical offset.

Example

This example sets the CTLE signal's vertical offset to 1 volt.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:VERTICAL:OFFSET 1"
20  END
```

Query

:SPRocessing:CTLequalizer:VERTical:OFFSet?

The:SPRocessing:CTLequalizer:VERTIcal:OFFSet? query returns the CTLE signal's vertical offset setting.

Returned format

[:SPRocessing:CTLequalizer:VERTical:OFFSet] <value><NL>

<value> The CTLE signal's vertical offset setting.

Example

This example places the current value of the CTLE signal's vertical offset in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":SPRocessing:CTLequalizer:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

CTLequalizer:VERTical:RANGe

Command

:SPRocessing:CTLequalizer:VERTical:RANGe <range>

The :SPRocessing:CTLequalizer:VERTial:RANGe command sets the CTLE signal's vertical range.

<range> A real number for the full-scale CTLE signal's vertical range.

Example

This example sets the CTLE signal's vertical range to 16 volts (2 volts times 8 divisions.)

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:VERTICAL:RANGE 16"
20  END
```

Query

:SPRocessing:CTLequalizer:VERTical:RANGe?

The :SPRocessing:CTLequalizer:VERTical:RANGe? query returns the CTLE signal's vertical range setting.

Returned Format

[:SPRocessing:CTLequalizer:VERTical:RANGe] <value><NL>

<value> The CTLE signal's vertical range setting.

Example

This example places the current value of the CTLE signal's vertical range in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":SPRocessing:CTLequalizer:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

CTLequalizer:ZERo

Command

```
:SPRocessing:CTLequalizer:ZERo <zero_freq>
```

The :CTLequalizer:ZERo command sets the zero frequency for the Continuous Time Linear Equalization.

`<zero_freq>` A real number

Example

This example sets the CTLE zero frequency to 900 MHz.

```
10  OUTPUT 707;":SPRocessing:CTLequalizer:ZERo 9e6"
20  END
```

Query

```
:SPRocessing:CTLequalizer:ZERo?
```

The :SPRocessing:CTLequalizer:ZERo? query returns the CTLE's zero frequency.

FFEQualizer:DISPlay

Command

```
:SPRocessing:FFEQualizer:DISPlay {(OFF | 0) | (ON | 1)}
```

The :FFEQualizer:DISPlay command turns the display of a Feed-Forward Equalized (FFE) real-time eye diagram on or off.

Example

This example turns on the display of a FFE real-time eye diagram.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:DISPlay ON"
20  END
```

Query

```
:SPRocessing:FFEQualizer:DISPlay?
```

The :SPRocessing:FFEQualizer:DISPlay? query returns whether or not the FFE real-time eye is displayed.

FFEQualizer:SOURce

Command

```
:SPRocessing:FFEQualizer:SOURce {CHANnel<N> | FUNCtion<N> | WMEMory<N>}
```

The :FFEQualizer:SOURce command sets the source for the Feed-Forward Equalization.

<N> CHANnel<N> is an integer, 1- 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1- 4, representing the selected function or waveform memory

Example

This example sets the FFE source to Channel 1.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:SOURce Channel1"
20  END
```

Query

```
:SPRocessing:FFEQualizer:SOURce?
```

The :SPRocessing:FFEQualizer:SOURce? query returns the FFE source.

FFEQualizer:NPRecursor

Command

```
:SPRocessing:FFEQualizer:NPRecursor <number>
```

The :FFEQualizer:NPRecursor command sets the number of precursor taps to be used in the FFE algorithm.

`<number>` An integer between 1 and (NTAPs - 1)

Example

This example sets the number of FFE precursor taps to 3.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:NPRecursor 3"
20  END
```

Query

```
:SPRocessing:FFEQualizer:NPRecursor?
```

The :SPRocessing:FFEQualizer:NPRecursor? query returns the number of FFE precursor taps.

FFEQualizer:NTAPs

Command

```
:SPRocessing:FFEQualizer:NTAPs <number>
```

The :FFEQualizer:NTAPs command sets the number of taps to be used in the FFE algorithm.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

`<number>` An integer between 2 and 40

Example

This example sets the number of FFE taps to 3.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:NTAPs 3"
20  END
```

Query

```
:SPRocessing:FFEQualizer:NTAPs?
```

The :SPRocessing:FFEQualizer:NTAPs? query returns the number of FFE taps.

FFEequalizer:RATe

Command `:SPRocessing:FFEequalizer:RATe <data_rate>`

The :FFEequalizer:RATe command sets the data rate for the FFE equalizer.

`<data_rate>` A real number

Example This example sets the FFE data rate to 3e9.

```
10  OUTPUT 707;":SPRocessing:FFEequalizer:RATe 3e9"
20  END
```

Query `:SPRocessing:FFEequalizer:RATe?`

The :SPRocessing:FFEequalizer:Rate? query returns the FFEE's data rate.

FFEQualizer:TAP

Command

```
:SPRocessing:FFEQualizer:TAP <tap>, <value>
```

The :FFEQualizer:TAP command sets the tap value for each FFE tap. For example, when <tap> is equal to 0 then the 0th tap is set to <value>.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

`<tap>` The tap number

`<value>` The tap value

Example

This example sets the second FFE tap to -1.432.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:TAP 2,-1.432"
20  END
```

Query

```
:SPRocessing:FFEQualizer:TAP?
```

The :SPRocessing:FFEQualizer:TAP? query returns the FFE tap values.

FFEQualizer:TAP:PLENgth

Command

```
:SPRocessing:FFEQualizer:TAP:PLENgth {CUSTom | PRBS51 | PRBS61 | PRBS71 | PRBS81 | PRBS91 | PRBS101},
<file>
```

In order for the tap optimization to work, the algorithm must know the input pattern. You can train the oscilloscope to a known pattern and then use the optimized taps on your live traffic. The :FFEQualizer:TAP:PLENgth command sets the pattern for the FFE tap optimization algorithm.

The file parameter is only used in CUSTom mode.

For more information on this parameter, refer to the N5461A Infiniium Seriald Data Equalization User's Guide.

<CUSTom>	Allows you to import a known pattern via a .prtn file (save at location <file>)
<PRBSX1>	Pseudo-random Binary Sequence of length 2^X - 1.

Example

This example sets the pattern to PRBS 2^8 - 1.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:TAP:PLENgth PRBS81"
20  END
```

Query

```
:SPRocessing:FFEQualizer:TAP:PLENgth?
```

The :SPRocessing:FFEQualizer:TAP:PLENgth? query returns the pattern used in optimizing the FFE tap values.

FFEQualizer:TAP:WIDTh

Command

```
:SPRocessing:FFEQualizer:TAP:WIDTh <width>
```

The :FFEQualizer:TAP:WIDTh command sets the Eye Width field for the FFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for FFE is 0.33. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

`<width>` A real number between 0.0 and 1.0.

Example

This example sets the eye width to 0.0.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:TAP:WIDTh 0.0"
20  END
```

Query

```
:SPRocessing:FFEQualizer:TAP:WIDTh?
```

The :SPRocessing:FFEQualizer:TAP:WIDTh? query returns the eye width used in the FFE tap optimization.

FFEQualizer:TAP:DELay

Command `:SPRocessing:FFEQualizer:TAP:DELay <delay>`

The :FFEQualizer:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

`<delay>` A real number

Query `:SPRocessing:FFEQualizer:TAP:DELay?`

The :SPRocessing:FFEQualizer:TAP:DELay? query returns the value for the FFE Delay field.

FFEQualizer:TAP:AUTomatic

Command

```
:SPRocessing:FFEQualizer:TAP:AUTomatic
```

The :FFEQualizer:TAP:AUTomatic command starts the FFE tap optimization. Be sure to first specify the number of taps and specify the Pattern and Eye Width parameters.

Example

This example starts the FFE tap optimization.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:TAP:AUTomatic"
20  END
```

Query

There is no query for this command.

FFEQualizer:TAP :BANDwidth

Command `:SPRocessing:FFEQualizer:TAP:BANDwidth <bandwidth>`

The :FFEQualizer:TAP:BANDwidth command is only needed if the FFEQualizer:TAP:BWMode command is set to CUSTom and in this case it sets the bandwidth at which the response generated by equalization rolls off. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

`<bandwidth>` The bandwidth at which the response generated by equalization rolls off.

Query `:SPRocessing:FFEQualizer:TAP:BANDwidth?`

The :SPRocessing:FFEQualizer:TAP:BANDwidth? query returns the current value for the BANDwidth parameter.

FFEQualizer:TAP :BWMode

Command

```
:SPRocessing:FFEQualizer:TAP:BWMode {TSBandwidth | TTDelay | CUSTom}
```

The :FFEQualizer:TAP:BWMode command sets the bandwidth at which the response generated by equalization is rolled off. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

Example

This example sets the FFE Bandwidth Mode to TTDELay.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:TAP:BWMode TTDelay"
20  END
```

Query

```
:SPRocessing:FFEQualizer:TAP:BWMode?
```

The :SPRocessing:FFEQualizer:TAP:BWMode? query returns the FFE Bandwidth Mode.

FFEQualizer:TAP :TDELay

Command `:SPRocessing:FFEQualizer:TAP:TDELay <delay_value>`

The :FFEQualizer:TAP:TDELay command is only needed if the FFEQualizer:TAP:TDMode is set to CUSTom. To determine what this value should be, use the equation: tap delay = 1/[(data rate)x(# of taps per bit)]. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

`<delay_value>` A real number

Query `:SPRocessing:FFEQualizer:TAP:TDELay?`

The :SPRocessing:FFEQualizer:TAP:TDELay? query returns the current value for the tap delay.

FFEQualizer:TAP :TDMode

Command

```
:SPRocessing:FFEQualizer:TAP:TDMode {TBITrate | CUSTom}
```

The :FFEQualizer:TAP:TDMode command sets Tap Delay field to either Track Data Rate or Custom. If you are using one tap per bit, use the TBITrate selection. If you are using multiple taps per bit, use CUSTom and then use the FFEQualizer:TAP:TDELay command to set the value. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

Example

This example sets the FFE Tap Delay mode to TBITrate.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:TAP:TDMode TBITrate"
20  END
```

Query

```
:SPRocessing:FFEQualizer:TAP:TDMode?
```

The :SPRocessing:FFEQualizer:TAP:TDMode? query returns the current Tap Delay mode.

FFEQualizer:VERTical

Command

```
:SPRocessing:FFEQualizer:VERTical {AUTO | MANual}
```

The :SPRocessing:FFEQualizer:VERTIcal command sets the FFE signal's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the FFEE signal's vertical scale mode to automatic.

```
10 OUTPUT 707;":SPRocessing:FFEQualizer:VERTical AUTO"
20 END
```

Query

```
:SPRocessing:FFEQualizer:VERTical?
```

The :SPRocessing:FFEQualizer:VERTical? query returns the current FFE signal's vertical scale mode setting.

Returned format

```
[:SPRocessing:FFEQualizer:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the FFE signal's vertical scale mode in the string variable Setting$, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":SPRocessing:FFEQualizer:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

FFEQualizer:VERTical:OFFSet

Command ::SPRocessing:FFEQualizer:VERTical:OFFSet <offset>

The ::SPRocessing:FFEQualizer:VERTial:OFFSet command sets the FFE signal's vertical offset.

<offset> A real number for the FFE signal's vertical offset.

Example This example sets the FFE signal's vertical offset to 1 volt.

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:VERTICAL:OFFSET 1"
20  END
```

Query :SPRocessing:FFEQualizer:VERTical:OFFSet?

The:SPRocessing:FFEQualizer:VERTIcal:OFFSet? query returns theFFE signal's vertical offset setting.

Returned format [:SPRocessing:FFEQualizer:VERTical:OFFSet] <value><NL>

<value> The FFE signal's vertical offset setting.

Example This example places the current value of the FFE signal's vertical offset in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":SPRocessing:FFEQualizer:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

FFEQualizer:VERTical:RANGe

Command

:SPRocessing:FFEQualizer:VERTical:RANGe <range>

The :SPRocessing:FFEQualizer:VERTial:RANGe command sets the FFE signal's vertical range.

<range> A real number for the full-scale FFE signal's vertical range.

Example

This example sets the FFE signal's vertical range to 16 volts (2 volts times 8 divisions.)

```
10  OUTPUT 707;":SPRocessing:FFEQualizer:VERTICAL:RANGE 16"
20  END
```

Query

:SPRocessing:FFEQualizer:VERTical:RANGe?

