Agilent Technologies OTDRs Programming Guide

Agilent Technologies

Notices

Warranty

Edition/Print Date

All Editions and Updates of this manual and their creation dates are listed below.

E4310-91016 Third Edition E0401

First Edition E0298
Second Edition E1098, E0599, E0500
Third Edition E0401

Assistance

Product maintenance agreements and other customer assistance agreements are available for Agilent Technologies products.

For any assistance, contact your nearest Agilent Technologies Sales and Service Office (see *"Service and Support" on page 6*).

ISO 9001 Certification

Produced to ISO 9001 international quality system standard as part of Agilent Technologies' objective of continually increasing customer satisfaction through improved process control.

Bellcore Certification of Excellence

Agilent Technologies is officially designated Bellcore Certification Eligible, and is awarded Bellcore's Certification of Excellence for its OTDR Data Format.

Agilent Technologies GmbH
Herrenberger Str. 130
71034 Böblingen
Germany

About This Manual

This manual contains information about SCPI commands which can be used to program all HP/Agilent Optical Time Domain Reflectometers.

Instruments affected are:

- HP/Agilent E4310A (8147A) OTDR (*Mainframe OTDR*)
- Agilent E6000 Mini-OTDR (*Mini-OTDR*)
- Agilent E6020A Mini-Fiber Break Locator (*Mini-FBL*)
- Agilent E605* and E606* Rack OTDRs (*Rack OTDR*).

Most SCPI commands can be used with all OTDRs, but a few are only applicable to particular instruments, or have slightly different names. For example, commands which may also be used with different Mini-OTDR submodules have an extra number in their name, indicating which submodule is affected.

Each command definition contains text showing which instrument is affected. A command which affects “All” can be used with all the instruments listed above.

The Structure of this Manual

This manual is divided into 4 parts:

- Chapter 1 gives a general introduction to SCPI programming with OTDRs.
- Chapter 2 lists the OTDR-specific SCPI commands.
- Chapters 3 to 5 give fuller explanations and examples of the OTDR-specific commands.
- Chapter 6 gives some example programs showing how the SCPI commands can be used with OTDRs.

In addition, there is an appendix containing information about the HP VEE driver.

Conventions used in this manual

- All commands and typed text is written in Courier font, for example `INIT[:IMM][:ALL]`.
- SCPI commands are written in mixed case: text that you MUST print is written in capitals; text which is helpful but nor necessary is written in lower case.
- So, the command `INITiate[:IMMediate][:ALL]` can be entered either as `init[:imm][:all]`, or as `initiate[:immediate][:all]`. It does not matter whether you enter text using capitals or lower-case letters.
- SCPI commands often contain extra arguments in square brackets. These arguments may be helpful, but they need not be entered.
- So, the command `INITiate[:IMMediate][:ALL]` can be entered as `init` or `initiate`.
- A SCPI command which can be either a command or a query is appended with the text `/?`.

  So, `SYSTem:SET/?` refers to both the command `SYSTem:SET` and the query `SYSTem:SET?`.

Related Publications

You can find more information about the instruments covered by this manual in the following manuals:

- *HP 8147A Optical Time Domain Reflectometer User's Guide* (Agilent Product Number E4310-91011).
- *Agilent E6000C Mini-OTDR User's Guide* (Agilent Product Number E6000-91031)

- *Agilent E6020A Mini-Fiber Break Locator User's Guide* (Agilent Product Number E6020-91011)
- *Agilent E6053A, E6058A and E6060A Rack OTDR User's Guide* (Agilent Product Number E6050-91011).

NOTE Please note that these User Guides no longer contain programming information, and must now be used in conjunction with this manual.

If you are not familiar with the GPIB, then refer to the following books:

- HP publication 5952-0156, *Tutorial Description of HP-IB*.
- ANSI/IEEE-488.1-1978, *IEEE Standard Digital Interface for Programmable Instrumentation*, and ANSI/IEEE-488.2-1987, *IEEE Standard Codes, Formats, and Common Commands*, published by the Institute of Electrical and Electronic Engineers.

In addition, the commands not from the IEEE 488.2 standard are defined according to the Standard Commands for Programmable Instruments (SCPI).

For an introduction to SCPI and SCPI programming techniques, refer to the following documents:

- Hewlett-Packard Press (Addison-Wesley Publishing Company, Inc.): *A Beginners Guide to SCPI* by Barry Eppler.
- The SCPI Consortium: *Standard Commands for Programmable Instruments*, published periodically by various publishers. To obtain a copy of this manual, contact your Agilent TEchnologies representative.

Service and Support

Any adjustment, maintenance, or repair of this product must be performed by qualified personnel. Contact your customer engineer through your local Agilent Technologies Service Center. You can find a list of local service representatives on the Web at: http://www.agilent-tech.com/services/English/index.html

If you do not have access to the Internet, one of these centers can direct you to your nearest representative:

United States	Test and Measurement Call Center (800) 452-4844 (Toll free in US)
Canada	(905) 206-4725
Europe	(31 20) 547 9900
Japan	Measurement Assistance Center (81) 426 56 7832 (81) 426 56 7840 (FAX)
Latin America	(305) 267 4245 (305) 267 4288 (FAX)
Australia/New Zealand	1 800 629 485 (Australia) 0800 738 378 (New Zealand)
Asia-Pacific	(852) 2599 7777 (852) 2506 9285 (FAX)

Front Matter

Front Matter

Front Matter

Introduction to Programming

Introduction to Programming

This chapter introduces some background information that may help you when programming OTDRs. You can find general information about SCPI commands here, and lists and descriptions of some useful IEEE standard common commands.

1.1 Command Messages

A command message is a message from the controller to the OTDR. The following are a few points about command messages:

- Either upper-case or lower-case characters can be used.
- The parts in upper-case characters in the command descriptions must be given. The parts in lower-case characters can also be given, but they are optional.
- The parts in brackets [ ] in the command description can be given, but they are optional.
- In the syntax descriptions the characters between angled brackets (<...>) show the kind of data that you require. You do not type these brackets in the actual command. “<wsp>” stands for a white space character.
- A command message is ended by a line feed character (LF) or `<CR><LF>`.
- Several commands can be sent in a single message. Each command must be separated from the next one by a semicolon “;”.

Units

Where units are given with a command, usually only the base units are specified. The full sets of units are given in the table below.

Unit	Default	Allowed Mnemonics
meters	M	NM, UM, MM, M, KM
miles	MI	MIles
feet	FT	FT, KFT
decibel	DB	MDB, DB
second	S	NS, US, MS, S

The default unit of length is usually mm.

Trace Array

The Mini-OTDR and Rack OTDR can load up to two traces into their memory. The Mainframe OTDR can load up to four traces. These traces form a trace array. One of the entries in this array is always the current entry. Most operations work on this entry.

Data

With the commands you give parameters to the OTDR and receive response values from the OTDR. Unless explicitly noticed these data are given in ASCII format (in fact, only the trace data are given in binary format). The following types of data are used:

- **Boolean** data may only have the values 0 or 1.
- Data of type **short** may have values between -32768 and 32767. When the OTDR returns a short value, it always explicitly gives the sign.
- **Float** variables may be given in decimal or exponential writing (0.123 or 123E-3).
- A **string** is contained between a " at the start and at the end or a ' at the start and at the end. When the OTDR returns a string, it is always included in " " and terminated by <END>.
- When a **register** value is given or returned (for example *ESE), the decimal values for the single bits are added. For example, a value of nine means that bit 0 and bit 3 are set.
- Larger blocks of data are given as **Binary Blocks**, preceded by “#*HLen*Numbytes”, terminated by <END>; *HLen* represents the length of the Numbytes block. For example: #16TRACES<END>.

Message Exchange

The OTDR exchanges messages using an input and an output queue. Error messages are kept in a separate error queue.

The Input Queue

The input queue is a FIFO queue (first-in first-out). Incoming bytes are stored in the input queue as follows:

- Receiving a byte:
    - Clears the output queue.
    - Clears Bit 7 (MSB).
- No modification is made inside strings or binary blocks.

  Outside strings and binary blocks, the following modifications are made:
    - Lower-case characters are converted to upper-case.
    - Two or more blanks are truncated to one.
- The parser is started if the LF character is received or if the input queue is full.

Clearing the Input Queue

Switching the power off causes commands that are in the input queue, but have not been executed to be lost.

The Output Queue

The output queue contains responses to query messages. The OTDR transmits any data from the output queue immediately.

On the Mainframe OTDR, each response message ends with a carriage return (CR, $0D_{16}$) and a LF ($0A_{16}$), with EOI=TRUE. If no query is received, or if the query has an error, the output queue remains empty.

The Error Queue

The error queue is 30 errors long. It is a FIFO queue (first-in first-out). That is, the first error read is the first error to have occurred.

If more than 29 errors are put into the queue, the message '-350, "Queue overflow" ' is placed as the last message in the queue. The queue continues to work, but now with only the first 29 positions.

The oldest error message in the queue is discarded each time a new error message added.

1.2 Common Commands

The IEEE 488.2 standard has a list of reserved commands, called common commands. Some of these commands must be implemented by any instrument using the standard, others are optional. The OTDR implements all the necessary commands, and some optional ones. This section describes the implemented commands.

Common Command Summary

Table 1-1gives a summary of the common commands.

Table 1-1 **Common Command Summary**

Command	Parameter	Function
*CLS		Clear Status Command
*ESE		Standard Event Status Enable Command
*ESE?		Standard Event Status Enable Query
*ESR?		Standard Event Status Register Query
*FTY		Reset defaults and reboot *(not possible on Mainframe OTDR)*
*IDN?		Identification Query
*LRN?		Read instrument settings
*OPC?		Operation Complete Query
*OPT?		Options Query
*RCL	<location>	Recall Instrument Setting
*RST		Reset Command
*SAV	<location>	Save Instrument Setting
*STB?		Read Status Byte Query
*TST?		Self Test Query
*WAI		Wait Command

NOTE **These commands are described in more detail in “IEEE-Common Commands” on page 45**

Common Status Information

There are four registers for the status information. Two of these are status-registers and two are enable-registers. These registers conform to the IEEE Standard 488.2-1987. You can find further descriptions of these registers under *ESE, *ESR?, *SRE, and *STB?. The following figure shows how the registers are organized.