The :SPRocessing:FFEQualizer:VERTical:RANGe? query returns the FFE signal's vertical range setting.

Returned Format

[:SPRocessing:FFEQualizer:VERTical:RANGe] <value><NL>

<value> The FFE signal's vertical range setting.

Example

This example places the current value of the FFE signal's vertical range in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":SPRocessing:FFEQualizer:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

DFEQualizer:STATe

Command

```
:SPRocessing:DFEQualizer:STATe {(OFF | 0) | (ON | 1)}
```

The :DFEQualizer:STATe command turns the Decision Feedback Equalizion on or off.

Example

This example turns on DFE.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:STATe ON"
20  END
```

Query

```
:SPRocessing:DFEQualizer:STATe?
```

The :SPRocessing:DFEQualizer:STATe? query returns whether or not DFE is turned on.

DFEQualizer:SOURce

Command

```
:SPRocessing:FFEQualizer:SOURce {CHANnel<N> | FUNCtion<N> | WMEMory<N> | EQUalized}
```

The :DFEQualizer:SOURce command sets the source for the Decision Feedback Equalization.

Setting the source to EQUalized means the Feed-Forward Equalized (FFE) waveform is used as the DFE source.

<N> CHANnel<N> is an integer, 1- 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1- 4, representing the selected function or waveform memory

Example

This example sets the DFE source to Channel 1.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:SOURce Channel1"
20  END
```

Query

```
:SPRocessing:DFEQualizer:SOURce?
```

The :SPRocessing:DFEQualizer:SOURce? query returns the DFE source.

DFEQualizer:NTAPs

Command

```
:SPRocessing:DFEQualizer:NTAPs <number>
```

The :DFEQualizer:NTAPs command sets the number of taps to be used in the DFE algorithm.

DFE tap indices always begin with 1 and extend to the number of taps.

`<number>` An integer between 2 and 40

Example

This example sets the number of DFE taps to 3.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:NTAPs 3"
20  END
```

Query

```
:SPRocessing:DFEQualizer:NTAPs?
```

The :SPRocessing:DFEQualizer:NTAPs? query returns the number of DFE taps.

DFEQualizer:TAP

Command

```
:SPRocessing:DFEQualizer:TAP <tap>, <value>
```

The :DFEQualizer:TAP command sets the tap value for each DFE tap. For example, when <tap> is equal to 0 then the 0th tap is set to <value>.

DFE tap indices always begin with 1 and extend to the number of taps.

`<tap>` The tap number

`<value>` The tap value

Example

This example sets the second DFE tap to -1.432.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP 2,-1.432"
20  END
```

Query

```
:SPRocessing:DFEQualizer:TAP?
```

The :SPRocessing:DFEQualizer:TAP? query returns the DFE tap values.

DFEQualizer:TAP:WIDTh

Command

```
:SPRocessing:DFEQualizer:TAP:WIDTh <width>
```

The :DFEQualizer:TAP:WIDTh command sets the Eye Width field for the DFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for DFE is 0.0. For more information on this parameter, refer to the N5461A Infiniium Seriald Data Equalization User's Guide.

`<width>` A real number between 0.0 and 1.0.

Example

This example sets the eye width to 0.0.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:WIDTh 0.0"
20  END
```

Query

```
:SPRocessing:DFEQualizer:TAP:WIDTh?
```

The :SPRocessing:DFEQualizer:TAP? query returns the eye width used in the DFE tap optimization.

DFEQualizer:TAP:DELay

Command `:SPRocessing:DFEQualizer:TAP:DELay <delay>`

The :DFEQualizer:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the N5461A Infiniium Seriald Data Equalization User's Guide.

`<delay>` A real number

Query `:SPRocessing:DFEQualizer:TAP:DELay?`

The :SPRocessing:DFEQualizer:TAP:DELay? query returns the value for the DFE Delay field.

DFEQualizer:TAP:MAX

Command `:SPRocessing:DFEQualizer:TAP:MAX <max_tap_value>`

Some standards have upper and lower limits on the tap values. The :DFEQualizer:TAP:MAX command sets the upper limit on taps determined through optimization.

`<max_tap_value>` A real number

Example This example sets the Upper Limit field to 3.23.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:MAX 3.23"
20  END
```

Query `:SPRocessing:DFEQualizer:TAP:MAX?`

The :SPRocessing:DFEQualizer:TAP:MAX? query returns the Upper Limit used in the DFE tap optimization.

DFEQualizer:TAP:MIN

Command `:SPRocessing:DFEQualizer:TAP:MIN <min_tap_value>`

Some standards have upper and lower limits on the tap values. The :DFEQualizer:TAP:MIN command sets the lower limit on taps determined through optimization.

`<min_tap_value>` A real number

Example This example sets the Lower Limit field to 3.23.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:MIN 3.23"
20  END
```

Query `:SPRocessing:DFEQualizer:TAP:MIN?`

The :SPRocessing:DFEQualizer:TAP:MIN? query returns the Lower Limit used in the DFE tap optimization.

DFEQualizer:TAP:GAIN

Command `:SPRocessing:DFEQualizer:TAP:GAIN <gain>`

The eye diagram drawn after DFE is applied is attenuated. To amplify the eye back to its original size (so you can directly compare the eye at the receiver to the eye at the transmitter), a gain factor needs to be applied. The :DFEQualizer:TAP:GAIN command allows you to set this gain. For more information on this parameter, refer to the N5461A Infiniium Seriald Data Equalization User's Guide.

`<gain>` A real number

Example This example sets the gain to 3.23.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:GAIN 3.23"
20  END
```

Query `:SPRocessing:DFEQualizer:TAP:GAIN?`

The :SPRocessing:DFEQualizer:TAP:GAIN? query returns the current gain value.

DFEQualizer:TAP:UTARget

Command

```
:SPRocessing:DFEQualizer:TAP:UTARget <upper_target>
```

The Upper Target field dictates the logical high value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical high, its value will be equal to Upper Target. The :DFEQualizer:TAP:UTARget command allows you to set this value.

`<upper_target>` A real number

Example

This example sets the Upper Target to 1.0.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:UTARget 1.0"
20  END
```

Query

```
:SPRocessing:DFEQualizer:TAP:UTARget?
```

The :SPRocessing:DFEQualizer:TAP:UTARget? query returns the current value for the Upper Target field.

DFEQualizer:TAP:LTARget

Command

```
:SPRocessing:DFEQualizer:TAP:LTARget <lower_target>
```

The Lower Target field dictates the logical low value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical low, its value will be equal to Lower Target. The :DFEQualizer:TAP:LTARget command allows you to set this value.

`<lower_target>` A real number

Example

This example sets the Lower Target to 1.0.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:LTARget 1.0"
20  END
```

Query

```
:SPRocessing:DFEQualizer:TAP:LTARget?
```

The :SPRocessing:DFEQualizer:TAP:LTARget? query returns the current value for the Lower Target field.

DFEQualizer:TAP:AUTomatic

Command

```
:SPRocessing:DFEQualizer:TAP:AUTomatic
```

The :DFEQualizer:TAP:AUTomatic command starts the DFE tap optimization. Be sure to first specify the number of taps and the max/min tap values.

Example

This example starts the DFE tap optimization.

```
10  OUTPUT 707;":SPRocessing:DFEQualizer:TAP:AUTomatic"
20  END
```

Query

There is no query for this command.

Error Messages

Error Messages

This chapter describes the error messages and how they are generated. The possible causes for the generation of the error messages are also listed in the following table.

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error -350, “Queue overflow.” Anytime the error queue overflows, the oldest errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the “Queue overflow” message). Reading an error from the head of the queue removes that error from the queue, and opens a position at the tail of the queue for a new error. When all errors have been read from the queue, subsequent error queries return 0, “No error.”

The error queue is cleared when any of the following occur:

- the instrument is powered up,
- a *CLS command is sent,
- the last item from the queue is read, or
- the instrument is switched from talk only to addressed mode on the front panel.

Error Numbers

The error numbers are grouped according to the type of error that is detected.

- +0 indicates no errors were detected.
- -100 to -199 indicates a command error was detected
- -200 to -299 indicates an execution error was detected.
- -300 to -399 indicates a device-specific error was detected.
- -400 to-499 indicates a query error was detected.
- +1 to +32767 indicates an oscilloscope specific error has been detected.

Command Error

An error number in the range -100 to -199 indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class sets the command error bit (bit 5) in the event status register and indicates that one of the following events occurred:

- An IEEE 488.2 syntax error was detected by the parser. That is, a computer-to-oscilloscope message was received that is in violation of the IEEE 488.2 standard. This may be a data element that violates the oscilloscope's listening formats, or a data type that is unacceptable to the oscilloscope.
- An unrecognized header was received. Unrecognized headers include incorrect oscilloscope-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 program message.

Events that generate command errors do not generate execution errors, oscilloscope-specific errors, or query errors.

Execution Error

An error number in the range -200 to -299 indicates that an error was detected by the instrument's execution control block. The occurrence of any error in this class causes the execution error bit (bit 4) in the event status register to be set. It also indicates that one of the following events occurred:

- The program data following a header is outside the legal input range or is inconsistent with the oscilloscope's capabilities.
- A valid program message could not be properly executed due to some oscilloscope condition.

Execution errors are reported by the oscilloscope after expressions are evaluated and rounding operations are completed. For example, rounding a numeric data element will not be reported as an execution error. Events that generate execution errors do not generate command errors, oscilloscope specific errors, or query errors.

Device- or Oscilloscope-Specific Error

An error number in the range of -300 to -399 or +1 to +32767 indicates that the instrument has detected an error caused by an oscilloscope operation that did not properly complete. This may be due to an abnormal hardware or firmware condition. For example, this error may be generated by a self-test response error, or a full error queue. The occurrence of any error in this class causes the oscilloscope-specific error bit (bit 3) in the event status register to be set.

Query Error

An error number in the range-400 to-499 indicates that the output queue control of the instrument has detected a problem with the message exchange protocol. An occurrence of any error in this class should cause the query error bit (bit 2) in the event status register to be set. An occurrence of an error also means one of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending.
- Data in the output queue has been lost.

List of Error Messages

Table 25-1 a list of the error messages that are returned by the parser on this oscilloscope.

Table 31-1 Error Messages

0	**No error**	**The error queue is empty. Every error in the queue has been read (SYSTEM:ERROR? query) or the queue was cleared by power-up or *CLS.**
-100	**Command error**	**This is the generic syntax error used if the oscilloscope cannot detect more specific errors.**
-101	**Invalid character**	**A syntactic element contains a character that is invalid for that type.**
-102	**Syntax error**	**An unrecognized command or data type was encountered.**
-103	**Invalid separator**	**The parser was expecting a separator and encountered an illegal character.**
-104	**Data type error**	**The parser recognized a data element different than one allowed. For example, numeric or string data was expected but block data was received.**
-105	**GET not allowed**	**A Group Execute Trigger was received within a program message.**
-108	**Parameter not allowed**	**More parameters were received than expected for the header.**
-109	**Missing parameter**	**Fewer parameters were received than required for the header.**
-112	**Program mnemonic too long**	**The header or character data element contains more than twelve characters.**
-113	**Undefined header**	**The header is syntactically correct, but it is undefined for the oscilloscope. For example, *XYZ is not defined for the oscilloscope.**
-121	**Invalid character in number**	**An invalid character for the data type being parsed was encountered. For example, a “9” in octal data.**
-123	**Numeric overflow**	**Number is too large or too small to be represented internally.**
-124	**Too many digits**	**The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros.**
-128	**Numeric data not allowed**	**A legal numeric data element was received, but the oscilloscope does not accept one in this position for the header.**
-131	**Invalid suffix**	**The suffix does not follow the syntax described in IEEE 488.2 or the suffix is inappropriate for the oscilloscope.**
-138	**Suffix not allowed**	**A suffix was encountered after a numeric element that does not allow suffixes.**
-141	**Invalid character data**	**Either the character data element contains an invalid character or the particular element received is not valid for the header.**
-144	**Character data too long**	
-148	**Character data not allowed**	**A legal character data element was encountered where prohibited by the oscilloscope.**
-150	**String data error**	**This error can be generated when parsing a string data element. This particular error message is used if the oscilloscope cannot detect a more specific error.**

-151	Invalid string data	A string data element was expected, but was invalid for some reason. For example, an END message was received before the terminal quote character.
-158	String data not allowed	A string data element was encountered but was not allowed by the oscilloscope at this point in parsing.
-160	Block data error	This error can be generated when parsing a block data element. This particular error message is used if the oscilloscope cannot detect a more specific error.
-161	Invalid block data	
-168	Block data not allowed	A legal block data element was encountered but was not allowed by the oscilloscope at this point in parsing.
-170	Expression error	This error can be generated when parsing an expression data element. It is used if the oscilloscope cannot detect a more specific error.
-171	Invalid expression	
-178	Expression data not allowed	Expression data was encountered but was not allowed by the oscilloscope at this point in parsing.
-200	Execution error	This is a generic syntax error which is used if the oscilloscope cannot detect more specific errors.
-212	Arm ignored	
-213	Init ignored	
-214	Trigger deadlock	
-215	Arm deadlock	
-220	Parameter error	
-221	Settings conflict	
-222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value is outside the legal range defined by the oscilloscope.
-223	Too much data	Indicates that a legal program data element of block, expression, or string type was received that contained more data than the oscilloscope could handle due to memory or related oscilloscope-specific requirements.
-224	Illegal parameter value	
-230	Data corrupt or stale	
-231	Data questionable	
-240	Hardware error	
-241	Hardware missing	
-250	Mass storage error	
-251	Missing mass storage	
-252	Missing media	
-253	Corrupt media	
-254	Media full	
-255	Directory full	
-256	File name not found	
-257	File name error	

-258	**Media protected**	
-260	**Expression error**	
-261	**Math error in expression**	
-300	**Device specific error**	
-310	**System error**	**Indicates that a system error occurred.**
-311	**Memory error**	
-312	**PUD memory error**	
-313	**Calibration memory lost**	
-314	**Save/recall memory lost**	
-315	**Configuration memory lost**	
-321	**Out of memory**	
-330	**Self-test failed**	
-350	**Queue overflow**	**Indicates that there is no room in the error queue and an error occurred but was not recorded.**
-370	**No sub tests are defined for the selected self test**	
-371	**Self Test status is corrupt or no self test has been executed**	
-372	**This product configuration does not support the requested self test**	
-373	**This product configuration does not support the requested source**	
-374	**The requested self test log file could not be found**	
-375	**Attenuator relay actuation counts can only be modified during factory service**	
-400	**Query error**	**This is the generic query error.**
-410	**Query INTERRUPTED**	
-420	**Query UNTERMINATED**	
-430	**Query DEADLOCKED**	
-440	**Query UNTERMINATED after indefinite response**	

Index

Symbols

...
    Ellipsis 1-5

Numerics

707 1-20
9.99999E+37
    Infinity Representation 6-12

A

Aborting a digitize operation 2-12
aborting a digitize operation 1-18
ABSolute 15-5
absolute voltage
    and VMAX 22-190
    and VMIN 22-194
accuracy and probe calibration 10-4
Acquire Commands 8-2
    AVERage 8-3
    BANdwidth 8-8
    COMPlete 8-5
    COMPlete STATe 8-7
    COUNt 8-4
    INTerpolate 8-9
    MODE 8-10
    POINts 8-12
    POINts AUTO 8-16, 8-17
    SRATe 8-21
    SRATe AUTO 8-24
acquisition
    ACQuire AVER and completion 8-5
    points 8-12
    record length 8-12
    sample program 7-7
    sample rate 8-21
ADD 15-6
address, GPIB default 2-7
advisory line, reading and writing to 25-2
AER? 23-4, 23-5
algebraic sum of functions 15-6
ALIGn 21-4
AlignFIT 21-5
ALL, and VIEW 28-49
alphanumeric
    characters in embedded string 1-13
    strings 1-11
AMPS as vertical units 11-47, 11-58
AND
    ENABle 27-6
AND:ENABle:SOURce 27-7
AREA 16-3, 22-9, 22-29, 22-99
Arm Event Register
    ARM bit 12-21
Arming the trigger 2-12
ASCII
    and FORMat 28-32
    character 32 1-5
    linefeed 1-13
ATER? 23-6
AttenSET?
    in self-test commands 24-3
attenuation factor for probe 10-4, 11-34
AUTO 8-24, 21-15
automatic measurements
    sample program 7-8
AUToscale 23-7, 23-8, 23-9, 23-10
    during initialization 1-15
Aux Out connector 10-6
availability of measured data 4-2
AVERage 8-3, 15-7, 21-16
    and acquisition completion 8-5
    and count 8-4, 21-17
AXIS 17-4

B

BANDpass query 28-6, 30-4, 30-5, 30-6, 30-7, 30-8, 30-9, 30-13, 30-14, 30-15, 30-16, 30-17, 30-18, 30-19, 30-20, 30-21, 30-22, 30-23, 30-24, 30-25, 30-26, 30-27, 30-31, 30-32, 30-33, 30-34, 30-35, 30-36, 30-37, 30-38, 30-39, 30-40, 30-41, 30-42
bandwidth limit 28-6
basic command structure 1-16
basic operations 1-2
BASIC sample programs 7-2
BEEP 23-10
BIND
    in MTESt SCALe command 21-38
Bit Definitions in Status Reporting 4-3
BLANk 23-12
    and VIEW 23-33
blanking the user text area 14-21
block data 1-4, 1-21
    in learnstring 1-4
Block Diagram
    Status Reporting Overview 4-3
Braces 1-5
Brackets
    Square 1-5
buffer, output 1-10, 1-19
buffered responses 6-12
Bus Activity, Halting 2-12
Bus Commands 2-12
BWIDth 22-10
    in TRIG COMM 27-8
BYTE
    and FORMat 28-33
    Understanding the format 28-28
BYTeorder 28-7
    and DATA 28-14

C

C Program
    DATA? Analog Channels 28-15
C sample programs 7-2
Calibration Commands 10-2, 10-5
    OUTPut 10-6
    SKEW 10-7
    STATus? 10-8
calibration status 10-8
CANCel
    in self-test command 24-3
CDIRectory 13-3
CDISplay (Clear DISplay) 23-13
center screen voltage 11-31, 11-33, 11-45
CGRade 14-3, 14-20
Channel Commands 11-2
    DISPlay 11-4, 11-5, 11-6, 11-7, 11-8, 11-9, 11-11, 11-12, 11-14
    EADapter 11-38
    ECoupling 11-40
    INPut 11-16, 11-17, 11-19, 11-21, 11-23, 11-24, 11-25, 11-27, 11-29, 11-32
    OFFSet 11-31, 11-33
    PROBe 11-34
    PROBe ATTenuation 11-36, 11-37
    PROBe EXTernal 11-42
    PROBe EXTernal GAIN 11-43
    PROBe EXTernal OFFSet 11-45
    PROBe EXTernal UNITs 11-47
    PROBe GAIN 11-49, 11-50, 11-51, 11-52

PROBe ID? 11-53
PROBe SKEW 11-54
PROBe STYPe 11-55
RANGe 11-56
SCALe 11-57
UNITs 11-58
CHANnel PROBe ID? 11-53
channels, and VIEW 28-49
channel-to-channel skew factor 10-7
character program data 1-11
cleaning the instrument Notices-1
CLEar 22-21
Clearing
Buffers 2-12
Pending Commands 2-12
clearing
DONE bit 4-17
error queue 4-18, 31-3
registers and queues 4-19
Standard Event Status Register 4-11, 12-7
status data structures 12-4
TRG bit 4-10, 4-17
clipped waveforms, and measurement error 22-8
CLOCk 22-22
and STATe 27-60
in TRIG STATe 27-60
CLOCk METHod 22-23
CLOCk VERTical 22-26, 30-10, 30-28
CLOCk VERTical RANGe 22-28, 30-12, 30-30
*CLS (Clear Status) 12-4
CME bit 12-6, 12-8
COLumn 14-7
combining
commands in same subsystem 1-8
long- and short-form headers 1-11
combining compound and simple commands 1-14
Command
*ESE 12-5
ABSolute 15-5
ADD 15-6
AER? 23-4, 23-5
ALIGn 21-4
AlignFIT 21-5
AMASk CREate 21-7
AMASk SAVE|STORe 21-9
AMASk SOURce 21-8
AMASk UNITs 21-10
AMASk XDELta 21-11
AMASk YDELta 21-13
AREA 16-3, 22-9, 22-29, 22-99
ATER? 23-6
AUTO 21-15
AUToscale 23-7, 23-8, 23-9, 23-10
AVERage 8-3, 15-7, 21-16
AVERage COUNt 21-17
AXIS 17-4
BANdwidth 8-8
BEEP 23-11
BLANk 23-12
BWIDth 22-10
CANCel 24-3
CDIRectory 13-3
CDISplay 23-13
CGRade 14-3, 14-20
LEVels? 14-5
CGRade CROSsing 22-13
CGRade DCDistortion 22-14
CGRade EHEight 22-15
CGRade EWIDth 22-16
CGRade JITTer 22-19
CGRade QFACtor 22-20
CHANnel PROBe ID? 11-53
CLEar 22-21
CLear Status 12-4
CLOCk 22-22
CLOCk METHod 22-23
CLOCk VERTical 22-26, 30-10, 30-28
CLOCk VERTical RANGe 22-28, 30-12, 30-30
COLumn 14-7
COMMonmode 15-8
COMPlete 8-5
COMPlete STATe 8-7
CONNect 14-8, 28-42, 28-47
COPY 13-4
COUNt 8-4
COUNt FAILures? 21-18, 21-19, 21-21
COUNt FWAVeforms? 21-20
COUNt WAVeforms? 21-22
CTCDutycycle 22-30
CTCJitter 22-32
CTCNwidth 22-34
CTCPwidth 22-36
CURSor? 20-3
DATA? 14-9
DATarate 22-11, 22-38, 22-206
DATE 25-3
DEBug 25-4
DELay 18-3
DELete 13-5, 21-23
DELTatime 22-41
DIFF 15-9
DIGitize 1-17, 23-14
DIRectory? 13-6
DISPlay 11-4, 11-5, 11-6, 11-7, 11-8, 11-9, 11-11, 11-12, 11-14, 15-10, 29-3, 29-4
DIVide 15-11
DPRinter 16-4
DSP 25-6
DUTYcycle 22-45
EADapter 11-38
ECoupling 11-40
EDGE 18-9
ENABle 21-24
ERRor? 25-7
Event Status Enable 12-5
FACTors 16-6
FAIL 18-4, 19-3
FALLtime 22-47
FFT DFRequency 22-49
FFT FREQuency 15-12, 22-53
FFT MAGNitude 22-55
FFT PEAK1 22-57
FFT PEAK2 22-58
FFT REFerence 15-13
FFT RESolution 15-14
FFT THReshold 22-59
FFT WINDow 15-15
FFTMagnitude 15-17
FFTPhase 15-18
FOLDing 21-25
FOLDing:BITS 21-26
FREQuency 22-60
GPIB Mode 2-6
GRATicule 14-10
GRATicule INTensity 14-10
HAMPlitude 21-27
HEADer 25-8, 25-10
HIGHpass 15-19, 15-25