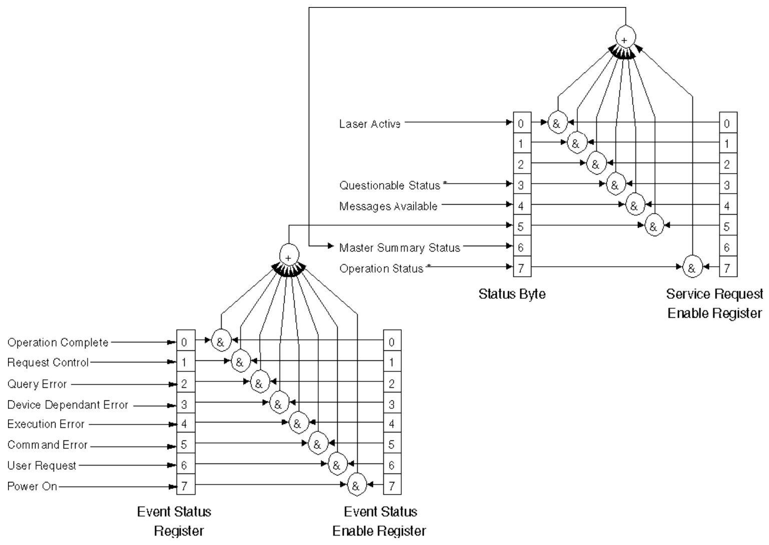


Figure 1-1 **Common Status Registers**

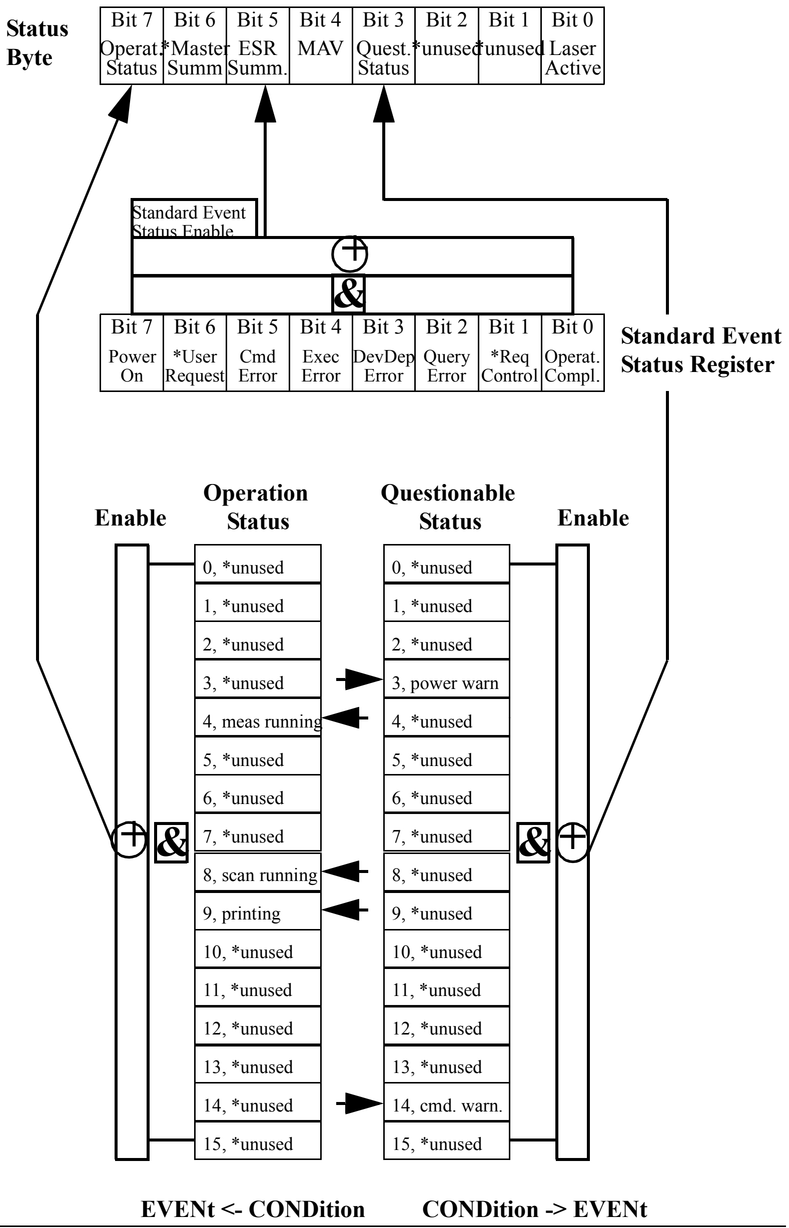
* The questionable and operation status command trees are described in “Status Reporting – The STATus Subsystem” on page 56.

ATTENTION Unused bits in any of the registers return 0 when you read them.

For information about the status model, see “Status Reporting – The STATus Subsystem” on page 56

1.3 HP/Agilent OTDR Status Model

The following figure describes the relevant bit patterns and their relationship of the SCPI status/error model


Status Byte
Bit 7 Operat. Status
Bit 6 *Master Summ
Bit 5 ESR Summ.
Bit 4 MAV
Bit 3 Quest. Status
Bit 2 *unused
Bit 1 *unused
Bit 0 Laser Active
Standard Event Status Enable
&
Bit 7 Power On
Bit 6 *User Request
Bit 5 Cmd Error
Bit 4 Exec Error
Bit 3 DevDep Error
Bit 2 Query Error
Bit 1 *Req Control
Bit 0 Operat. Compl.
Standard Event Status Register
Enable
Operation Status
Questionable Status
Enable
0, *unused
1, *unused
2, *unused
3, *unused
4, meas running
5, *unused
6, *unused
7, *unused
8, scan running
9, printing
10, *unused
11, *unused
12, *unused
13, *unused
14, *unused
15, *unused
0, *unused
1, *unused
2, *unused
3, power warn
4, *unused
5, *unused
6, *unused
7, *unused
8, *unused
9, *unused
10, *unused
11, *unused
12, *unused
13, *unused
14, cmd. warn.
15, *unused
&
&
EVENt <- CONDition
CONDition -> EVENt

Bits marked with * are not used and therefore always set to **0**. The few used bits in the operation are marked with arrows, as are the questionable status registers.

Annotations

Status Byte:

- Bit 0 is set any time the laser is on (measurement running)
- Bits 1 and Bit 2 are unused (0)
- Bit 3 is built from the questionable status event register and its enable mask.
- Bit 4 (MAV) is generally 0.
- Bit 5 is built from the SESR and its SESE.
- Bit 6 is always 0 because the SRE mask is always 0 (no service request).
- Bit 7 is built from the operation status and its enable mask.

Standard Event Status Register

- Bit 0 is set if an operation complete event has been received since the last call to `*ESR?`.
- Bit 1 is always 0 (no service request).
- Bit 2 is set if a query error has been detected.
- Bit 3 is set if a device dependent error has been detected.
- Bit 4 is set if an execution error has been detected.
- Bit 5 is set if a command error has been detected.
- Bit 6 is always 0 (no service request).
- Bit 7 is set for the first call of `*ESR` after Power On.

Operation/Questionable Status

- The Operation/Questionable Status consists of a condition and an event register.
- A "rising" bit in the condition register is copied to the event register.
- A "falling" bit in the condition register has no effect on the event register.
- Reading the condition register is non-destructive.
- Reading the event register is destructive.
- A summary of the event register and its enable mask is set in the status byte.

Operation Status

- Bit 4 is set if a measurement is running, and reset when the measurement is stopped.
- Bit 8 is set if the scan trace is running, and reset when the scan trace is stopped.
- Bit 9 is set if a printout has been started, and reset when the printout is finished or cancelled.
- All other bits are unused, and therefore set to 0.

Questionable Status

- Bit 3 is set if a weak power supply has been detected (DC supply, battery low).
- Bit 14 is set if a questionable command has been received (for example, starting the scan trace or printout with no valid trace data).
- All other bits are unused, and therefore set to 0.

Status Command Summary

*STB?	returns status byte, value 0 .. +255
*ESE	sets the standard event status enable register, parameter 0 .. +255
*ESE?	returns SESE, value 0 .. +255
*ESR?	returns the standard event status register, value 0 .. +255
*OPC?	returns 1 if all operations (scan trace printout, measurement) are completed. Otherwise it returns 0.
*CLS	clears the status byte and SESR, and removes any entries from the error queue.
*RST	clears the error queue, loads the default setting, and restarts communication. **NOTE:** *RST does NOT touch the STB or SESR. A running measurement is stopped.
*TST?	initiates an instrument selftest and returns the results as a 32 bit LONG. If a measurement is running, the status of the latest selftest is returned and an error is set. +0 means "passed". The bits of the 32 bit long integer have the following meaning:

Mini-OTDR and Rack OTDR Bit Table

	<-------- Overall State - "0" means passed, "1" means ST failed or not tested -------->											
Bit 31 ST-Error	Bit 30 Main-frame State	Bit 29 Video State	Bit 28 Batt State	Bit 27 RTC State	Bit 26 SMC State	Bit 25 Check Sum State	Bit 24 Power 6V State	Bit 23 Flash State	Bit 22 Floppy State	Bit 21 DAP State	Bit 20 Sub-Module State	Bit 19 Module State	Bits 18 .. 16 Unused

Bits 15 .. 8 Submodule Error	Bits 7 .. 0 Module Error
<----------------- Error code ----------------->	<----------------- Error code ----------------->

Mainframe OTDR Bit Table

MSW:	Bit 31 Selftest ERROR	Bits 30 .. 26 unused					Bit 25 Module Init failed	Bit 24 IBI-test failed
	Bit 23 FATAL ST-Error	Bit 22 ST non-fatal Error	Bit 21 analog summ	Bit 20 digital summ	Bit 19 MOD Temp.	Bit 18 LAS Temp.	Bit 17 APD-L Temp.	Bit 16 APD-H Temp.
LSW:	Bit 15 APD-HV	Bit 14 RCV-OFFS	Bit 13 OFFS HILIN	Bit 12 OFFS Higain	Bit 11 OFFS Logain	Bit 10 RMS HILIN	Bit 9 RMS Higain	Bit 8 RMS Logain
	Bit 7 not used	Bit 6 DAP-ALU	Bit 5 DSP-Code	Bit 4 CAL-Data	Bit 3 LOG-Table	Bit 2 SHOT-RAM	Bit 1 DAP-RAM	Bit 0 DSP-RAM

Other Commands

*RCL	recalls a pre-defined setting. This is the same as “*RCL” on page 21, except that it is read from a harddisk.
*SAV	stores the current setting. This is the same as “*SAV” on page 21, except that it is stored on a harddisk.
*OPT?	returns a string containing the installed options: <FLOPPY opt>, <COLOR opt>. For example, *OPT? → FLOPPY, 0 An uninstalled option returns 0.
*WAI	causes the remote control part of the instrument to wait for at least 2 seconds before continuing to parse commands. This gives the instrument a chance to accomplish pending tasks. The instrument returns to receiving commands after 2 seconds, or the completion of a printout or scan trace or a limited measurement time (averaging time > 0). **NOTE:** During a running measurement *WAI does NOT wait for the scan trace to finish as it runs continuously.
*IDN?	is an identification string, like “*LRN?” on page 21.
*FTY	resets the defaults and reboot *(not possible on Mainframe OTDR)*

2

Specific Commands

Specific Commands

This chapter gives information about the HP/Agilent OTDR remote commands. It lists all the remote commands relating to OTDRs, with a single-line description.

Each of these summaries contains a page reference for more detailed information about the particular command later in this manual.

2.1 Specific Command Summary

The commands are ordered in a command tree. Every command belongs to a node in this tree.

The root nodes are also called the subsystems. A subsystem contains all commands belonging to a specific topic. In a subsystem there may be further subnodes.

All the nodes have to be given with a command. For example in the command `hcop:item:all`

- `HCOPy` is the subsystem containing all commands for controlling the print out,
- `ITEM` is the subnode that provides selecting what should be printed,
- `ALL` is the command selecting everything for the print out.

NOTE

If a command and a query are both available, the command ends `/?`.

So, `disp:brig/?` means that `disp:brig` and `disp:brig?` are both available.

Table 2-1 gives an overview of the command tree. You see the nodes, the subnodes, and the included commands.

Command	Description	Page
ABORt[1/2]	Stops a running measurement.	79
CALCulate:MATH:EXPRession		
:NAME?	Allows calculating loss and attenuation values.	87
:REFLex?	Calculates Reflectance.	87
:SPLice?	Calculates Splice Loss.	88
:TYPE/?	Sets/queries whether Reflection Height or Reflectance is used.	88
DISPLay		
:BRIGhtness/?	Changes or queries the current LCD brightness.	123
:CONTrast/?	Changes or queries the current LCD contrast.	123
:ENABle/?	Enables, disables, or checks the internal LCD.	124
DISPLay[:WINDow]:GRAPhics		
:COLor/?	Changes or queries the trace color.	125
:LTYPe/?	Changes or queries the trace linestyle.	125
DISPLay[:WINDow]:TEXT		
:DATA/?	Sets or requests a comment.	126
DISPLay[:WINDow]:X		
:SCALe/?	Changes or queries the zooming mode (full trace or zoom)	127
DISPLay[:WINDow]:X[:SCALe]		
:PDIVision/?	Changes or queries the scaling of the X-axis.	128
DISPLay[:WINDow]:Y[:SCALe]		
:PDIVision/?	Changes or queries the scaling of the Y-axis.	129
FETCh[:SCALar]		
:POWer[:DC]?	Reads current power meter value (triggers a measurement).	79

Table 2-1 **Specific Command Summary**

Command	Description	Page
HCOPy		
:ABORt	Cancels the current print job.	130
:DESTination/?	Changes or queries the active printer.	130
[:IMMediate]	Immediately starts printing everything selected.	131
HCOPy:ITEM		
:ALL[:IMMediate]	Start printing everything.	132
HCOPy:ITEM[:WINDow]		
[:IMMediate]	Immediately starts printing the parameter window.	132
:STATe/?	Enables or queries printing the parameter window.	132
HCOPy:ITEM[:WINDow]:TEXT		
[:IMMediate]	Immediately starts printing the event table.	133
:STATe/?	Enables or queries printing the event table.	133
HCOPy:ITEM[:WINDow]:TRACe		
[:IMMediate]	Immediately starts printing the trace.	134
:STATe/?	Enables or queries printing the trace.	133
HCOPy:ITEM[:WINDow]:TRACe:GRATicule		
:STATe/?	Enables or queries printing the trace window grid.	135
HCOPy:PAGE		
:SIZE/?	Selects or queries the size of the paper.	136
INITiate[1][:IMMediate]		
[:ALL]	Starts a measurement.	80
INITiate2	Starts a power meter measurement.	80
:CONTinuous/?	Starts or Queries a single/continuous power meter measurement.	80
KEYBoard	Allows the use of a terminal as an external keyboard	81

Table 2-1 **Specific Command Summary (continued)**

Command	Description	Page
MMEMory		
:CATalog?	Returns contents of current directory.	137
:CDIRectory/?	Changes or queries the current directory.	138
:DELete	Deletes a file.	138
:FREE	Reclaims free space.	139
:FREE?	Returns the amount of free space and the amount used	139
:INITialize	Formats the specified storage device	139
:MDIRectory	Creates a directory on the current storage device.	140
:MSIS/?	Changes or queries the current storage device.	141
:NAME/?	Changes or queries the name of the current trace.	141
MMEMory:COPY		
:FILE	Copies a file to a new name/device	138
MMEMory:LOAD		
:FILE?	Returns a Bellcore binary file.	140
:STATe	Loads a settings file.	140
:TRACe	Loads a trace file.	140
MMEMory:SAVE		
:FILE	Downloads a Bellcore binary file	142
MMEMory:STORe		
:STATe	Saves a settings file.	142
:TRACe	Saves a trace file.	142
:TRACe:REVision/?	Sets or requests the Bellcore file revision used.	142
PROGram:EXPLicit		
:CHECk:LIMit/?	Sets or queries the Pass/Fail Test limits	83
:EXECute	Executes a special task.	85

Table 2-1 **Specific Command Summary (continued)**

Command	Description	Page
:NUMBer/?	Sets or requests the threshold in mdB	85
:STATe/?	Controls a running task.	86
READ[:SCALar]		
:POWer[:DC]?	Reads current power meter value (no measurement triggered).	82
SENSe:AVERage		
:COUNt/?	Sets or queries the current averaging time.	89
SENSe:AVERage:COUNt		
:NUMBer/?	Sets or queries the number of averages for measurements	90
SENSe:DETector		
[:FUNCtion]/?	Sets or queries the current measurement mode.	91
[:FUNCtion:]AUTO/?	Enables or checks the auto mode.	92
[:FUNCtion:]OPTimize/?	Sets or queries the optimization mode.	92
:MODE/?	Sets or returns the current Mini-OTDR mode	93
SENSe:DETector:SAMPle		
:DISTance?	Returns the current sample distance.	94
SENSe:FIBer		
:REFRindex/?	Sets or returns the current refractive index.	94
:SCATtercoeff/?	Sets or returns the current scatter coefficient.	95
:TYPE?	Returns the current fiber type.	95
SENSe:POWer		
:FREQuency?	Queries the detected power meter input frequency	96
:REFerence/?	Sets or Queries the power meter reference value.	96
:UNIT/?	Sets or Queries the power meter power units.	98
:WAVelength/?	Sets or Queries the power meter wavelength.	98

Table 2-1 **Specific Command Summary (continued)**

Command	Description	Page
SENSe:POWer:REFerence		
:DISPlay	Takes current power meter value as reference value.	97
:STATe/?	Sets or Queries type of power meter display (relative or absolute).	97
[SOURce:]		
HOFFset/?	Sets or returns the horizontal offset	101
WAVelength[1/2][:CW]/?	Sets or returns the current wavelength.	108
[SOURce:]AM[:INTERNAL]		
:FREQuency[1/2]/?	Sets or returns frequency of chosen source.	100
[SOURce:]MARKer1/2/3		
:POINt/?	Sets or returns the position of the marker.	102
[:STATe]/?	Activates, disables, or checks the marker.	103
SOURce:POWer		
:STATe[1/2]	Switches the laser of the chosen source on or off.	104
:STATE[1/2]?	Queries the state of the chosen source.	104
[SOURce:]PULSe		
:WIDTh/?	Sets or returns the pulsewidth.	104
:WIDTh:LLIMit?	Returns the lower limit of the measurement hardware.	105
:WIDTh:ULIMit?	Returns the upper limit of the measurement hardware.	105
[SOURce:]RANGe		
:LUNit/?	Sets or returns the current length unit.	106
:SPAN/?	Sets or returns the current measurement span.	106
:STARt/?	Sets or returns the current measurement start.	107
[SOURce:]WAVelength[1/2][:CW]		
AVAilable?	Returns the available wavelength(s)	109

Table 2-1 **Specific Command Summary (continued)**

Command	Description	Page
STATus		
:PRESet	Presets all registers and queues.	58
STATus:OPERation		
[:EVENt]?	Returns the event register.	56
:CONDition?	Returns the condition register.	56
:ENABle/?	Sets or queries the enable mask for the event register.	56
STATus:POWer		
:ACDC?	Queries how the battery is powered.	57
:CAPacity?	Returns the power capacity of the battery.	57
:CURRent?	Returns the current of the battery in mA.	58
:REMain?	Returns the operating time in minutes.	58
STATus:QUEStionable		
[:EVENt]?	Returns the event register.	59
:CONDition?	Returns the condition register.	59
:ENABle/?	Sets or queries the enable mask for the event register.	59
SYSTem		
:BRIDge	Passes communication from serial port 1 to serial port 2	61
:DATE/?	Sets or returns the OTDR's internal date.	69
:ERRor?	Returns the contents of the OTDR's error queue.	70
:HELP?	Returns a Help page on a specified topic	70
:KEY/?	Simulates or Returns a key stroke on the OTDR's front panel.	71
:PRESet	Loads a predefined instrument setting.	73
:SET/?	Sets or returns the current setting	73
:TIME/?	Sets or returns the OTDR's internal time.	74
:UPTime?	Returns the time (in seconds) run on the OTDR	74

Table 2-1 **Specific Command Summary (continued)**

Command	Description	Page
:VERSion?	Returns the OTDR’s SCPI version	75
SYSTem:COMMunicate		
:GPIB[:SELF]:ADDRess/?	Sets or returns the OTDR’s GPIB address.	61
SYSTem:COMMunicate:SERial		
:FEED/?	Sends a command or query to serial port 2	64
[:RECeive]:PORT?	Returns the port used (RS232 or RS485) by the Rack OTDR	68
[:RECeive]:SBITS/?	Sets or queries the number of stop bits.	68
SYSTem:COMMunicate:SERial[1|2][:RECeive]		
:BAUD/?	Sets or queries the baud rate.	62
:BITS/?	Sets or queries the number of data bits.	63
:PACE/?	Sets or queries the pace for the communication.	65
:PARity[:TYPE]/?	Sends or returns the parity	66
:PARity:CHECk/?	Activates the parity.	67
TRACe		
:CATalog?	Returns positions and names of currently loaded traces.	110
:DATA?	Reads a complete trace data array.	111
:DELete	Closes the current trace.	117
:DELete:ALL	Closes all loaded traces.	117
:FEED:CONTrol/?	Sets or queries the current trace.	117
:FREE?	Returns the number of unused trace array values.	118
:POINts	Sets the number of samples for the current trace.	118
:POINts?	Returns the number of data points of the current trace.	119
TRACe:DATA		
:FCRetloss?	Returns the Front Connector return loss	112
:LINE?	Reads samples	114

Table 2-1 **Specific Command Summary (continued)**

Command	Description	Page
:TABLe?	Returns an event table.	115
:TABLe:LOCK/?	Sets or queries whether or not event table is locked.	115
:TORL?	Returns the total optical return loss	116
:VALue?	Returns a measured value at a sample point.	116
TRACe:DATA:CHECk		
:TABLe?	Returns a Pass/Fail Test table.	111
:STATe?	Queries the Pass/Fail Test Table state.	112
TRACe:DATA:LANDmark		
:ADD	Adds a landmark	112
:DELete	Deletes a landmark	113
TRAFficdet/?	Sets/queries whether traffic detection is on or off	82

Table 2-1 **Specific Command Summary (continued)**

3

Instrument Setup and Status

Instrument Setup and Status

This chapter gives descriptions of commands that you can use when setting up your OTDR. The commands are split into the following separate subsystems:

- IEEE Specific commands: which were introduced in “Common Commands” on page 20
- **:STATUS**: commands which relate to the status model.
- **:SYSTEM**: commands which control the serial interface and internal data.

Other commands are described in Chapter 4 “Operations on Traces and Measurements”, and Chapter 5 “Mass Storage, Display, and Print Functions”.