HISTogram HITS 22-63
HISTogram M1S 22-65
HISTogram M2S 22-67
HISTogram M3S 22-69
HOLDtime 22-78
HORizontal 15-20
HORizontal POSition 15-21
HORizontal RANGe 15-22
HYSTeresis 18-10
HYSTersis 18-12
IMAGe 16-7
IMPedance 21-28
INPut 11-16, 11-17, 11-19, 11-21, 11-23, 11-24, 11-25, 11-27, 11-29, 11-32
INTegrate 15-23
INTerpolate 8-9
INVert 15-24, 21-30
JITTer HISTogram 22-81
JITTer MEASurement 22-82
JITTer SPECtrum 22-83
JITTer SPECtrum HORizontal 22-84
JITTer SPECtrum HORizontal POSition 22-85
JITTer SPECtrum HORizontal RANGe 22-87
JITTer SPECtrum VERTical 22-88
JITTer SPECtrum VERTical OFFSet 22-89
JITTer SPECtrum VERTical RANGe 22-90
JITTer SPECtrum WINDow 22-91
JITTer STATistics 22-92
JITTer TRENd 22-93
JITTer TRENd SMOoth 22-94
JITTer TRENd SMOoth POINts 22-95
JITTer TRENd VERTical 22-96
JITTer TRENd VERTical OFFSet 22-27, 22-97, 30-11, 30-29
JITTer TRENd VERTical RANGe 22-98
LAMPlitude 21-31
LINE 14-13
LLEVel 18-13
LLIMit 18-5, 19-4
LOAD 13-7, 21-32, 29-5
LONGform 25-11
MAGNify 15-26
MAXimum 15-27
MDIRectory 13-8
MEASure FFT DMAGnitude 22-51
MEASurement 18-6, 19-5
MINimum 15-28
MODE 8-10, 17-5, 18-8, 18-18, 20-4
MODel? 23-17
MULTiply 15-29
NCJitter 22-100
NREGions? 21-33
NWIDth 22-105
OFFSet 11-31, 11-33, 15-30
OPEE 23-18
OPER? 23-19
Operation Complete (*OPC) 12-12
Option (*OPT) 12-13
OUTPut 10-6
OVERshoot 22-107
OVLRegister? 23-20
PATTern 18-16
PERiod 22-109
PERSistence 14-14
PHASe 22-111
PLACement 18-19
POINts 8-12
POINts AUTO 8-16, 8-17
POSition 26-3
Power-on Status Clear (*PSC) 12-14
PRESet 25-13
PREShoot 22-114
PRINt 23-22
PRINters? 16-8
PROBe 11-34
PROBe ATTenuation 11-36, 11-37
PROBe EXTernal 11-42
PROBe EXTernal GAIN 11-43
PROBe EXTernal OFFSet 11-45
PROBe EXTernal UNITs 11-47
PROBe GAIN 11-49, 11-50, 11-51, 11-52
PROBe IMPedance? 21-34
PROBe SKEW 11-54
PROBe STYPe 11-55
PWD? 13-9
PWIDth 22-116
QUALifier CONDition 22-102, 22-103, 22-104, 22-113, 22-118, 22-179
QUALifier SOURce 22-119
QUALifier STATe 22-120
RANGe 11-56, 15-31, 26-4
Recall (*RCL) 12-15
RECall SETup 23-23
REFClock 26-5
REFerence 26-6
Reset (*RST) 12-16
RESults? 22-121
RISetime 22-124
RJDJ BER 22-128, 22-129
RJDJ EDGE 22-131
RJDJ INTerpolate 22-132
RJDJ PLENgth 22-133
RJDJ SOURce 22-135
RJDJ STATe 22-137
RJDJ UNITs 22-139
ROW 14-15
RUMode 21-35
SOFailure 21-37
RUN 23-24
SAVE 29-6
SAVe:IMAGe 13-10
SAVe:JITTer 13-11
SAVe:MEASurements 13-12
SAVe:SETup 13-13
SAVe:WAVeform 13-14
SCALe 11-57, 26-7
SCALe BIND 21-38
SCALe SIZE 17-6
SCALe X1 21-39
SCALe XDELta 21-40
SCALe Y1 21-41
SCALe Y2 21-42
SCOLor 14-16
SCOPETEST 24-4
SCRatch 22-140
SEGMented 13-35
SENDvalid 22-141
SERial 23-25
Service Request Enable (*SRE) 12-18
SETup 25-14
SETuptime 22-142
SINGle 23-26
SKEW 10-7
SLEWrate 22-145
SMOoth 15-32
SOURce 18-11, 18-14, 18-17, 18-20,

21-43, 22-147
SQRT 15-33
SQUare 15-34
SRATe 8-21
SRATe AUTO 8-24
STARt | STOP 21-44
STATe 18-21
STATistics 22-149
STATus? 10-8
STIMe 21-45, 21-47
STOP 23-28
STORe
WAVEform 23-31
STORe SETup 23-29, 23-30
STRing 14-19
SUBTract 15-35
TEDGe 22-150
TER? 23-32
TEST 19-7
TEXT 14-21
THResholds
ABSolute 22-152
HYSTeresis 22-154
METHod 22-156
PERCent 22-158
TOPBase 22-160, 22-162
TIEClock2 22-164
TIEData 22-167, 22-172
TIME 25-16
TITLe? 21-46
TMAX 22-169, 22-170, 22-171, 22-173
TMIN 22-175
TRIG AND 27-6, 27-7
TRIG COMM BWID 27-8
TRIG COMM ENCode 27-9
TRIG COMM PATTern 27-10
TRIG COMM POLarity 27-11
TRIG COMM SOURce 27-12
TRIG DEL ARM SLOPe 27-14
TRIG DEL ARM SOURce 27-13
TRIG DEL EDEL SLOPe 27-17
TRIG DEL TDEL TIME 27-19
TRIG DEL TRIG SLOPe 27-21
TRIG DEL TRIG SOURce 27-20
TRIG DELay EDELay COUNt 27-15
TRIG DELay EDELay SOURce 27-16
TRIG EDGE SLOPe 27-22
TRIG EDGE SOURce 27-23
TRIG GLITch POLarity 27-24
TRIG GLITch SOURce 27-25
TRIG GLITch WIDTh 27-26
TRIG HOLDoff 27-27
TRIG HTHR 27-28
TRIG HYSTeresis 27-29
TRIG LEVel 27-30
TRIG LTHR 27-31
TRIG PATT CONDition 27-35
TRIG PATT LOGic 27-36
TRIG PWID DIR 27-37
TRIG PWID POL 27-38
TRIG PWID WIDT 27-41
TRIG PWIDth SOURce 27-39
TRIG PWIDth TPOInt 27-40
TRIG RUNT POLarity 27-42
TRIG RUNT QUALified 27-43, 27-78
TRIG RUNT SOURce 27-44
TRIG RUNT TIME 27-45
TRIG SHOL CSO 27-46, 27-47, 27-48, 27-49, 27-50, 27-51, 27-52, 27-53
TRIG SHOL CSO EDGE 27-55
TRIG SHOL DSO 27-56
TRIG SHOL HTIMe 27-57
TRIG SHOL MODE 27-58
TRIG SHOL STIMe 27-59
TRIG STATe CLOCk 27-60
TRIG STATe LTYPe 27-62
TRIG STATe SLOPe 27-63
TRIG SWEep 27-64
TRIG TIM CONDition 27-65
TRIG TIM SOURce 27-66
TRIG TIM TIME 27-67
TRIG TRAN DIRection 27-68
TRIG TRAN SOUR 27-69
TRIG TRAN TIME 27-70, 27-79
TRIG TRAN TYPE 27-71
TRIG TV STV FIELd 27-73
TRIG TV STV LINE 27-72
TRIG TV STV SPOLarity 27-74, 27-76
TRIG TV UDTV ENUMber 27-77
TRIG TV UDTV PGTHan 27-80
TRIG TV UDTV POLarity 27-81
TRIG WIND SOURce 27-83
TRIG WIND TIME 27-84
TRIG WIND TPOI 27-85
TRIG WINDow CONDition 27-82
Trigger (*TRG) 12-22
TRIGger DELay MODE 27-18
TRIGger MODE 27-5, 27-32
TVOLt 22-177
ULEVel 18-15
ULIMit 18-7, 19-8
UNITinterval 22-180
UNITs 11-58
VAMPlitude 22-182
VAVerage 22-184
VBASe 22-186
VERSus 15-36
VERTical 15-37
VIEW 23-33, 26-8
VLOWer 22-188
VMAX 22-190
VMIDdle 22-192
VMIN 22-194
VPP 22-196
VRMS 22-198
VTIMe 22-200
VTOP 22-202
VUPPer 22-204
Wait-to-Continue (*WAI) 12-24
WAVeform BYTeorder 28-7
WAVeform FORMat 28-32
WAVeform SOURce 28-46
WAVeform VIEW 28-49
WINDow DEFault 17-7
WINDow DELay 26-9
WINDow POSition 26-11
WINDow RANGe 26-12
WINDow SCALe 26-13
WINDow SOURce 17-8
WINDow X1Position|LLIMit 17-9
WINDow X2Position|RLIMit 17-10
WINDow Y1Position|TLIMit 17-11
WINDow Y2Position|BLIMit 17-12
X1Position 20-5
X1Y1source 20-7
X2Position 20-6
X2Y2source 20-9
XOFFset 29-7
XRANge 29-8
Y1Position 20-12
Y2Position 20-13
YOFFset 29-9
YRANge 29-10

command
execution and order 3-4
structure 1-16
Command and Data Concepts
GPIB 2-6
Command Error 31-5
Status Bit 4-3
Command tree 6-4, 6-6
Command Types 6-4
Commands
MTEE 23-15
commands embedded in program messages 1-14
commas and spaces 1-6
comma-separated
variable file format 7-13
Common Command Header 1-8
Common Commands 12-2
Clear Status (*CLS) 12-4
Event Status Enable (*ESE) 12-5
Event Status Register (*ESR) 12-7
Identification Number (*IDN) 12-9
Learn (*LRN) 12-10
Operation Complete (*OPC) 12-12
Option (*OPT?) 12-13
Power-on Status Clear (*PSC?) 12-14
Recall (*RCL) 12-15
Reset (*RST) 12-16
Save (*SAV) 12-17
Service Request Enable (*SRE) 12-18
Status Byte (*STB?) 12-20
Test (*TST?) 12-23
Trigger (*TRG) 12-22
Wait-to-Continue (*WAI) 12-24
within a program message 12-3
COMMonmode 15-8
commonmode voltage of operands 15-8
Communicating Over the GPIB Interface 2-7
Communicating Over the LAN Interface 2-8
COMPlete 8-5
COMPlete query 28-8
COMPlete STATe 8-7
compound command header 1-7
compound queries 3-4
Computer Code and Capability 2-5
concurrent commands 6-12
CONDition
in TRIG PATTern 27-35
in TRIG TIM 27-65
in TRIG WINDow 27-82
CONNect 14-8, 28-42, 28-47
conventions of programming 6-2
converting waveform data
from data value to Y-axis units 28-5
sample program 7-12
COPY 13-4
copying files 13-4
COUNt 8-4
in MTESt AVERage command 21-17
in TRIG DEL EDEL 27-15
COUNt query 28-9
COUPling query 28-10
coupling, input 11-16
CREate
in MTESt AMASk command 21-7
CROSsing
in MEASure CGRade command 22-13
CTCDutycycle 22-30
CTCJitter 22-32
CTCNwidth 22-34
CTCPwidth 22-36
CURSor? 20-3

D

data
acquisition 28-4
conversion 28-5
data in a learnstring 1-4
data in a program 1-6
Data Mode
GPIB 2-6
Data Structures
and Status Reporting 4-5
data transmission mode
and FORMat 28-32
DATA? 14-9, 28-11
Analog Channels C Program 28-15
DATarate 22-11, 22-38, 22-206
DATE 25-3
DCDistortion
in MEASure CGRade command 22-14
DDE bit 12-6, 12-8
DEBug 25-4
decimal 32 (ASCII space) 1-5
Decision Chart for Status Reporting 4-20
DEFault
in HISTogram WINDow command 17-7
Default
GPIB Address 2-7
Startup Conditions 2-4
default setup 25-13
Default Startup Conditions 2-4
defining functions 15-2
def-length block response data 1-21
DELay 18-3
delay
and WINDow DELay 26-9
DELete 13-5, 21-23
deleting files 13-5
DELTatime 22-41
derivative of functions 15-9
Device Address
GPIB 2-7
LAN 2-8
device address 1-3, 1-4
Device Clear (DCL) 2-12
Device Clear Code and Capability 2-5
Device Dependent Error (DDE), Status Bit 4-4
Device- or Oscilloscope-Specific Error 31-7
Device Trigger Code and Capability 2-5
device-dependent data 1-21
DFREQuency
in MEASure FFT command 22-49
DIFF 15-9
DIGitize 23-14
setting up for execution 8-2
Digitize
Aborting 2-12
DIRectory? 13-6
Disabling Serial Poll 2-12
discrete derivative of functions 15-9
Disk Commands 13-2
CDIRectory 13-3
COPY 13-4
DELete 13-5
DIRectory? 13-6
LOAD 13-7
MDIRectory 13-8
PWD? 13-9