3.1 IEEE-Common Commands

“Common Commands” on page 20 gave a brief introduction to the IEEE-common commands which can be used with OTDRs. This section gives fuller descriptions of each of these commands.

command:	***CLS**
syntax:	*CLS
description:	The CLear Status command *CLS clears all the event registers summarized in the Status Byte register. Except for the output queue, all queues summarized in the Status Byte register are emptied. The error queue is emptied. Neither the Standard Event Status Enable register, nor the Service Request Enable register are affected by this command. After the *CLS command the instrument is left in the idle state. The command does not alter the instrument setting.
parameters:	none
response:	none
example:	`*CLS`
affects:	All instruments

command:	***ESE**
syntax:	*ESE<wsp><value>
description:	The standard Event Status Enable command (*ESE) sets bits in the Standard Event Status Enable register. A 1 in a bit in the enable register enables the corresponding bit in the Standard Event Status register. The register is cleared at power-on. The *RST and *CLS commands do not affect the register.
parameters:	The bit value for the register (a **short** or a **float**):

Bit	**Mnemonic**	**Decimal Value**
7 (MSB)	Power On	128
6	User Request	64
5	Command Error	32
4	Execution Error	16
3	Device Dependent Error	8
2	Query Error	4
1	Request Control	2
0 (LSB)	Operation Complete	1

response:	none
example:	`*ESE 21`
affects:	All instruments

command:	***ESE?**
syntax:	*ESE?
description:	The standard Event Status Enable query *ESE? returns the contents of the Standard Event Status Enable register (see *ESE for information on this register).
parameters:	none
response:	The bit value for the register (a **short** value).
example:	`*ESE? → 21<END>`
affects:	All instruments

command: ***ESR?**

syntax: *ESR?

description: The standard Event Status Register query *ESR? returns the contents of the Standard Event Status register. The register is cleared after being read.

parameters: none

response: The bit value for the register (a **short** or a **float**):

Bit	**Mnemonic**	**Decimal Value**
7 (MSB)	Power On	128
6	User Request	64
5	Command Error	32
4	Execution Error	16
3	Device Dependent Error	8
2	Query Error	4
1	Request Control	2
0 (LSB)	Operation Complete	1

example: `*ESR? → 21<END>`

affects: All instruments

command: ***FTY**

syntax: *FTY

description: The FacTorY defaults command *FTY resets your OTDR to the factory defaults and reboots the OTDR.

parameters: none

response: none

example: `*FTY`

affects: Mini-OTDR, Mini-FBL and Rack OTDR

command:	***IDN?**
syntax:	*IDN?
description:	The IDeNtification query *IDN? gets the instrument identification over the interface.
parameters:	none
response:	The identification terminated by <END>:

HP E6000 Mini-Optical Time Domain Reflectometer Mainframe: *nnnnnnnnnn*, Module: *mmmmmmmmmm* SW_Rev *i.j*

HP:	manufacturer
mmmm:	instrument model number (for example E6000)
ssssssss	serial number
rrrrrrrrrr	firmware revision level
SW_Rev *i.j*	Software Revision number, for example 1.1 or 1.0

example:

```
*IDN? → HP E6000 Mini Optical Time Domain
Reflectometer Mainframe 0123456789, Module:
ABCDE54321 SW_Rev 6.0<END>
```

NOTE **The response from *IDN? for Mini FBLs, Rack OTDRs and Mainframe OTDRs is respectively:**

`Agilent E6020A Fiber Break Locator Instrument...`

`HP E60`*`xx`*`A Rack Optical Time Domain Reflectometer...`

and

`HP 8147 Optical Time Domain Reflectometer...`

affects: All instruments

command:	***LRN?**
syntax:	*LRN?
description:	The LeaRN query *LRN? reads the complete instrument setting in a binary block. The binary block can be directly stored as a setting file.
parameters:	none
response:	Binary block.
example:	`*LRN?` → *binblock*
affects:	All instruments

command:	***OPC?**
syntax:	`*OPC?`
description:	The OPeration Complete query *OPC? parses all program message units in the input queue. If a print, measurement or scan trace is active, *OPC? returns 0. Otherwise, *OPC? returns 1. The following actions cancel the *OPC? query (and put the instrument into Operation Complete, Command Idle State): • Power-on • the Device Clear Active State is asserted on the interface. • *CLS • *RST
parameters:	none
response:	0<END> *print, measurement, Scan Trace active*, or 1<END>
example:	`*OPC?` → `1<END>`
affects:	All instruments

command: ***OPT?**

syntax: *OPT?

description: The OPTions query *OPT? gets a list of the installed options over the interface. All three options are always listed, in the same order, separated by commas. If an option is not installed in the instrument, 0 is sent in its position in the list.

parameters: none

response: **Mini-OTDR / Mini-FBL response:**

module-type|0, FLOPPY|0, COLOR|0, EXTFLASH|0, *submodule-type* : *submodule serial no*|0

Rack OTDR response:

module-type|0, FLOPPY|0, COLOR|0, EXTFLASH|0, *submodule-type* : *submodule serial no*|0 RS232|RS485

Mainframe OTDR response:

module-type|0, DC|0, PRINTER|0, COLOR|0, HPIB|0, LAN|0

NOTE **The second and third arguments for the Rack OTDR (FLOPPY and COLOR) are included for the sake of consistency.**

The Rack OTDR has no floppy option, and is always configured as a color unit.

NOTE **In this release of the Mini-OTDR and Rack OTDR, the fourth argument (EXTFLASH) will always be 0.**

example: **Mini-OTDR / Mini-FBL example:**

```
*OPT? → E6003A, FLOPPY, 0, 0, E6006A :
DE13A00108<END>
```

Rack OTDR example:

```
*OPT? → E6053A, 0, 0, 0, 0 RS485<END>
```

Mainframe OTDR example:

```
*OPT? → E4316A, DC, 0, 0, HPIB, LAN<END>
```

affects: All instruments

command:	***RCL**
syntax:	*RCL<wsp><location>
description:	The instrument setting is changed to one saved on the internal storage device. Saved settings are in the form *n*.SET, so `*RCL 2` recalls setting `SET2.SET`.
parameters:	a **short** value (between 0 and 5) giving the number of the setting to be saved.
response:	none
related commands	*SAV
example:	`*RCL 3`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	***RST**
syntax:	*RST
description:	The ReSeT command *RST sets the instrument to reset setting (standard setting) stored in internal storage. Pending *OPC? actions are cancelled. The instrument is placed in the idle state awaiting a command. The *RST command clears the error queue. The following are not changed: • Output queue • Service Request Enable register (SRE) • Standard Event Status Enable register (ESE) The following parameters are reset • **Start**: 0 km (Auto) • **Stop**: 2 km (Auto) (Mini and Rack); 40 km (Auto) (Mainframe OTDR) • **Pulsewidth**: 1 µs (Auto) • **First Wavelength**: 1310 nm • **Refractive Index**, **Scatter Coefficient**: nominal for 1310 nm • **Measurement Mode**: Averaging • **Averaging Time**: 3 min (Mini and Rack); unlimited (Mainframe OTDR) • **Optimize Mode**: Standard • **Data Points**: 16000 • **Front Connector Threshold**: -30 dB • **Reflective Threshold**: 0 • **Non-Reflective Threshold**: 0 • **End Threshold**: 5 dB (Mini and Rack); 3 dB (Mainframe OTDR)
parameters:	none
response:	none
example:	`*RST`
affects:	All instruments

command:	***SAV**
syntax:	*SAV<wsp><location>
description:	With the SAVe command *SAV the instrument setting is stored on the internal storage device. The instrument can store 4 settings, in locations 1 to 4. The scope of the saved setting is identical to the standard setting (see *RST). Settings are in the form *n*.SET, so `*SAV 2` saves the current setting as `SET2.SET`.
parameters:	a **short** value (between 0 and 5) giving the number of the setting to be saved.
related commands:	`*RCL`
response:	none
example:	`*SAV 3`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	***STB?**
syntax:	*STB?
description:	The STatus Byte query *STB? returns the contents of the Status Byte register. The Master Summary Status (MSS) bit is true when any enabled bit of the STB register is set (excluding Bit 6). The Status Byte register including, the master summary bit, MSS, is not directly altered because of an *STB? query.
parameters:	none
response:	The bit value for the register (a **short** value):

Bit	**Mnemonic**	**Decimal Value**
7 (MSB)	Operation Status	128
6	Master Summary Status	64
5	Event Status Bit	32
4	Message Available	16
3	Questionable Status	8
2	Not used	0
1	Not used	0
0 (LSB)	Laser Active Bit	1

example:	`*STB? → 1<END>`
affects:	All instruments

command:	***TST?**
syntax:	*TST?
description:	The self-TeST query *TST? makes the instrument perform a self-test and place the results of the test in the output queue. No further commands are allowed while the test is running. After the self-test the instrument is returned to the setting that was active at the time the self-test query was processed.
parameters:	none
response:	The sum of the results for the individual tests (a **32-bit signed integer** value):
example:	`*TST? → 0<END>`
affects:	All instruments

command:	***WAI**
syntax:	*WAI
description:	The WAIt command *WAI prevents the instrument from executing any further commands until the current command has finished executing. All pending operations are completed during the wait period.
parameters:	none
response:	none
example:	`*WAI`
affects:	All instruments

3.2 Status Reporting – The STATus Subsystem

The Status subsystem allows you to return and set details from the Status Model. For more details, see “HP/Agilent OTDR Status Model” on page 23

command:	**STATus:OPERation[:EVENt]?**
syntax:	STATus:OPERation[:EVENt]?
description:	Queries the operation event register
parameters:	none
response:	The bit value for the operation event register as a **short** value (0 .. +32767)
example:	`stat:oper?` → +0<END>
affects:	All instruments

command:	**STATus:OPERation:CONDition?**
syntax:	STATus:OPERation:CONDition?
description:	Queries the operation condition register
parameters:	none
response:	The bit value for the operation condition register as a **short** value (0 .. +32767)
example:	`stat:oper:cond?` → +16<END>
affects:	All instruments

command:	**STATus:OPERation:ENABle**
syntax:	STATus:OPERation:ENABle<wsp><value>
description:	Sets the operation enable mask for the event register
parameters:	The bit value for the operation enable mask as a **short** value (0 .. +32767)
response:	none
example:	`stat:oper:enab 128`
affects:	All instruments

command:	**STATus:OPERation:ENABle?**
syntax:	STATus:OPERation[:ENABle]?
description:	Returns the operation enable mask for the event register
parameters:	none
response:	The bit value for the operation enable mask as a **short** value (0 .. +32767)
example:	`stat:oper:enab?` → +128<END>
affects:	All instruments

command:	**STATus:POWer:ACDC?**
syntax:	STATus:POWer:ACDC?
description:	Queries how the battery is powered.
parameters:	none
response:	AC, DC or CHARging
example:	`stat:pow:acdc?` → AC<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**STATus:POWer:CAPacity?**
syntax:	STATus:POWer:CAPacity?
description:	Returns the power capacity of the battery.
parameters:	none
response:	percentage capacity of the battery
example:	`stat:pow:cap?` → 75%<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**STATus:POWer:CURRent?**
syntax:	STATus:POWer:CURRent?
description:	Returns the current of the battery in mA.
parameters:	none
response:	Battery current

NOTE **If the battery is discharging, the returned value will be negative.**

If the battery is charging, the returned value will be positive.

example:	`stat:pow:curr?` → 200MA<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**STATus:POWer:REMain?**
syntax:	STATus:POWer:REMain?
description:	Returns the operating time in minutes
parameters:	none
response:	Remaining time
example:	`stat:pow:rem?` → 180MIN<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**STATus:PRESet**
syntax:	STATus:PRESet
description:	Resets both enable masks to 0.
parameters:	none
response:	none
example:	`stat:pres`
affects:	All instruments

command:	**STATus:QUEStionable[:EVENt]?**
syntax:	STATus:QUEStionable[:EVENt]?
description:	Queries the questionable event register
parameters:	none
response:	The bit value for the questionable event register as a **short** value (0 .. +32767)
example:	`stat:ques?` → +0<END>
affects:	All instruments

command:	**STATus:QUEStionable:CONDition?**
syntax:	STATus:QUEStionable:CONDition?
description:	Queries the condition register
parameters:	none
response:	The bit value for the questionable condition register as a **short** value (0 .. +32767)
example:	`stat:ques:cond?` → +8<END>
affects:	All instruments

command:	**STATus:QUEStionable:ENABle**
syntax:	STATus:QUEStionable:ENABle<wsp><value>
description:	Sets the questionable enable mask for the event register
parameters:	The bit value for the questionable enable mask as a **short** value (0 .. +32767)
response:	none
example:	`stat:ques:enab 128`
affects:	All instruments

command:	**STATus:QUEStionable:ENABle?**
syntax:	STATus:QUEStionable[:ENABle]?
description:	Returns the questionable enable mask for the event register
parameters:	none
response:	The bit value for the questionable enable mask as a **short** value (0 .. +32767)
example:	`stat:ques:enab?` → +128<END>
affects:	All instruments

3.3 Interface/Instrument Behaviour Settings – The SYSTem Subsystem

The SYSTem subsystem lets you control the instrument’s serial interface. You can also control some internal data (like date, time zone, and so on)

command:	**SYSTem:BRIDge**
syntax:	SYSTem:BRIDge
description:	Allows you to send and receive data from the instrument connected to Serial1 to the instrument connected to Serial 2. Data characters are passed between Serial 1 and Serial 2 until the command `#SCPI` is detected.
parameters:	none
response:	none
example:	`syst:brid`
affects:	Rack OTDR only

command:	**SYSTem:COMMunicate:GPIB[:SELF]:ADDRess**
syntax:	SYSTem:COMMunicate:GPIB[:SELF]:ADDRess<wsp><value>
description:	Sets the OTDR’s GPIB address.
parameters:	Valid values for the address are 1 .. 32 (a **short** value).
response:	none
example:	`syst:comm:gpib:addr 15`
affects:	Mainframe OTDR only

.

command:	**SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?**
syntax:	SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?
description:	Queries the OTDR's current GPIB address.
parameters:	none
response:	Possible values for the address are 1 .. 32 (a **short** value).
example:	`syst:comm:gpib:addr? → +15<END>`
affects:	Mainframe OTDR only

command:	**SYSTem:COMMunicate:SERial[1|2][:RECeive]:BAUD**
syntax:	SYSTem:COMMunicate:SERial[:RECeive]:BAUD<wsp><value>
description:	Sets the baud rate for the OTDR serial interface

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only. If you are using a Rack OTDR, and you do not specify a serial port number, the baud rate for Serial 1 is set.**

NOTE **All changes take effect immediately. After this command, you must reconfigure your RS232 to continue communication.**

parameters:	Valid baud rates are 1200, 2400, 9600, 19200,38400, 57600, 115200.
response:	none
example:	`syst:comm:ser:baud 9600`
affects:	All instruments

command:	**SYSTem:COMMunicate:SERial[1|2][:RECeive]:BAUD?**
syntax:	SYSTem:COMMunicate:SERial[1|2][:RECeive]:BAUD?
description:	Returns the current baud rate for the OTDR serial interface

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only.
If you are using a Rack OTDR, and you do not specify a serial port number, the baud rate for Serial 1 is returned.**

parameters:	none
response:	Possible baud rates are 1200, 2400, 9600, 19200, 38400, 57600, 115200
example:	`syst:comm:ser:baud?` $\rightarrow$ +9600<END>
affects:	All instruments

command:	**SYSTem:COMMunicate:SERial[:RECeive]:BITS**
syntax:	SYSTem:COMMunicate:SERial[:RECeive]:BITS<wsp><value>
description:	Sets the number of data bits for the OTDR’s serial interface.

NOTE **All changes take effect immediately. After this command, you must reconfigure your RS232 to continue communication.**

parameters:	Valid numbers are 5 .. 8
response:	none
example:	`syst:comm:ser:bits 6`
affects:	Mainframe OTDR only

command:	**SYSTem:COMMunicate:SERial[:RECeive]:BITS?**
syntax:	SYSTem:COMMunicate:SERial[:RECeive]:BITS?
description:	Returns the number of data bits for the OTDR’s serial interface.
parameters:	none
response:	Possible numbers are 5 .. 8
example:	`syst:comm:ser:bits` $\rightarrow$ `+6<END>`
affects:	Mainframe OTDR only

command:	**SYSTem:COMMunicate:SERial:FEED**
syntax:	SYSTem:COMMunicate:SERial:FEED<wsp><command>
description:	Send a command to the instrument connected to Serial 2
parameters:	The command given as a text string in "".
response:	none
example:	`syst:comm:ser:feed "init"`
affects:	Rack OTDR and Mainframe OTDR

command:	**SYSTem:COMMunicate:SERial:FEED?**
syntax:	SYSTem:COMMunicate:SERial:FEED?<wsp><query>
description:	Send a query to the instrument connected to Serial 2
parameters:	The query given as a text string in "".
response:	none
example:	`syst:comm:ser:feed? "*idn?" → HP E6000 Mini-Optical Time Domain Reflectometer Mainframe 0123456789, Module: ABCDE54321 SW_Rev 6.0<END>`
affects:	Rack OTDR only

command:	**SYSTem:COMMunicate:SERial[1|2][:RECeive]:PACE**
syntax:	SYSTem:COMMunicate:SERial[1|2][:RECeive]:PACE<wsp> <pace>
description:	Sets the pace for the OTDR serial interface

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only.**
If you are using a Rack OTDR, and you do not specify a serial port number, the pace for Serial 1 is set.

You cannot use this command with a Rack OTDR Option 006 (RS485), as this does not have hardware handshaking.

NOTE **All changes take effect immediately. After this command, you must reconfigure your RS232 to continue communication.**

parameters:	Valid values are NONE, HARDware, XONXoff.

NOTE **XONX is only available with the Mainframe OTDR.**
However, for binary disk transfers HARD is recommended, and XONX is forbidden

response:	none
example:	`syst:comm:ser:pace hard`
affects:	All instruments

command:	**SYSTem:COMMunicate:SERial[1|2][:RECeive]:PACE?**
syntax:	SYSTem:COMMunicate:SERial[1|2][:RECeive]:PACE?
description:	Returns the pace for the OTDR serial interface

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only. If you are using a Rack OTDR, and you do not specify a serial port number, the pace for Serial 1 is requested.**

parameters:	none
response:	Possible values are NONE, HARDware, and XONXoff.

NOTE **XONX is only available with the Mainframe OTDR.**

example:	`syst:comm:ser:pace? → HARD<END>`
affects:	All instruments

command:	**SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity [:TYPE]**
syntax:	SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity[:TYPE]<wsp> <parity>
description:	Sets the type of parity checking for the OTDR’s serial interface.

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only. If you are using a Rack OTDR, and you do not specify a serial port number, the parity type for Serial 1 is set.**

NOTE **All changes take effect immediately. After this command, you must reconfigure your RS232 to continue communication.**

parameters:	Valid values are NONE, ODD, EVEN.
response:	none
example:	`syst:comm:ser:par odd`
affects:	All instruments

command: **SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity [:TYPE]?**

syntax: SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity[:TYPE]?

description: Returns the type of parity checking for the OTDR's serial interface.

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only.**
If you are using a Rack OTDR, and you do not specify a serial port number, the parity type for Serial 1 is requested.

parameters: none

response: Possible values are NONE, ODD, EVEN.

example: `syst:comm:ser:par? → ODD<END>`

affects: All instruments

command: **SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity: CHECk**

syntax: SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity:CHECk<wsp> <boolean>

description: Determines whether parity checking is enabled for the OTDR's serial interface.

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only.**
If you are using a Rack OTDR, and you do not specify a serial port number, the parity for Serial 1 is checked.

parameters: Possible values are 0 and 1

response: none

example: `syst:comm:ser:par:chec 1`

affects: All instruments

command:	**SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity :CHECk?**
syntax:	SYSTem:COMMunicate:SERial[1|2][:RECeive]:PARity:CHECk?
description:	Queries whether parity checking is enabled for the OTDR's serial interface.

NOTE **You can choose Serial 1 or 2 for the Rack OTDR only. If you are using a Rack OTDR, and you do not specify a serial port number, the parity checking state for Serial 1 is requested.**

parameters:	none
response:	Possible values are 0: checking disabled 1: checking enabled
example:	`syst:comm:ser:par:chec? → 1<END>`
affects:	All instruments

command:	**SYSTem:COMMunicate:SERial:PORT?**
syntax:	SYSTem:COMMunicate:SERial:PORT?
description:	Inquires the type of second serial port that is configured (Rack OTDR only).
parameters:	none
response:	RS232 or RS485
example:	`syst:comm:ser:port? → RS485`<END>
affects:	Rack OTDR only

command:	**SYSTem:COMMunicate:SERial[:RECeive]:SBITS**
syntax:	SYSTem:COMMunicate:SERial[:RECeive]:SBITS<wsp><bits>
description:	Sets the number of stop bits for the OTDR's serial interface.