SAVe:IMAGe 13-10
SAVe:JITTer 13-11
SAVe:MEASurements 13-12
SAVe:SETup 13-13
SAVe:WAVeform 13-14
SEGMented 13-35
DISPlay 11-4, 11-5, 11-6, 11-7, 11-8, 11-9, 11-11, 11-12, 11-14, 15-10, 29-3, 29-4
DISPlay Commands
CGRade 14-3, 14-20
CGRADE LEVels? 14-5
CGRade LEVels? 14-5
Display Commands 14-2
COLumn 14-7
CONNect 14-8, 28-42, 28-47
DATA? 14-9
GRATicule 14-10
GRATicule INTensity 14-10
LINE 14-13
PERSistence 14-14
ROW 14-15
SCOLor 14-16
STRing 14-19
TEXT 14-21
display persistence 14-14
DIVide 15-11
dividing functions 15-11
DMAGnitude
in MEASure FFT command 22-51
DPRinter 16-4
Driver Electronics Code and Capability 2-5
DSP (display) 25-6
duplicate mnemonics 1-9
DUTYcycle 22-45

E

EADapter 11-38
ECoupling 11-40
EHEight
in MEASure CGRade command 22-15
Ellipsis
... 1-5
embedded
commands 1-14
strings 1-3, 1-4, 1-13
ENABle 21-24
Enable Register 12-3
ENCode
in TRIG COMM 27-9
End Of String (EOS) 1-13
End Of Text (EOT) 1-13
End-Or-Identify (EOI) 1-13
Enhanced Bandwidth 8-8
ENUMber
in TRIG TV UDTV 27-77
EOI and IEEE 488.2 6-12
error
in measurements 22-7
messages 31-2
numbers 31-4
query interrupt 1-10, 1-19
Error Messages table 31-9
error queue 31-3
and status reporting 4-18
overflow 31-3
ERRor? 25-7
errors
exceptions to protocol 3-4
ESB (Event Status Bit) 4-4, 12-19, 12-21
ESB (Event Summary Bit) 12-5
*ESE (Event Status Enable) 12-5
ESR (Standard Event Status Register) 4-11
ETIMe 8-10
event monitoring 4-2
Event Registers Default 2-4
Event Status Bit (ESB) 4-4
Event Status Enable (*ESE)
Status Reporting 4-12
Event Summary Bit (ESB) 12-5
EWIDth
in MEASure CGRade command 22-16
EWINdow 22-17
Example Program 1-16
in initialization 1-16
example programs
C and BASIC 7-2
exceptions to protocol 3-4
EXE bit 12-6, 12-8
executing DIGITIZE 8-2
execution
errors, and command errors 31-5
of commands and order 3-4
Execution Error 31-6
Execution Error (EXE), Status Bit 4-3
exponential notation 1-12
exponents 1-12

F

FACTors 16-6
FAIL 18-4, 19-3
FAILures?
in MTESt COUNt command 21-18, 21-19, 21-21
fall time measurement setup 22-7
FALLtime 22-47
FFT Commands 22-7
FFTMagnitude 15-17
FFTPhase 15-18
FIELd
in TRIG TV STV 27-73
FOLDing 21-25
FOLDing:BITS 21-26
FORMat 28-32
and DATA 28-14
formatting query responses 25-2
fractional values 1-12
FREQuency 22-60
in FUNCtion FFT command 15-12
in MEASure FFT command 22-53
frequency measurement setup 22-7
full-scale vertical axis 11-56
FUNCtion 15-4
function
and vertical scaling 15-31
time scale 15-3
Function Commands 15-2
ADD 15-6
AVERage 15-7
COMMonmode 15-8
DIFF 15-9
DISPlay 15-10
DIVide 15-11
FFT FREQuency 15-12
FFT REFerence 15-13
FFT RESolution 15-14
FFT WINDow 15-15
FFTMagnitude 15-17
FFTPhase 15-18
FUNCtion? 15-4
HORizontal 15-20
HORizontal POSition 15-21
HORizontal RANGe 15-22
INTegrate 15-23
INVert 15-24
MAGNify 15-26

MAXimum 15-27
MINimum 15-28
MULTiply 15-29
OFFSet 15-30
RANGe 15-31
SMOoth 15-32
SQRT 15-33
SQUare 15-34
SUBTract 15-35
VERSus 15-36
VERTical 15-37
functional elements of protocol 3-3
functions
and VIEW 28-49
combining in instructions 1-8
FWAVeforms?
in MTESt COUNt command 21-20

G

GAIN 11-43
gain and offset of a probe 10-4
gain factor for user-defined probe 11-43
GPIB
Interface Connector 2-3
GRATicule 14-10
HARDcopy AREA 16-3
Group Execute Trigger (GET) 2-12

H

Halting bus activity 2-12
HAMPlitude 21-27
Hardcopy Commands 16-2
AREA 16-3
DPRinter 16-4
FACTors 16-6
IMAGe 16-7
PRINters? 16-8
hardcopy of the screen 16-2
hardcopy output and message termination 3-4
HEADer 25-8, 25-10
header
stripped 7-11
within instruction 1-4
headers 1-4
types 1-7
Histogram Commands 17-2
AXIS 17-4
MODE 17-5
SCALe SIZE 17-6
WINDow DEFault 17-7
WINDow SOURce 17-8
WINDow X1Position|LLIMit 17-9
WINDow X2Position|RLIMit 17-10
WINDow Y1Position|TLIMit 17-11
WINDow Y2Position|BLIMit 17-12
HITS
in MEASure HISTogram command 22-63
HOLDoff
in TRIGger 27-27
HOLDtime 22-78
HORizontal 15-20
horizontal
functions, controlling 26-2
offset, and XOFFset 29-7
range, and XRANge 29-8
scaling and functions 15-2
HORizontal POSition 15-21
HORizontal RANGe 15-22
Host language 1-4
HP BASIC 5.0 1-2
HTHReshold 27-28
hue 14-17
HYSTeresis
in TRIGger 27-29

I

*IDN? (Identification Number) 12-9
IEEE 488.1 3-2
and IEEE 488.2 relationship 3-2
IEEE 488.2 3-2
compliance 3-2
conformity 1-2
Standard 1-2
Standard Status Data Structure Model 4-2
IMAGe 16-7
image specifier, -K 25-15
image specifiers
and PREamble 28-39
IMPedance 21-28
impedance, input 11-16
IMPedance?
in MTESt PROBe command 21-34
individual commands language 1-2
Infinity Representation 6-12
initialization 1-15
event status 4-2
IO routine 7-5
sample program 7-4
initializing oscilloscope
sample program 7-6
INPut 11-16, 11-17, 11-19, 11-21, 11-23, 11-24, 11-25, 11-27, 11-29, 11-32
Input Buffer
Clearing 2-12
input buffer 3-3
default condition 3-4
input coupling
and COUPling? 28-10
instruction headers 1-4
Instrument Address
GPIB 2-7
instrument status 1-22
integer definition 1-12
INTegrate 15-23
intensity 14-10
Interface
Capabilities 2-5
Clear (IFC) 2-12
GPIB Select Code 2-7
interface
functions 2-2
interface, initializing 1-15
INTerpolate 8-9
interpreting commands, parser 3-3
interrupted query 1-10, 1-19
Introduction to Programming 1-2
INVert 15-24, 21-30
inverting functions 15-24
ISCan
DELay 18-3
EDGE 18-9
FAIL 18-4
HYSTeresis 18-10, 18-12
LLEVel 18-13
MEASurement 18-6
MODE 18-8
SOURce 18-11
ULIMit 18-7

J

JITTer

in MEASure CGRade command 22-19
JITTer HISTogram 22-81
JITTer MEASurement 22-82
JITTer SPECtrum 22-83
JITTer SPECtrum HORizontal 22-84
JITTer SPECtrum HORizontal POSition 22-85
JITTer SPECtrum HORizontal RANGe 22-87
JITTer SPECtrum VERTical 22-88
JITTer SPECtrum VERTical OFFSet 22-89
JITTer SPECtrum VERTical RANGe 22-90
JITTer SPECtrum WINDow 22-91
JITTer STATistics 22-92
JITTer TRENd 22-93
JITTer TRENd SMOoth 22-94
JITTer TRENd SMOoth POINts 22-95
JITTer TRENd VERTical 22-96
JITTer TRENd VERTical OFFSet 22-27, 22-97, 30-11, 30-29
JITTer TRENd VERTical RANGe 22-98

K

-K 25-15

L

LAMPlitude 21-31
language for program examples 1-2
Learn (*LRN) 12-10
learnstring block data 1-4
LEVel
in TRIGger 27-30
LEVels?
in DISPlay CGRade command 14-5
LINE 14-13
in TRIG TV STV 27-72
linefeed 1-13
List of Error Messages 31-9
Listener Code and Capability 2-5
Listeners, Unaddressing All 2-12
LLEVel 18-13
LLIMit 18-5, 19-4
LOAD 13-7, 21-32, 29-5
loading and saving 13-2
LOGic
and STATe 27-61
in TRIG PATT 27-36
in TRIG STATe 27-61
LONGform 25-11
long-form headers 1-11
lowercase 1-11
headers 1-11
*LRN (Learn) 12-10
*LRN?
and SYSTem SETup? 25-15
LSBFirst, and BYTeorder 28-7
LTESt
FAIL 19-3
LLIMit 19-4
MEASurement 19-5
RESults? 19-6
TEST 19-7
ULIMit 19-8
LTHReshold 27-31
LTYPe
and STATe 27-62
in TRIG STATe 27-62
luminosity 14-17

M

M1S
in MEASure HISTogram command 22-65
M2S
in MEASure HISTogram command 22-67
M3S
in MEASure HISTogram command 22-69
MAGNify 15-26
MAGNitude
in MEASure FFT command 22-55
MAIN, and VIEW 28-49
making measurements 22-7
Marker Commands 20-2
CURSor? 20-3
MODE 20-4
X1Position 20-5
X1Y1source 20-7
X2Position 20-6
X2Y2source 20-9
XDELta? 20-11
Y1Position 20-12
Y2Position 20-13
YDELta? 20-14
Mask Test Commands 21-2
ALIGn 21-4
AlignFIT 21-5
AMASk CREate 21-7
AMASk SAVE|STORe 21-9
AMASk SOURce 21-8
AMASk UNITs 21-10
AMASk XDELta 21-11
AMASk YDELta 21-13
AUTO 21-15
AVERage 21-16
AVERage COUNt 21-17
COUNt FAILures? 21-18, 21-19, 21-21
COUNt FWAVeforms? 21-20
COUNt WAVeforms? 21-22
DELete 21-23
ENABle 21-24
FOLDing 21-25
FOLDing:BITS 21-26
HAMPlitude 21-27
IMPedance 21-28
INVert 21-30
LAMPlitude 21-31
LOAD 21-32
NREGions? 21-33
PROBe IMPedance? 21-34
RUMode 21-35
RUMode SOFailure 21-37
SCALe
BIND 21-38
Y1 21-41
SCALe X1 21-39
SCALe XDELta 21-40
SCALe Y1 21-41
SCALe Y2 21-42
SOURce 21-43
STARt | STOP 21-44
STIMe 21-45, 21-47
TITLe? 21-46
mask, Service Request Enable Register 12-18
Master Summary Status (MSS)
and *STB 12-20
Status Bit 4-4
MAV (Message Available) 4-4
bit 12-19, 12-21
MAX
in MEASure HISTogram command