NOTE **All changes take effect immediately. After this command, you must reconfigure your RS232 to continue communication.**

parameters:	Valid numbers are ONE, ONEHalf, TWO
response:	none
example:	`syst:comm:ser:sbit two`
affects:	Mainframe OTDR only

command:	**SYSTem:COMMunicate:SERial[:RECeive]:SBITS?**
syntax:	SYSTem:COMMunicate:SERial[:RECeive]:SBITS?
description:	Returns the number of stop bits for the OTDR’s serial interface.
parameters:	none
response:	Possible values are ONE, ONEHalf, TWO
example:	`syst:comm:ser:sbit? → TWO<END>`
affects:	Mainframe OTDR only

command:	**SYSTem:DATE**
syntax:	SYSTem:DATE<wsp><day>,<month>,<year>
description:	Sets the OTDR’s internal date.
parameters:	The date in the format day, month,year (**short** values)
response:	none
example:	`syst:date 20,7,1995`
affects:	All instruments

command:	**SYSTem:DATE?**
syntax:	SYSTem:DATE?
description:	Returns the OTDR’s internal date.
parameters:	none
response:	The date in the format day, month,year (**short** values)
example:	`syst:date?` → +20,+7,+1995<END>
affects:	All instruments

command:	**SYSTem:ERRor?**
syntax:	SYSTem:ERRor?
description:	Returns the contents of the OTDR’s error queue. Removes the returned entry from the queue.
parameters:	none
response:	The number of the latest error, and its meaning.
example:	`syst:err?` → -113,"Undefined header"<END>
affects:	All instruments

command:	**SYSTem:HELP?**
syntax:	SYSTem:HELP?<wsp><keyword>
description:	Returns a help page corresponding to the specified keyword.
parameters:	keyword given as a string in "". For example, "SYSTem", "SOURce", "DISPlay", "IEEEcommon". "" returns a list of valid keywords.
response:	A Binary block containing the help page.
example:	`syst:help? "syst"` → #3316[help_page]<END>
affects:	All instruments

command:	**SYSTem:KEY**
syntax:	SYSTem:KEY<wsp><code>
description:	Simulates keystrokes on the OTDR's frontpanel.
parameters:	Valid key codes are as follows:

Mini-OTDR / Mini-FBL	**Rack OTDR**	**Mainframe OTDR**
0:Select key.	0: Enter/Return	0: Enter (RPG-click)
1:Run/Stop key.	1: *<f2>*	1: Softkey 1 (topmost)
2:Up key	2: Up arrow	2: Softkey 2
3:Down key	3: Down arrow	3: Softkey 3
4:Left key	4: Left arrow	4: Softkey 4
5:Right key	5: Right arrow	5: Softkey 5
6:Help key	6: *<f1>*	6: Softkey 6
		7: Help
		8: Zoom Horizontal Out
		9: Zoom Vertical In
		10: Zoom Vertical Out
		11: Zoom Horizontal In
		12: Next marker
		13: Print
		14: Full Trace
		15: Save
		16: Trace/Event
		17: Around Marker
		18: Auto
		19: Run/Stop
		20: Decrease Brightness
		21: Increase Brightness

response:	none
example:	`syst:key? 1`<END>
affects:	All instruments

command:	**SYSTem:KEY?**
syntax:	SYSTem:KEY?
description:	Returns either the last keystroke entered on the OTDR frontpanel (not possible on Mainframe OTDR), or the last keystroke emulated by the `SYSTem:KEY` remote command (all instruments).
parameters:	none
response:	Valid key codes are as follows:

Mini-OTDR / Mini-FBL	**Rack OTDR**	**Mainframe OTDR**
0:Select key.	0: Enter/Return	0: Enter (RPG-click)
1:Run/Stop key.	1: *<f2>*	1: Softkey 1 (topmost)
2:Up key	2: Up arrow	2: Softkey 2
3:Down key	3: Down arrow	3: Softkey 3
4:Left key	4: Left arrow	4: Softkey 4
5:Right key	5: Right arrow	5: Softkey 5
6:Help key	6: *<f1>*	6: Softkey 6
		7: Help
		8: Zoom Horizontal Out
		9: Zoom Vertical In
		10: Zoom Vertical Out
		11: Zoom Horizontal In
		12: Next marker
		13: Print
		14: Full Trace
		15: Save
		16: Trace/Event
		17: Around Marker
		18: Auto
		19: Run/Stop
		20: Decrease Brightness
		21: Increase Brightness

example:	`syst:key?` → 1<END>
affects:	All instruments

command:	**SYSTem:PRESet**
syntax:	SYSTem:PRESet
description:	Loads a predefined instrument setting that is also loaded on power on.
parameters:	none
response:	none
example:	syst:pres
affects:	All instruments

command:	**SYSTem:SET**
syntax:	SYSTem:SET<wsp><setting>
description:	Sets the specified instrument setting from a binary block.
parameters:	binary block
response:	none
example:	`syst:set` *binblock*
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**SYSTem:SET?**
syntax:	SYSTem:SET?
description:	Reads the complete instrument setting in a binary block. The binary block can be directly stored as a setting file.
parameters:	none
response:	binary block
example:	`syst:set?` → *binblock*
affects:	All instruments

command:	**SYSTem:TIME**
syntax:	SYSTem:TIME<wsp><hour>,<minute>,<second>
description:	Sets the OTDR's internal time.
parameters:	The time in the format hour,minute,second. Hours are counted 0...23 (**short** values).
response:	none
example:	`syst:time 20,15,30`
affects:	All instruments

command:	**SYSTem:TIME?**
syntax:	SYSTem:TIME?
description:	Returns the OTDR's internal time.
parameters:	none
response:	The time in the format hour,minute,second. Hours are counted 0...23 (**short** values).
example:	`syst:time?` → +20,+15,+30<END>
affects:	All instruments

command:	**SYSTem:UPTime?**
syntax:	SYSTem:UPTime?
description:	Returns the time (in seconds) since you switched on your OTDR.
parameters:	none
response:	The time in seconds (**int32** value).
example:	`syst:upt?` → 240<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SYSTem:VERSion?**
syntax:	SYSTem:VERSion?
description:	Returns the SCPI revision to which the OTDR complies.
parameters:	none
response:	The revision year and number.
example:	`syst:vers?` $\rightarrow$ 1995.0<END>
affects:	All instruments

4

Operations on Traces and Measurements

Operations on Traces and Measurements

This chapter gives descriptions of commands that you can use when taking traces and measurements from your OTDR. The commands are split into the following separate subsystems:

- Root level commands: general commands.
- **:PROGRAM/:CALCULATE**: commands which execute tasks or calculate values.
- **:SENSE**: commands which control measurement parameters.
- **:SOURCE**: commands which control the optical source and markers.
- **:TRACE**: commands which relate to the traces in the OTDR's memory.

Other commands are described in Chapter 3 "Instrument Setup and Status", and Chapter 5 "Mass Storage, Display, and Print Functions".

4.1 Root Layer Commands

command:	**ABORt[1/2]**
syntax:	ABORt[1/2]
description:	Stops a running measurement: `abor` or `abor1`: on the OTDR `abor2`: on the Visual Fault Finder

NOTE **You cannot use a Visual Fault Finder with a Mainframe OTDR. This means that `abor2` is not available.**

parameters:	none
response:	none
example:	`abor`
affects:	All instruments

command:	**FETCh[:SCAlar]:POWer[:DC]?**
syntax:	FETCh[:SCALar]:POWer[:DC]?
description:	Reads the current power meter value.

NOTE **If the power meter is not running, a measurement is triggered.**

parameters:	none
response:	The reference as a **float** value in dBm, W or dB.

NOTE **If the reference state is absolute, units are dBm or W.**
If the reference state is relative, units are dB.

example:	`fetc:pow?` → +4DBM<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**INITiate[1|2][:IMMediate][:ALL]**
syntax:	INITiate[1|2][:IMMediate][:ALL]
description:	Starts a measurement: `init` or `init1`: internal source `init2`: power meter

NOTE **You cannot use a Visual Fault Finder with a Mainframe OTDR. This means that `init2` is not available.**

parameters:	none
response:	none
example:	`init`
affects:	All instruments

command:	**INITiate2[:IMMediate]:CONTinuous**
syntax:	INITiate2[:IMMediate]:CONTinuous<wsp><boolean>
description:	Starts a power meter measurement.
parameters:	A **boolean** value: 0 – single measurement made 1 – continuous measurement made
response:	none
example:	`init2:cont 1`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**INITiate2[:IMMediate]:CONTinuous?**
syntax:	INITiate2[:IMMediate]:CONTinuous?
description:	Queries whether power meter measurement is continuous
parameters:	none
response:	A **boolean** value: 0 – single measurement 1 – continuous measurement
example:	`init2:cont?` → 1<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**KEYBoard**
syntax:	KEYBoard
description:	Allows the use of a terminal as an external keyboard
parameters:	none
response:	none
example:	`keyb`

NOTE **`keyb` allows you to add text from a terminal (for example, when specifying the name of a file to be saved). To use this facility, you should do the following:**

1. **Attach your OTDR to a terminal. In this context, a terminal is any PC or palmtop running a terminal program. The terminal should have its own keyboard.**

   You can attach the terminal using an RS232 cable. For details of attaching an RS232 cable to an OTDR, see the appropriate Guide.

2. **Enter `keyb` from your terminal keyboard.**
3. **Enter text as required from your terminal keyboard. All text is treated literally until you enter `<CTRL>Z` (ASCII character 26) (see below).**
4. **To finish entering text, enter `<CTRL>Z` from your terminal keyboard.**

For example, after [File]<Save As..>`New Name`, you see a keyboard on the OTDR screen. Instead of using this keyboard you can enter the following text from your terminal:

```
keyb
T1.SOR
^Z
```

This is the equivalent of entering `T1.SOR` from the screen keyboard.

affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**READ[:SCAlar]:POWer[:DC]?**
syntax:	READ[:SCALar]:POWer[:DC]?
description:	Reads the current power meter value.

NOTE **The power meter must be running for this command to be effective**

parameters:	none
response:	The reference as a **float** value in dBm, W or dB.

NOTE **If the reference state is absolute, units are dBm or W. If the reference state is relative, units are dB.**

example:	`read:pow?` → +4DBM<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**TRAFficdet**
syntax:	TRAFficdet<wsp><onoff>
description:	Turn traffic detection on or off
parameters:	ON: turn traffic detection on OFF: turn traffic detection off.
response:	none
example:	`traf on`
affects:	Mini-OTDR and Rack OTDR

command:	**TRAFficdet?**
syntax:	TRAFficdet?
description:	Queries whether traffic detection is on or off
parameters:	none
response:	ON: traffic detection is on OFF: traffic detection is off.
example:	`traf?` → `ON<END>`
affects:	Mini-OTDR and Rack OTDR

4.2 Playing With Data – The PROGram and CALCulate Subsystems

The PROGram and CALCulate subsystems allow you to execute special tasks and calculating several loss and attenuation values

command: **PROGram:EXPLicit:CHECk:LIMit**

syntax: PROGram:EXPLicit:CHECk:LIMit<wsp><param><wsp><value>

description: Set the Pass/Fail test limits for the specified parameter.

parameters: Valid values are as follows.

Units	**Units**	**Limit**
REFLective	mdB	10000 .. 65000
NONReflective	mdB	0 .. 5000
ATTenuation	mdB/km	0 .. 5000
CONNector loss	mdB	0 .. 5000
LOSS	mdB	0 .. 50000
LENGTh	mm	0 .. 500000000
TOLerance	mm	0 .. 50000000
NEW events	0=off, non-zero=on	
SORT results	0=severity, 1=distance	0 or 1

The units specified above are implied, so you may only enter a positive integer within the specified limits.

NOTE **For more information about the Pass/Fail test limits, please consult the *E6000C Mini-OTDR User's Guide* (English Agilent Product number E6000-91031).**

response: none

example: `prog:expl:chec:lim refl 30000`

affects: Mini-OTDR and Rack OTDR

.

command:	**PROGram:EXPLicit:CHECk:LIMit?**
syntax:	PROGram:EXPLicit:CHECk:LIMit?<wsp><param>
description:	Query the Pass/Fail test limits for the specified parameter.
parameters:	Valid units are: REFLective NONReflective ATTenuation CONNector loss LOSS LENGTh TOLerance NEW events SORT results
response:	The units and limits as the same as for PROGram:EXPLicit:CHECk:LIMit on page 83.

NOTE **For more information about the Pass/Fail test limits, please consult the *E6000C Mini-OTDR User's Guide* (English Agilent Product number E6000-91031).**

example:	`prog:expl:chec:lim? refl`→ -30000<END>
affects:	Mini-OTDR and Rack OTDR

command:	**PROGram:EXPLicit:EXECute**
syntax:	PROGram:EXPLicit:EXECute<wsp><task>
description:	Allows executing special tasks on the OTDR.
parameters:	A string specifying the task. Currently only `"scan"` is valid on all instruments. On the Mini-OTDR and Rack OTDR, you can also enter `"check"` to start the Pass/Fail test. Valid units are: SCAN: run a scan trace (not Mini-FBL) CHECK: run a Pass/Fail Test (not Mini-FBL) ORL?: inquire Optical Return Loss (not Mini-FBL) TORL?: inquire Total ORL of a Link

NOTE **Because this command does not accept character data, you must put quotation marks around the parameter `scan` or `check`.**

response:	none
example:	`prog:expl:exec "scan"`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**PROGram:EXPLicit:NUMBer**
syntax:	PROGram:EXPLicit:NUMBer<wsp><type>,<value>
description:	Sets the threshold.
parameters:	REFLective, NONReflective, or END threshold value (**int32**) in mdB On the Mini-FBL, only END is valid.
response:	none
example:	`prog:expl:numb refl, 60000`
affects:	All instruments

command:	**PROGram:EXPLicit:NUMBer?**
syntax:	PROGram:EXPLicit:NUMBer?<wsp><type>
description:	Requests the threshold value.
parameters:	REFLective, NONReflective, or END On the Mini-FBL, only END is possible.
response:	threshold value (**int32**) in mdB
example:	`prog:expl:numb? refl` → 60000<END>
affects:	All instruments

command:	**PROGram:EXPLicit:STATe**
syntax:	PROGram:EXPLicit:STATe<wsp>"scan",<boolean>
description:	Allows terminating the currently running task
parameters:	A **boolean** value: 0 – terminate the task 1 – no action
response:	none
example:	`prog:expl:stat "scan",0`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**PROGram:EXPLicit:STATe?**
syntax:	PROGram:EXPLicit:STATe?<wsp>"scan"
description:	Queries whether a task is still running.
parameters:	none
response:	A **boolean** value: 0 – task is not running 1 – task is still running
example:	`prog:expl:stat? "scan"` → 1<END>
affects:	All instruments

command:	**CALCulate:MATH:EXPRession:NAME?**
syntax:	CALCulate:MATH:EXPRession:NAME?<wsp><expr>
description:	Allows calculating several loss and attenuation values. All calculations use the stretch between markers A and B.
parameters:	Valid values are: LOSS LSAattenuation ATTenuation. ORL: Optical Return Loss
response:	The loss is returned in dB. The attenuations are returned in mdB/km.
example:	`calc:math:expr:name? att` → 291MDB/KM<END>
affects:	All instruments

command:	**CALCulate:MATH:EXPRession:REFLex?**
syntax:	CALCulate:MATH:EXPRessionREFLex?<wsp><pos1>,<pos2>, <pos3>
description:	Calculate the Reflectance of an event

NOTE **The active marker must be at the position of the Event.**

parameters:	3 aux marker positions with length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.
response:	reflectance or reflection height in dB

NOTE **The type of measurement given (reflectance or reflection height) depends on how you have configured your instrument.**
You specify a new configuration with `calc:math:expr:type`.

example:	`calc:math:expr:refl? 9.5km,9800m,1001000cm` → -55.5000DB (*Marker at 10km*).
affects:	All instruments

command:	**CALCulate:MATH:EXPRession:SPLice?**
syntax:	CALCulate:MATH:EXPRession:SPLice?<wsp><pos1>,<pos2>, <pos3>,<pos4>
description:	Calculate the splice loss of an event.

NOTE **The active marker must be at the position of the splice.**

parameters:	4 aux marker positions with length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.
response:	splice loss in mdB
example:	`calc:math:expr:spl? 9.5km,9800m,10500m,10.8km` → 100MDB (*Marker at 10km*).
affects:	All instruments

command:	**CALCulate:MATH:EXPRession:TYPE**
syntax:	CALCulate:MATH:EXPRession:TYPE<wsp><type>
description:	Sets the reflection parameter used for the return value of `calc:math:expr:refl?` and the event table (for example, `trac:data:tabl`).
parameters:	Valid values are: REFLectance and HEIGht.
response:	none
example:	`calc:math:expr:type refl`
affects:	Mini-OTDR and Rack OTDR

command:	**CALCulate:MATH:EXPRession:TYPE?**
syntax:	CALCulate:MATH:EXPRession:TYPE?
description:	Queries the reflection parameter used for the return value of `calc:math:expr:refl?` and the event table (for example, `trac:data:tabl`).
parameters:	none
response:	REFL or HEIG
example:	`calc:math:expr:type` → REFL<END>
affects:	Mini-OTDR and Rack OTDR

4.3 Measurement Functions – The SENSe Subsystem

The SENSe subsystem lets you control measurement parameters like the averaging time, the detector's bandwidth, and fiber parameters.

command:	**SENSe:AVERage:COUNt**
syntax:	SENSe:AVERage:COUNt<wsp><value>
description:	Sets the averaging time.
parameters:	Averaging time in seconds (a **short** value). A value of 0 means that the measurement runs until it is stopped by the user.
response:	none
example:	`sens:aver:coun 180`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**SENSe:AVERage:COUNt?**
syntax:	SENSe:AVERage:COUNt?<wsp><boolean>
description:	Queries the averaging time.
parameters:	A **boolean** value: 0 – returns averaging time; 1 – returns time elapsed since start of measurement.
response:	Averaging time in seconds (a **short** value).

NOTE **If your instrument is configured to measure Number of Averages, rather than Averaging Time, you receive a response of 0. Use `sens:aver:coun` to configure your instrument for Averaging Time (Mini-OTDR only).**

example:	`sens:aver:coun? 0` → +180<END>
affects:	All instruments

command:	**SENSe:AVERage:COUNt:NUMBer**
syntax:	SENSe:AVERage:COUNt:NUMBer<wsp><value>
description:	Sets the number of averages to measure.
parameters:	Number of averages as a power of 2 (a **short** value). For example, if you enter 14, 2^{14} averages are taken. A value of 0 means that the measurement runs until it is stopped by the user.

NOTE **You may only enter 0 or an integer between 14 and 22.**

response:	none
example:	`sens:aver:coun:numb 14`
affects:	Mini-OTDR and Rack OTDR

command:	**SENSe:AVERage:COUNt:NUMBer?**
syntax:	SENSe:AVERage:COUNt?<wsp><boolean>
description:	Queries the number of averages measured.
parameters:	A **boolean** value: 0 – returns averaging time; 1 – returns time elapsed since start of measurement.
response:	Number of averages as a power of 2 (a **short** value). For example, if you see 14, the instrument is configured to take 2^{14} averages.

NOTE **If your instrument is configured to measure Averaging Time, rather than Number of Averages, you receive a response of 0. Use `sens:aver:coun:numb` to configure your instrument for Number of Averages.**

example:	`sens:aver:coun? 0` → 14<END>
affects:	Mini-OTDR and Rack OTDR

command:	**SENSe:DETector[:FUNCtion]**
syntax:	SENSe:DETector[:FUNCtion]<wsp><mode>
description:	Sets the current measurement mode.
parameters:	Valid modes are: AVERage REAL time (not Mini-FBL) CONTinue (not Mini-FBL) CW RETLoss (Mainframe OTDR only) M2kHz (not Mainframe OTDR)
response:	none
example:	`sens:det` aver
affects:	All instruments

command:	**SENSe:DETector[:FUNCtion]?**
syntax:	SENSe:DETector[:FUNCtion]?
description:	Returns the current measurement mode.
parameters:	none
response:	Possible responses are: AVERage REAL time (not Mini-FBL) CONTinue (not Mini-FBL) CW RETLoss (Mainframe OTDR only) M2kHz (not Mainframe OTDR)
example:	`sens:det?` $\rightarrow$ AVER<END>
affects:	All instruments

command:	**SENSe:DETector[:FUNCtion:]AUTO**
syntax:	SENSe:DETector[:FUNCtion]:AUTO<wsp><boolean>
description:	Enables or disables the automatic measurement mode.
parameters:	A **boolean** value: 0 – disable auto mode 1 – enable auto mode
response:	none
example:	sens:det:auto 1
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**SENSe:DETector[:FUNCtion:]AUTO?**
syntax:	SENSe:DETector[:FUNCtion]:AUTO?
description:	Queries whether the automatic measurement mode is enabled.
parameters:	none
response:	A **boolean** value: 0 – auto mode disabled 1 – auto mode enabled
example:	`sens:det:auto?` → 1<END>
affects:	All instruments

command:	**SENSe:DETector[:FUNCtion:]OPTimize**
syntax:	SENSe:DETector[:FUNCtion]:OPTimize<wsp><mode>
description:	Sets the optimization mode
parameters:	Valid modes are: NONE – standard optimization RESolution – optimize for resolution DYNamic – optimize for dynamic LINearity - optimize for linearity (Mainframe OTDR only)
response:	none
example:	sens:det:opt res
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**SENSe:DETector[:FUNCtion:]OPTimize?**
syntax:	SENSe:DETector[:FUNCtion]:OPTimize?
description:	Returns the current optimization mode.
parameters:	none
response:	Possible modes are NONE – standard optimization RESolution – optimize for resolution DYNamic – optimize for dynamic LINearity - optimize for linearity (Mainframe OTDR only)
example:	sens:det:opt? → RES<END>
affects:	All instruments

command:	**SENSe:DETector:MODE**
syntax:	SENSe:DETector:MODE<wsp><mode>
description:	Selects the mode of the OTDR screen
parameters:	Valid modes are: OTDR – OTDR mode BREAK – Fiber Break Locator SOURce – Source mode
response:	none
example:	`sens:det:mode otdr`
affects:	Mini-OTDR and Rack OTDR

command:	**SENSe:DETEctor:MODE?**
syntax:	SENSe:DETector:MODE?
description:	Returns the current mode of the OTDR
parameters:	none
response:	Possible modes are OTDR, BREAK, SOUR
example:	`sens:det:mode` → OTDR<END>
affects:	Mini-OTDR, Mainframe OTDR, and Rack OTDR

command:	**SENSe:DETector:SAMPle:DISTance?**
syntax:	SENSe:DETector:SAMPle:DISTance?
description:	Queries the current sample distance.
parameters:	none
response:	The sample distance in mm.
example:	`sens:samp:dist?` → +4600<END>
affects:	All instruments

command:	**SENSe:FIBer:REFRindex**
syntax:	SENSe:FIBer:REFRindex<wsp><value>
description:	Sets the fiber's refractive index.
parameters:	The refractive index (a **float** value).
response:	none
example:	`sens:fib:refr 1.458`
affects:	All instruments

command:	**SENSe:FIBer:REFRindex?**
syntax:	SENSe:FIBer:REFRindex?
description:	Returns the current refractive index.
parameters:	none
response:	The refractive index (a **float** value).
example:	`sens:fib:refr?` →+1.4580000<END>
affects:	All instruments