22-71
MAXimum 15-27
MDIRectory 13-8
MEAN
in MEASure HISTogram command 22-72
MEASure
RESults and statistics 22-149
Measure Commands 22-2
ABSolute 15-5
AREA 22-9, 22-29, 22-99
BWIDth 22-10
CGRade CROSsing 22-13
CGRade DCDistortion 22-14
CGRade EHEight 22-15
CGRade EWIDth 22-16
CGRade JITTer 22-19
CGRade QFACtor 22-20
CLEar 22-21
CLOCk 22-22
CLOCk METHod 22-23
CLOCk VERTical 22-26, 30-10, 30-28
CLOCk VERTical RANGe 22-28, 30-12, 30-30
CTCDutycycle 22-30
CTCJitter 22-32
CTCNwidth 22-34
CTCPwidth 22-36
DATarate 22-11, 22-38, 22-206
DELTatime 22-41
DUTYcycle 22-45
FALLtime 22-47
FFT DFRequency 22-49
FFT DMAGnitude 22-51
FFT FREQuency 22-53
FFT MAGNitude 22-55
FFT PEAK1 22-57
FFT PEAK2 22-58
FFT THReshold 22-59
FREQuency 22-60
HISTogram HITS 22-63
HISTogram M1S 22-65
HISTogram M2S 22-67
HISTogram M3S 22-69
HISTogram MAX 22-71
HISTogram MEAN 22-72
HISTogram MEDian 22-73
HISTogram MIN 22-74
HISTogram PEAK 22-75
HISTogram PP 22-76
HISTogram STDDev 22-77
HOLDtime 22-78
JITTer HISTogram 22-81
JITTer MEASurement 22-82
JITTer SPECtrum 22-83
JITTer SPECtrum HORizontal 22-84
JITTer SPECtrum HORizontal POSition 22-85
JITTer SPECtrum HORizontal RANGe 22-87
JITTer SPECtrum VERTical 22-88
JITTer SPECtrum VERTical OFFSet 22-89
JITTer SPECtrum VERTical RANGe 22-90
JITTer SPECtrum WINDow 22-91
JITTer STATistics 22-92
JITTer TRENd 22-93
JITTer TRENd SMOoth 22-94
JITTer TRENd SMOoth POINts 22-95
JITTer TRENd VERTical 22-96
JITTer TRENd VERTical OFFSet 22-27, 22-97, 30-11, 30-29
JITTer TRENd VERTical RANGe 22-98
NCJitter 22-100
NWIDth 22-105
OVERshoot 22-107
PERiod 22-109
PHASe 22-111
PREShoot 22-114
PWIDth 22-116
QUALifier CONDition 22-102, 22-103, 22-104, 22-113, 22-118, 22-179
QUALifier SOURce 22-119
QUALifier STATe 22-120
RESults? 22-121
RISetime 22-124
RJDJ BER 22-128, 22-129
RJDJ EDGE 22-131
RJDJ INTerpolate 22-132
RJDJ PLENgth 22-133
RJDJ SOURce 22-135
RJDJ STATe 22-137
RJDJ UNITs 22-139
SCRatch 22-140
SENDvalid 22-141
SETuptime 22-142
SLEWrate 22-145
SOURce 22-147
STATistics 22-149
TEDGe 22-150
THResholds
ABSolute 22-152
HYSTeresis 22-154
METHod 22-156
PERCent 22-158
TOPBase 22-160, 22-162
TIEClock2 22-164
TIEData 22-167, 22-172
TMAX 22-169, 22-170, 22-171, 22-173
TMIN 22-175
TVOLt 22-177
UNITinterval 22-180
VAMPlitude 22-182
VAVerage 22-184
VBASe 22-186
VLOWer 22-188
VMAX 22-190
VMIDdle 22-192
VMIN 22-194
VPP 22-196
VRMS 22-198
VTIMe 22-200
VTOP 22-202
VUPPer 22-204
MEASurement 18-6, 19-5
LLIMit 18-5
measurement
error 22-7
setup 22-7
source 22-147
MEDian
in MEASure HISTogram command 22-73
memories, and VIEW 28-49
message
queue 4-19
termination with hardcopy 3-4
Message (MSG), Status Bit 4-4
Message Available (MAV)
and *OPC 12-12

Status Bit 4-4
Message Communications and System Functions 3-2
Message Event Register 4-10
message exchange protocols
of IEEE 488.2 3-3
MIN
in MEASure HISTogram command 22-74
MINimum 15-28
Mnemonic Truncation 6-3
MODE 8-10, 17-5, 18-8, 18-18, 20-4
in TRIGger MODE 27-5, 27-32
MODel? 23-17
monitoring events 4-2
MSBFirst, and BYTeorder 28-7
MSG
bit in the status register 4-10
MSG bit 12-19, 12-21
MSS bit and *STB 12-20
MTEE 23-15
MTER? 23-16
multiple
program commands 1-14
queries 1-22
subsystems 1-14
Multiple numeric variables 1-22
MULTiply 15-29

N

NCJitter 22-100
NL (New Line) 1-13
NONMonotonic
EDGE 18-9
HYSTeresis 18-10
SOURce 18-11
NREGions? 21-33
numeric
program data 1-12
variable example 1-20
variables 1-20
NWIDth 22-105

O

OFFSet 11-31, 11-33, 11-45, 15-30
offset and gain of a probe 10-4
*OPC (Operation Complete) 12-12
OPC bit 12-6, 12-8
OPEE 23-18
OPER bit 12-19, 12-21
OPER query 23-19
operands and time scale 15-3
operating the disk 13-2
Operation Complete (*OPC) 12-12
Status Bit 4-4
operation status 4-2
*OPT (Option) 12-13
Options, Program Headers 1-11
order of commands and execution 3-4
oscilloscope
trigger modes and commands 27-5
Oscilloscope Default GPIB Address 2-7
OUTPut 10-6
output buffer 1-10, 1-19
Output Command 1-4
Output Queue
Clearing 2-12
output queue 1-10, 4-18
default condition 3-4
definition 3-3
OUTPUT statement 1-3
overlapped and sequential commands 6-12
OVERshoot 22-107
OVLRegister query 23-20

P

Parallel Poll Code and Capability 2-5
parametric measurements 22-2
Parser
Resetting 2-12
parser 1-15, 3-3
default condition 3-4
definition 3-3
passing values across the bus 1-10
PATTern 18-16
in TRIG COMM 27-10
PDETect 8-10
PEAK
in MEASure HISTogram command 22-75
PEAK1
in MEASure FFT command 22-57
PEAK2
in MEASure FFT command 22-58
peak-to-peak voltage, and VPP 22-196
Pending Commands, Clearing 2-12
PERiod 22-109
period measurement setup 22-7
PERsistence 14-14
PGTHan
in TRIG TV UDTV 27-80
PHASe 22-111
PLACement 18-19
POINts 8-12
POINts AUTO 8-16, 8-17
POINts query 28-35
POLarity
and GLITch 27-24
and RUNT 27-42
in TRIG COMM 27-11
in TRIG TV UDTV 27-81
in TRIGger GLITch 27-24
in TRIGger RUNT 27-42
PON bit 12-8
POSition 26-3
position
and WINDow POSition 26-11
pound sign (#) and block data 1-21
Power On (PON) status bit 4-3, 12-6
Power-up Condition 2-4
PP
in MEASure HISTogram command 22-76
PREamble 28-36
and DATA 28-14
PRESet 25-13
PREShoot 22-114
PRINt 23-22
PRINters? 16-8
printing
specific screen data 16-3
the screen 16-2
PROBe 11-34
PROBe ATTenuation 11-36, 11-37
probe attenuation factor 10-4
Probe Calibration 10-4
PROBe EXTernal 11-42
PROBe EXTernal GAIN 11-43
PROBe EXTernal OFFSet 11-45
PROBe EXTernal UNITs 11-47
PROBe GAIN 11-49, 11-50, 11-51, 11-52
PROBe SKEW 11-54
PROBe STYPe 11-55
program data 1-6

Program example 1-16
Program Header Options 1-11
program message terminator 1-13
program overview
  initialization example 1-16
programming basics 1-2
Programming Conventions 6-2
programming examples language 1-2
Programming Getting Started 1-14
protocol
  exceptions and operation 3-4
*PSC (Power-on Status Clear) 12-14
pulse width measurement setup 22-7
PWD? 13-9
PWIDth 22-116
PWIDth DIRection 27-37
PWIDth POLarity 27-38
PWIDth SOURce 27-39
PWIDth TPOInt 27-40
PWIDth WIDTh 27-41

Q

QFACtor
  in MEASure CGRade command 22-20
Query
  *SRE? 12-18
QUALified
  and RUNT 27-43, 27-78
  in TRIGger RUNT 27-43, 27-78
QUALifier
  CONDition 22-102, 22-104, 22-118, 22-179
  SOURce 22-119
  STATe 22-120
quantization levels 7-12
Query 1-4, 1-10
  *ESE? (Event Status Enable) 12-5
  *ESR? (Event Status Register) 12-7
  *STB? (Status Byte) 12-20
  AER? 23-4, 23-5, 23-21
  AREA? 16-3
  ATER? 23-6, 23-9
  AVERage? 8-3
  BANDpass? 28-6, 30-4, 30-5, 30-6, 30-7, 30-8, 30-9, 30-13, 30-14, 30-15, 30-16, 30-17, 30-18, 30-19, 30-20, 30-21, 30-22, 30-23, 30-24, 30-25, 30-26, 30-27, 30-31, 30-32, 30-33, 30-34, 30-35, 30-36, 30-37, 30-38, 30-39, 30-40, 30-41, 30-42
  BANdwidth? 8-8
  BYTeorder? 28-7, 30-4, 30-5, 30-6, 30-7, 30-8, 30-9, 30-13, 30-14, 30-15, 30-16, 30-17, 30-18, 30-19, 30-20, 30-21, 30-22, 30-23, 30-24, 30-25, 30-26, 30-27, 30-31, 30-32, 30-33, 30-34, 30-35, 30-36, 30-37, 30-38, 30-39, 30-40, 30-41, 30-42
  CHANnel PROBe ID? 11-53
  COLumn? 14-7
  COMPlete STATe? 8-7
  COMPlete? 8-6, 28-8
  CONNect? 14-8, 28-42, 28-47
  COUNt? 8-4, 21-17, 28-9
  COUPling? 28-10
  CURSor? 20-3
  DATA? 14-9, 28-11
  DATE? 25-3
  DEBug? 25-5
  DEFine? 22-44
  DELay? 18-3
  DELTatime? 22-42
  DIRectory? 13-6
  DISPlay? 11-4, 11-5, 11-6, 11-7, 11-8, 11-9, 11-11, 11-12, 11-14, 11-42, 15-10, 29-4
  DPRinter? 16-5
  DSP? 25-6
  DUTYcycle? 22-46
  EADapter? 11-39
  ECoupling? 11-41
  EDGE 18-9
  ERRor? 25-7
  FACTors? 16-6
  FAIL? 18-4, 19-3
  FALLtime? 22-48
  FFT RESolution? 15-14
  FORMat? 28-34
  FREQuency? 22-62
  FUNCtion? 15-4
  GRATicule? 14-11, 14-12
  HEADer 25-8, 25-10
  HORizontal POSition? 15-21
  HORizontal RANGe? 15-22
  HORizontal? 15-20
  HYSTeresis 18-10, 18-12
  Identification Number (*IDN?) 12-9
  IMAGe? 16-7
  INPut? 11-16, 11-17, 11-19, 11-21, 11-25, 11-27, 11-29, 11-32
  INTerpolate? 8-9
  Learn (*LRN?) 12-10
  LLEVel? 18-13
  LLIMit? 18-5, 19-4
  LONGform? 25-11
  MEASure FALLtime? 22-48
  MEASure FFT DFRequency? 22-49
  MEASure FFT DMAGnitude? 22-51
  MEASure FFT FREQuency? 22-53
  MEASure FFT MAGNitude? 22-55
  MEASure FFT PEAK1? 22-57
  MEASure FFT PEAK2? 22-58
  MEASure FFT THReshold? 22-59
  MEASurement 18-6, 19-5
  MODE 18-8, 18-18
  MODE? 8-11, 20-4
  MODel? 23-17
  MTEE? 23-15
  MTER? 23-16
  NWIDth? 22-106
  OFFSet? 11-31, 11-33, 11-46, 15-30
  Option (*OPT?) 12-13
  OUTPut? 10-6
  OVERshoot? 22-108
  PATTern 18-16
  PERiod? 22-110
  PERSistence? 14-14
  PHASe? 22-112
  PLACement 18-19
  POINts AUTO? 8-16, 8-17, 8-20
  POINts? 8-15, 28-35
  POSition? 26-3
  Power-on Status Clear (*PSC?) 12-14
  PREamble? 28-36
  PREShoot? 22-115
  PRINters? 16-8
  PROBe ATTenuation? 11-36, 11-37
  PROBe GAIN? 11-44, 11-49, 11-50, 11-51, 11-52
  PROBe MODE? 11-55
  PROBe SKEW? 11-54
  PROBe? 11-35
  PWD? 13-9
  PWIDth? 22-117

QUALifier CONDition 22-102, 22-103, 22-104, 22-113, 22-118, 22-179
QUALifier SOURce? 22-119
QUALifier STATe? 22-120
RANGe? 11-56, 15-31, 26-4
REFClock 26-5
REFerence? 26-6
RESults 19-6
RESults? 22-121
RISetime? 22-125
RJDJ ALL? 22-126
RJDJ BER? 22-128, 22-130
RJDJ EDGE? 22-131
RJDJ INTerpolate? 22-132
RJDJ PLENgth? 22-133
RJDJ SOURce? 22-135
RJDJ STATe? 22-137
RJDJ TJRJDJ? 22-138
RJDJ UNITs? 22-139
ROW? 14-15
SCALe? 11-57, 26-7
SCOLor? 14-18
SCOPETEST? 24-4
SENDvalid? 22-141
SETup? 25-14
SETuptime? 22-80, 22-144
SKEW? 10-7
SLEWrate? 22-145
SOURce 18-11, 18-14, 18-17, 18-20
SOURce? 22-147, 28-46
SRATe AUTO? 8-24
SRATe? 8-22
STATe 18-21
STATistics? 22-149
Status Byte (*STB) 12-20
STATus? 10-8
TEDGe? 22-151
TER? 23-32
Test (*TST?) 12-23
TEST? 19-7
TMAX? 22-169, 22-170, 22-171, 22-173
TMIN? 22-175
TRIG AND:ENABle? 27-6
TRIG AND:SOURce? 27-7
TRIG COMM BWID? 27-8
TRIG COMM ENCode? 27-9
TRIG COMM PATTern? 27-10
TRIG COMM POLarity? 27-11
TRIG COMM SOURce? 27-12
TRIG DEL ARM SLOPe? 27-14
TRIG DEL ARM SOURce 27-13
TRIG DEL EDEL COUN? 27-15
TRIG DEL EDEL SLOPe? 27-17
TRIG DEL EDEL SOURce? 27-16
TRIG DEL TDEL TIME? 27-19
TRIG DEL TRIG SLOPe? 27-21
TRIG DEL TRIG SOURce? 27-20
TRIG EDGE SLOPe? 27-22
TRIG EDGE SOURce? 27-23
TRIG GLITch POLarity? 27-24
TRIG GLITch SOURce? 27-25
TRIG HOLDoff? 27-27
TRIG HTHR? 27-28
TRIG HYSTeresis? 27-29
TRIG LEVel? 27-30
TRIG LTHR? 27-31
TRIG PATT COND? 27-35
TRIG PATT LOGic? 27-36
TRIG PWID DIR? 27-37
TRIG PWID POL? 27-38
TRIG PWID WIDT? 27-41
TRIG PWIDth SOURce? 27-39
TRIG PWIDth TPOInt? 27-40
TRIG RUNT POLarity? 27-42
TRIG RUNT QUALified? 27-43, 27-78
TRIG RUNT SOURce? 27-44
TRIG SHOL CSO EDGE? 27-55
TRIG SHOL CSO? 27-46, 27-47, 27-48, 27-49, 27-51, 27-52, 27-53, 27-54
TRIG SHOL DSO? 27-56
TRIG SHOL HTIMe? 27-57
TRIG SHOL MODE? 27-58
TRIG SHOL STIMe? 27-59
TRIG STATe CLOCk? 27-60
TRIG STATe LOGic? 27-61
TRIG STATe LTYPe? 27-62
TRIG STATe SLOPe? 27-63
TRIG SWEep? 27-64
TRIG TIM COND? 27-65
TRIG TIM SOUR? 27-66
TRIG TIM TIME? 27-67
TRIG TRAN DIR? 27-68
TRIG TRAN SOUR? 27-69
TRIG TRAN TIME? 27-70, 27-79
TRIG TRAN TYPE? 27-71
TRIG TV SOURce? 27-75
TRIG TV STV FIELd? 27-73
TRIG TV STV LINE? 27-72
TRIG TV STV SPOLarity? 27-74, 27-76
TRIG TV UDTV ENUMber? 27-77
TRIG TV UDTV PGTHan? 27-80
TRIG TV UDTV POLarity? 27-81
TRIG WIND COND? 27-82
TRIG WIND SOUR? 27-83
TRIG WIND TIME? 27-84
TRIG WIND TPOI? 27-85
TRIGger DELay MODE? 27-18
TRIGger GLITch WIDTh? 27-26
TRIGger MODE? 27-34
TRIGger RUNT TIME? 27-45
TVOLt? 22-177
TYPE? 28-48
ULEVel? 18-15
ULIMit? 18-7, 19-8
UNITs? 11-48, 11-58
VAMPlitude? 22-182
VAVerage? 22-185
VBASe? 22-187
VIEW? 26-8, 28-50
VLOWer? 22-188
VMAX? 22-191
VMIDdle? 22-192
VMIN? 22-195
VPP? 22-197
VRMS? 22-199
VTIMe? 22-200
VTOP? 22-203
VUPPer? 22-205
WAVeform SEGMented COUNt? 28-43
WAVeform SEGMented TTAG? 28-44, 28-45
WINDow DELay? 26-10
WINDow POSition? 26-11
WINDow RANGe? 26-12
WINDow SCALe? 26-13
X1Position? 20-5
X1Y1source? 20-7
X2Position? 20-6

X2Y2source? 20-9
XDELta? 20-11
XDISplay? 28-51
XINCrement? 28-52
XOFFset? 29-7
XORigin? 28-53
XRANge? 28-54, 29-8
XREFerence? 28-55
XUNits? 28-56
Y1Position? 20-12
YDELta? 20-14
YDISplay? 28-57
YINCrement? 28-58
YOFFset? 29-9
YORigin? 28-59
YRANge? 28-60, 29-10
YREFerence? 28-61
YUNits? 28-62
query
headers 1-10
interrupt 1-10
response 1-19
responses, formatting 25-2
Query Error 31-8
QYE Status Bit 4-4
query interrupt 1-19
question mark 1-10
queue, output 1-10
quoted strings 14-13
quotes, with embedded strings 1-13
QYE bit 12-6, 12-8

R

RANGe 11-56, 15-31, 26-4
range
and WINDow RANGe 26-12
*RCL (Recall) 12-15
real number definition 1-12
real time mode 8-10
and interpolation 8-9
RECall 23-23
Receiving Common Commands 12-3
Receiving Information from the Instrument 1-19
REFClock 26-5
REFerence 26-6
in FUNCtion FFT command 15-13
register
save/recall 12-15, 12-17
Standard Event Status Enable 4-12
reliability of measured data 4-2
Remote Local Code and Capability 2-5
remote programming basics 1-2
REPetitive 8-10
Repetitive Strain Injury 5-2
Description 5-3
Using the Mouse 5-5, 5-6
representation of infinity 6-12
Request Control (RQC)
Status Bit 4-4
Request Service (RQS)
Default 2-4
status bit 4-4
Reset (*RST) 12-16
Resetting the Parser 2-12
RESolution
in FUNCtion FFT command 15-14
response
data 1-21
generation 6-12
responses, buffered 6-12
result state code, and SENDvalid 22-141
RESults? 19-6, 22-121
Returning control to system computer 2-12
rise time measurement setup 22-7
RISetime 22-124
RJDJ
ALL? 22-126
BER 22-128, 22-129
EDGE 22-131
INTerpolate 22-132
PLENgth 22-133
SOURce 22-135
STATe 22-137
TJRJDJ 22-138
UNITs 22-139
RMS voltage, and VRMS 22-198
Root level command
MTEE 23-15
Root level commands 23-2
AER? 23-4, 23-5, 23-21
ATER? 23-6, 23-9
AUToscale 23-7, 23-8, 23-9, 23-10
BEEP 23-11
BLANk 23-12
CDISplay 23-13
DIGitize 23-14
MODel? 23-17
OPEE 23-18
OPER? 23-19
OVLRegister? 23-20
PRINt 23-22
RECall 23-23
RUN 23-24
SERial 23-25
SINGle 23-26
STOP 23-28
STORe 23-29, 23-30
STORe WAVEform 23-31
TER? 23-32
VIEW 23-33
ROW 14-15
RQC (Request Control) 4-4
bit 12-6, 12-8
RQS (Request Service) 4-4
and *STB 12-20
Default 2-4
RQS/MSS bit 12-21
RSI
Description 5-3
Introduction 5-2
Using the Mouse 5-5, 5-6
*RST (Reset) 12-16
RTIMe 8-10
rule of truncation 6-3
rules of traversal 6-5
RUMode 21-35
RUN 23-24
and GET relationship 2-12
RUNT
HYSTeresis 18-12
PATTern 18-16
SOURce 18-14
ULEVel 18-15

S

sample programs 7-2
segments 7-3
sample rate 8-21
sampling mode 8-10
saturation 14-17
*SAV (Save) 12-17
SAVE 29-6
save/recall register 12-15, 12-17

SAVe:IMAGe 13-10
SAVe:JITTer 13-11
SAVe:MEASurements 13-12
SAVe:SETup 13-13
SAVe:WAVeform 13-14
SAVE|STORe
  in MTESt AMASk command 21-9
saving and loading 13-2
SCALe 11-57, 26-7
  Y1 21-41
SCOLor 14-16
SCOPETEST
  in self-test commands 24-4
SCRatch 22-140
SCReen
  HARDcopy AREA 16-3
SEGMented 13-35
  COUNt? 28-43
  TTAG? 28-44, 28-45
segments of sample programs 7-3
Selected Device Clear (SDC) 2-12
Selecting Multiple Subsystems 1-14
self test 12-23
Self-Test Commands 24-2
  CANCel 24-3
  SCOPETEST 24-4
semicolon usage 1-8
sending compound queries 3-4
SENDvalid 22-141
separator 1-5
Sequential and Overlapped Commands 6-12
SERial
  SOURce 18-17
SERial (SERial number) 23-25
Serial Poll
  Disabling 2-12
serial poll
  (SPOLL) in example 4-9
  of the Status Byte Register 4-9
serial prefix, reading 12-9
Service Request
  Code and Capability 2-5
Service Request Enable
  (*SRE) 12-18
  Register (SRE) 4-10
  Register Bits 12-19
  Register Default 2-4
setting
  bits in the Service Request Enable Register 4-10
  horizontal tracking 15-20
  Standard Event Status Enable Register bits 4-12
  time and date 25-16
  TRG bit 4-10
  voltage and time markers 20-2
setting up
  for programming 1-14
  the instrument 1-15
SETup 25-14
setup recall 12-15
SETuptime 22-142
SHOLd CSOurce EDGE 27-55
SHOLd DSOurce 27-56
SHOLd HoldTIMe 27-57
SHOLd MODE 27-58
SHOLd SetupTIMe 27-59
Short form 1-11
short-form headers 1-11
short-form mnemonics 6-3
simple command header 1-7
SINGle 23-26
SIZE
  in HISTogram SCALe command 17-6
SKEW, in CALibrate command 10-7
SLEWrate 22-145
SLOPe
  and STATe 27-63
  in TRIG DEL EDEL 27-17
  in TRIG DEL TRIGger 27-21
  in TRIG STATe 27-63
  in TRIGger EDGE 27-22
SMOoth 15-32
SOFailure
  in MTESt RUMode command 21-37
software version, reading 12-9
SOURce 18-14, 18-17, 18-20, 21-43, 22-147, 28-46
  and GLITch 27-25
  and measurements 22-8
  and RUNT 27-44
  in HISTogram WINDow command 17-8
  in MTEST AMASk command 21-8
  in TRIG COMM 27-12
  in TRIG DEL EDEL 27-16
  in TRIG DEL TRIGger 27-20
  in TRIGger EDGE 27-23
  in TRIGger GLITch 27-25
  in TRIGger RUNT 27-44
spaces and commas 1-6
spelling of headers 1-11
SPOLarity
  in TRIG TV STV 27-74, 27-76
SPOLL example 4-9
SQRT 15-33
SQUare 15-34
Square Brackets 1-5
SRATe 8-21
*SRE (Service Request Enable) 12-18
SRE (Service Request Enable Register) 4-10
Standard Event Status Enable Register (SESER) 4-12
  Bits 12-6
  Default 2-4
Standard Event Status Register
  bits 12-8
Standard Event Status Register (ESR) 4-11
Standard Status Data Structure Model 4-2
STARt | STOP 21-44
STATe 18-21
STATistics 22-149
status 1-22
  of an operation 4-2
Status Byte
  (*STB) 12-20
Status Byte Register 4-8, 4-9
  and serial polling 4-9
  bits 12-21
Status Registers 1-22, 12-3
Status Reporting 4-2
  Bit Definitions 4-3
  Data Structures 4-5
Status Reporting Decision Chart 4-20
STATus, in CALibrate command 10-8
*STB (Status Byte) 12-20
STDDev
  in MEASure HISTogram command 22-77
STIMe 21-45, 21-47
STOP 23-28
STORe 23-29, 23-30

STORe WAVEform 23-31
storing waveform
  sample program 7-13
STRing 14-19
string variables 1-20
  example 1-20
string, quoted 14-13
strings, alphanumeric 1-11
SUBTract 15-35
suffix multipliers 1-12, 3-5
suffix units 3-5
summary bits 4-8
SWEep
  in TRIGger 27-64
syntax error 31-5
SYSTem
  SETup and *LRN 12-11
System Commands 25-2
  DATE 25-3
  DEBug 25-4
  DSP 25-6
  ERRor? 25-7
  HEADer 25-8, 25-10
  LONGform 25-11
  PRESet 25-13
  SETup 25-14
  TIME 25-16
System Computer
  Returning control to 2-12

T

Talker
  Code and Capability 2-5
  Unaddressing 2-12
TEDGe
  in MEASure command 22-150
temperature and calibration 10-3
TER? (Trigger Event Register) 23-32
termination of message during hardcopy 3-4
Terminator 1-13
TEST 19-7
Test (*TST) 12-23
TEXT 14-21
THReshold
  in MEASure FFT command 22-59
THResholds
  ABSolute 22-152
  HYSTeresis 22-154
  METHod 22-156
  PERCent 22-158
  TOPBase 22-160, 22-162
TIEClock2 22-164
TIEData 22-167, 22-172
TIME 25-16
  and RUNT 27-45
  in TRIG DEL TDEL 27-19
  in TRIGger RUNT 27-45
time and date, setting 25-2
Time Base Commands 26-2
  POSition 26-3
  RANGe 26-4
  REFClock 26-5
  REFerence 26-6
  SCALe 26-7
  VIEW 26-8
  WINDow DELay 26-9
  WINDow RANGe 26-12
time buckets
  and POINts? 28-35
time information
  of waveform 7-13
time scale
  operands and functions 15-3
TIMeout SOURce 27-66
TIMeout TIME 27-67
TITLe? 21-46
TMAX 22-173
TMIN 22-175
transferring waveform data 28-2
  sample program 7-10
TRANsition DIRection 27-68
TRANsition SOURce 27-69
TRANsition TIME 27-70, 27-79
TRANsition TYPE 27-71
transmission mode
  and FORMat 28-32
traversal rules 6-5
Tree Traversal
  Examples 6-11
  Rules 6-5
*TRG (Trigger) 12-22
TRG
  bit 12-19, 12-21
  bit in the status byte 4-10
  Event Enable Register 4-4
Trigger
  (*TRG) 12-22
  *TRG status bit 4-4
Trigger Commands 27-2
  TRIG AND:ENABle 27-6
  TRIG AND:SOURce 27-7
  TRIG COMM BWID 27-8
  TRIG COMM ENCode 27-9
  TRIG COMM PATTern 27-10
  TRIG COMM POLarity 27-11
  TRIG COMM SOURce 27-12
  TRIG DEL ARM SLOPe 27-14
  TRIG DEL ARM SOURce 27-13
  TRIG DEL EDEL COUN 27-15
  TRIG DEL EDEL SLOPe 27-17
  TRIG DEL EDEL SOURce 27-16
  TRIG DEL TDEL TIME 27-19
  TRIG DEL TRIG SLOPe 27-21
  TRIG DEL TRIG SOURce 27-20
  TRIG EDGE SLOPe 27-22
  TRIG EDGE SOURce 27-23
  TRIG GLITch POLarity 27-24
  TRIG GLITch SOURce 27-25
  TRIG GLITch WIDTh 27-26
  TRIG HOLDoff 27-27
  TRIG HTHR 27-28
  TRIG HYSTeresis 27-29
  TRIG LEVel 27-30
  TRIG LTHR 27-31
  TRIG PATT CONDition 27-35
  TRIG PATT LOGic 27-36
  TRIG PWID DIR 27-37
  TRIG PWID POL 27-38
  TRIG PWID WIDT 27-41
  TRIG PWIDth SOURce 27-39
  TRIG PWIDth TPOInt 27-40
  TRIG RUNT POLarity 27-42
  TRIG RUNT QUALified 27-43, 27-78
  TRIG RUNT SOURce 27-44
  TRIG RUNTTIME 27-45
  TRIG SHOL CSO 27-46, 27-47, 27-48, 27-49, 27-50, 27-51, 27-52, 27-53
  TRIG SHOL CSO EDGE 27-55
  TRIG SHOL DSO 27-56
  TRIG SHOL HTIMe 27-57
  TRIG SHOL MODE 27-58
  TRIG SHOL STIMe 27-59
  TRIG STATe CLOCk 27-60

TRIG STATe LOGic 27-61
TRIG STATe LTYPe 27-62
TRIG STATe SLOPe 27-63
TRIG SWEep 27-64
TRIG TIM CONDition 27-65
TRIG TIM SOUR 27-66
TRIG TIM TIME 27-67
TRIG TRAN DIRection 27-68
TRIG TRAN SOUR 27-69
TRIG TRAN TIME 27-70, 27-79
TRIG TRAN TYPE 27-71
TRIG TV SOURce 27-75
TRIG TV STV FIELd 27-73
TRIG TV STV LINE 27-72
TRIG TV STV SPOLarity 27-74, 27-76
TRIG TV UDTV ENUMber 27-77
TRIG TV UDTV PGTHan 27-80
TRIG TV UDTV POLarity 27-81
TRIG WIND SOUR 27-83
TRIG WIND TIME 27-84
TRIG WIND TPOI 27-85
TRIG WINDow CONDition 27-82
TRIGger DELay MODE 27-18
TRIGger MODE 27-32
Trigger Event Register (TRG) 4-10
trigger mode 27-5
trigger other instruments 10-6
truncating numbers 1-12
Truncation Rule 6-3
*TST (Test) 12-23
TVOLt 22-177
TYPE query 28-48

U

ULEVel 18-15
ULIMit 18-7, 19-8
Unaddressing all listeners 2-12
UNITinterval 22-180
UNITs 11-47, 11-58
in MTESt AMASk command 21-10
units, vertical 11-47, 11-58
UNKnown vertical units 11-47, 11-58
uppercase 1-11
headers 1-11
letters and responses 1-11
URQ bit (User Request) 12-5
User Request (URQ) status bit 4-3
User Request Bit (URQ) 12-5
User-Defined Measurements 22-7
Using the Digitize Command 1-17
USR bit 12-19, 12-21

V

VAMPlitude 22-182
VAVerage 22-184
VBASe 22-186
version of software, reading 12-9
VERSus 15-36
VERTical 15-37
vertical
axis control 11-2
axis offset, and YRANge 29-9
scaling and functions 15-2
scaling, and YRANge 29-10
vertical axis, full-scale 11-56
vertical units 11-47, 11-58
VIEW 23-33, 26-8, 28-49
VIEW and BLANk 23-12
VLOWer 22-188
VMAX 22-190
VMIDdle 22-192
VMIN 22-194
voltage at center screen 11-31, 11-33, 11-45
voltage information
of waveform 7-13
VOLTS as vertical units 11-47, 11-58
VPP 22-196
VRMS 22-198
VTIMe 22-200
VTOP 22-202
VUPPer 22-204

W

*WAI (Wait-to-Continue) 12-24
Wait-to-Continue (*WAI) 12-24
WATTS as vertical units 11-47, 11-58
waveform
data and preamble 28-4
saving 13-14
storing time and voltage 7-13
time and voltage information 7-13
view parameters 28-50
Waveform Commands 28-2, 30-2
BANDpass? 28-6, 30-4, 30-5, 30-6, 30-7, 30-8, 30-9, 30-13, 30-14, 30-15, 30-16, 30-17, 30-18, 30-19, 30-20, 30-21, 30-22, 30-23, 30-24, 30-25, 30-26, 30-27, 30-31, 30-32, 30-33, 30-34, 30-35, 30-36, 30-37, 30-38, 30-39, 30-40, 30-41, 30-42
BYTeorder 28-7
COMPlete? 28-8
COUNt? 28-9
COUPling? 28-10
FORMat 28-32
POINts? 28-35
TYPE? 28-48
VIEW 28-49
WAVeform SOURce 28-46
XDISplay? 28-51
XINCrement? 28-52
XORigin? 28-53
XRANge? 28-54
XREFerence? 28-55
XUNits? 28-56
YDISplay? 28-57
YINCrement? 28-58
YORigin? 28-59
YRANge? 28-60
YREFerence? 28-61
YUNits? 28-62
Waveform Memory Commands 29-2
DISPlay 29-3, 29-4
LOAD 29-5
SAVE 29-6
XOFFset 29-7
XRANge 29-8
YOFFset 29-9
YRANge 29-10
waveform type
and COMPlete? 28-8
and COUNt? 28-9
and TYPE? 28-48
WAVeforms?
in MTEST COUNt command 21-22
white space (separator) 1-5
WIDTh
and GLITch 27-26
in TRIGger GLITch 27-26
WINDow
and VIEW 28-49
DELay 26-9
in FUNCtion FFT command 15-15

POSition 26-11
RANGe 26-12
SCALe 26-13
WINDow and VIEW 26-8
WINDow SOURce 27-83
WINDow TIME 27-84
WINDow TPOInt 27-85
WORD
and FORMat 28-33
Understanding the format 28-28
writing
quoted strings 14-13
text to the screen 14-19

X

x axis, controlling 26-2
X vs Y 15-36
X1
in MTESt SCALe command 21-39
X1Position 20-5
X1Position|LLIMit
in HISTogram WINDow command 17-9
X1Y1source 20-7
X2Position 20-6, 20-13
X2Position|RLIMit
in HISTogram WINDow command 17-10
X2Y2source 20-9
x-axis
offset, and XOFFset 29-7
range, and XRANge 29-8
units and XUNits 28-56
x-axis duration
and XRANge? 28-54
XDELta
in MTESt AMASk command 21-11
in MTESt SCALE command 21-40
XDELta? 20-11
XDISplay query 28-51
XINCrement query 28-52
XOFFset 29-7
XORigin query 28-53
XRANge 29-8
XRANge query 28-54
XREFerence? 28-55
XUNits query 28-56

Y

Y1
in MTESt SCALe command 21-41
Y1Position 20-12
in HISTogram WINDow command 17-11
Y2
in MTESt SCALe command 21-42
Y2Position
in HISTogram WINDow command 17-12
Y-axis control 11-2
YDELta
in MTESt AMASk command 21-13
YDELta? 20-14
YDISplay? 28-57
YINCrement query 28-58
YOFFset 29-9
YORigin query 28-59
YRANge 29-10
YRANge query 28-60
YREFerence query 28-61
YUNits query 28-62

Z

ZONE
MODE 18-18
PLACement 18-19
SOURce 18-20
STATe 18-21

Safety Notices

This apparatus has been designed and tested in accordance with IEC Publication 1010, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

Warnings

• Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.

• Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock or fire hazard.

• If you energize this instrument by an auto transformer (for voltage reduction or mains isolation), the common terminal must be connected to the earth terminal of the power source.

• Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.

• Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

• Do not install substitute parts or perform any unauthorized modification to the instrument.

• Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.

• Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

• Do not use the instrument in a manner not specified by the manufacturer.

To clean the instrument

If the instrument requires cleaning: (1) Remove power from the instrument. (2) Clean the external surfaces of the instrument with a soft cloth dampened with a mixture of mild detergent and water. (3) Make sure that the instrument is completely dry before reconnecting it to a power source.

Safety Symbols

Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product..

Hazardous voltage symbol.

Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

Agilent Technologies Inc.
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Notices

© Agilent Technologies, Inc. 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.

Manual Part Number
54913-97009, July 2010

Print History

54913-97009, July 2010
54913-97006, August 2008
54913-97003, April 2008
54913-97000, December 2007

Agilent Technologies, Inc.
1601 California Street
Palo Alto, CA 94304 USA

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.

Document 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.

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.

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.

Trademark Acknowledgements

Windows and MS Windows are U.S. registered trademarks of Microsoft Corporation.