command:	**SENSe:FIBer:SCATtercoeff**
syntax:	SENSe:FIBer:SCATtercoeff<wsp><value>[dB|mdB]
description:	Sets the fiber’s scatter coefficient.
parameters:	The scatter coefficient in mdB (default) or dB (a **float** value).
response:	none
example:	`sens:fib:scat 51500mdb`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**SENSe:FIBer:SCATtercoeff?**
syntax:	SENSe:FIBer:SCATtercoeff?
description:	Returns the current scatter coefficient.
parameters:	none
response:	The scatter coefficient in dB (a **float** value).
example:	`sens:fib:scat?` $\rightarrow$ +51.500DB<END>
affects:	All instruments

command:	**SENSe:FIBer:TYPE?**
syntax:	SENSe:FIBer:TYPE?
description:	Queries the fiber type of the measurement module.
parameters:	none
response:	Possible values are: MONOmode MULTimode
example:	`sens:fib:type?` $\rightarrow$ MULT<END>
affects:	All instruments

command:	**SENSe:POWer:FREQuency?**
syntax:	SENSe:POWer:FREQuency?
description:	Queries the detected power meter input frequency.
parameters:	none
response:	Valid responses are: CW, LI, and the current frequency in Hz or KHz
example:	`sens:pow:freq?` → 270HZ<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:REFerence**
syntax:	SENSe:POWer:REFerence<wsp><value> [pW|nW|uW|mW|Watt|dBm]
description:	Sets the power meter reference value
parameters:	The reference as a **float** value. You may append a unit type. Valid units are: pW, nW, uW, mW, Watt, and dBm. If no unit type is specified, dBm is implied.
response:	none
example:	`sens:pow:ref 4dBm`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:REFerence?**
syntax:	SENSe:POWer:REFerence?
description:	Queries the power meter reference value and units
parameters:	none
response:	The reference as a **float** value in dBm, W or dB.

NOTE **If the reference state is relative, units are dBm or W.**
If the reference state is absolute, units are dB

example:	`sens:pow:ref?` → +4DBM<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:REFerence:DISPlay**
syntax:	SENSe:POWer:REFerence:DISPlay
description:	Takes the current power meter value as the reference value
parameters:	none
response:	none
example:	`sens:pow:ref:disp`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:REFerence:STATe**
syntax:	SENSe:POWer:REFerence:STATe<wsp><boolean>
description:	Sets the power meter display to relative or absolute
parameters:	A **boolean** value: 0 – relative 1 - absolute
response:	none
example:	`sens:pow:ref:stat 1`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:REFerence:STATe?**
syntax:	SENSe:POWer:REFerence:STATe?
description:	Inquires whether the current power meter display is relative or absolute
parameters:	none
response:	A **boolean** value: 0 – relative 1 - absolute
example:	`sens:pow:ref:stat?` → 1<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:UNIT**
syntax:	SENSe:POWer:UNIT<wsp><boolean>
description:	Sets the power meter power unit
parameters:	A **boolean** value: 0 – dBm 1 - Watt or DBM or W
response:	none
example:	`sens:pow:unit 1`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSe:POWer:UNIT?**
syntax:	SENSe:POWer:UNIT?
description:	Inquires the current power meter power unit
parameters:	none
response:	DBM or W
example:	`sens:pow:unit?` → W<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSE:POWer:WAVelength**
syntax:	SENSE:POWer:WAVelength<wsp><value>[NM | UM | MM | M]
description:	Sets the current power meter wavelength.
parameters:	The wavelength as a **float** value in nm/um/mm/m.
response:	none
example:	`sens:pow:wav 1550E-3um`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**SENSE:POWer:WAVelength?**
syntax:	SENSE:POWer:WAVelength?
description:	Inquires the current power meter wavelength.
parameters:	none
response:	The wavelength as a **float** value in nm.
example	`sens:pow:wav?` → +1550NM<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

4.4 Signal Generation – The SOURce Subsystem

The SOURce subsystem allows controlling the OTDR's optical source. It also controls positions and appearance of the markers

command:	**[SOURce:]AM[:INTernal]:FREQuency[1]**
syntax:	[SOURce:]AM[:INTernal]:FREQency[1]<wsp><freq>
description:	Sets the modulation frequency of the internal source
parameters:	Valid units are: CW, F270HZ, F1KHZ, F2KHZ, and CODE
response:	none
example:	`am:freq f270hz`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**[SOURce:]AM[:INTernal]:FREQuency[1]?**
syntax:	[SOURce:]AM[:INTernal]:FREQency[1]?
description:	Queries the current modulation frequency of the internal source
parameters:	none
response:	Valid units are: CW, F270HZ, F1KHZ, F2KHZ, and CODE
example:	`am:freq?` → F270HZ<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**[SOURce:]AM[:INTernal]:FREQuency2**
syntax:	[SOURce:]AM[:INTernal]:FREQency2<wsp><freq>
description:	Sets the modulation frequency of the Visual Fault Finder
parameters:	Valid units are: CW and F1HZ
response:	none
example:	`am:freq2 f1hz`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**[SOURce:]AM[:INTernal]:FREQuency2?**
syntax:	[SOURce:]AM[:INTernal]:FREQuency2?
description:	Queries the current modulation frequency of the Visual Fault Finder
parameters:	none
response:	Valid units are: CW and F1HZ
example:	`am:freq2?` → F1HZ<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

.

command:	**[SOURce:]HOFFset**
syntax:	[SOURce:]HOFFset<wsp><value>[MM | CM | M | KM | MI | FT | KFT]
description:	Sets the horizontal offset.
parameters:	The offset as a **float** value. You may append a length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.

NOTE **A value of 0 clears the horizontal offset.**

response:	none
example:	`hoff 5km`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]HOFFset?**
syntax:	[SOURce:]HOFFset?
description:	Returns the current horizontal offset.
parameters:	none
response:	The offset as a **float** value in the current length unit.
example:	`hoff?` → +5.0000000KM<END>
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]MARKer1|2|3:POINt**
syntax:	[SOURce:]MARKer1|2|3:POINt<wsp><position>[length unit]
description:	Sets the position of the selected marker (MARK1 = marker A, MARK2 = marker B, MARK3 = marker C).

NOTE **The Mini-OTDR and Rack OTDR have no Marker C. MARK3 is therefore only valid for the Mainframe OTDR.**

parameters:	Position in length unit.
response:	none
example:	`mark2:poin 1000m`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]MARKer1|2|3:POINt?**
syntax:	[SOURce:]MARKer1|2| 3:POINt?
description:	Returns the position of the selected marker (MARK1 = marker A, MARK2 = marker B, MARK3 = marker C).

NOTE **The Mini-OTDR and Rack OTDR have no Marker C. MARK3 is therefore only valid for the Mainframe OTDR.**

parameters:	none
response:	Position in length unit.
example:	`mark2:poin?` → +1KM <END>
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]MARKer1|2|3 [:STATe]**
syntax:	[SOURce:]MARKer1|2|3[:STATe]<wsp><boolean>
description:	Activates or disables the selected marker (MARK1 = marker A, MARK2 = marker B, MARK3 = marker C).

NOTE **The Mini-OTDR and Rack OTDR have no Marker C. MARK3 is therefore only valid for the Mainframe OTDR.**

parameters:	A **boolean** value:	0 – disables marker
		1 – enables marker
response:	none	
example:	`mark2 1`	
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR	

command:	**[SOURce:]MARKer1|2|3[:STATe]?**
syntax:	[SOURce:]MARKer1|2|3[:STATe]?
description:	Queries the state of the selected marker (MARK1 = marker A, MARK2 = marker B, MARK3 = marker C).

NOTE **The Mini-OTDR and Rack OTDR have no Marker C. MARK3 is therefore only valid for the Mainframe OTDR.**

parameters:	none	
response:	A **boolean** value:	0 – marker disabled
		1 – marker enabled
example:	`mark2?` $\rightarrow$ 1<END>	
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR	

command:	**[SOURce:]POWer:STATe[1|2]**
syntax:	[SOURce:]POWer:STATe[1|2]
description:	Switches the laser of the chosen source on or off: `stat` or `stat1`: internal source (default) `stat2`: Visual Light Source
parameters:	A **boolean** value: 0 – Laser Off 1 - Laser On
response:	none
example:	`pow:stat 1`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**[SOURce:]POWer:STATe[1|2]?**
syntax:	[SOURce:]POWer:STATe[1|2]?
description:	Queries the laser state of the chosen source: `stat` or `stat1`: internal source (default) `stat2`: Visual Light Source
parameters:	none
response:	A **boolean** value: 0 – Laser Off 1 - Laser On
example:	`pow:stat` → 1<END>
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**[SOURce:]PULSe:WIDTh**
syntax:	[SOURce:]PULSe:WIDTh<wsp><value>[NS|US|MS|S]
description:	Sets the measurement pulsewidth.
parameters:	The pulsewidth in ns/us (a **float** value).
response:	none
example:	puls:widt `3000E-9s`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]PULSe:WIDTh?**
syntax:	[SOURce:]PULSe:WIDTh?
description:	Returns the measurement pulsewidth.
parameters:	none
response:	The pulsewidth in ns/us (a **short** value).
example:	`puls:widt?` → 3US<END>
affects:	All instruments

command:	**[SOURce:]PULSe:WIDTh:LLIMit?**
syntax:	[SOURce:]PULSe:WIDTh:LLIMit?
description:	Returns the lower limit for the pulsewidth determined by the measurement hardware.
parameters:	none
response:	The pulsewidth in ns/us (a **short** value).
example:	`puls:widt:llim?` → +10NS<END>
affects:	All instruments

command:	**[SOURce:]PULSe:WIDTh:ULIMit?**
syntax:	[SOURce:]PULSe:WIDTh:ULIMit?
description:	Returns the upper limit for the pulsewidth determined by the measurement hardware.
parameters:	none
response:	The pulsewidth in ns/us (a s**hort** value).
example:	`puls:widt:ulim?` →+10US<END>
affects:	All instruments

command:	**[SOURce:]RANGe:LUNit**	
syntax:	[SOURce:]RANGe:LUNit<wsp><unit>	
description:	Sets the length unit.	
parameters:	Valid units are:	M – meters
		FT – feet
		MI – miles
response:	none	
example:	`rang:lun m`	
affects:	All instruments	

command:	**[SOURce:]RANGe:LUNit?**	
syntax:	[SOURce:]RANGe:LUNit?	
description:	Queries the current length unit.	
parameters:	none	
response:	Valid units are:	M – meters
		FT – feet
		MI – miles
example:	`rang:lun?` $\rightarrow$ M<END>	
affects:	All instruments	

command:	**[SOURce:]RANGe:SPAN**
syntax:	[SOURce:]RANGe:SPAN<wsp><value>[MM | CM | M | KM | MI | FT | KFT]
description:	Sets the measurement span.
parameters:	The span as a **float** value. You may append a length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.
response:	none
example:	`rang:span 50mi`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]RANGe:SPAN?**
syntax:	[SOURce:]RANGe:SPAN?
description:	Returns the current measurement span.
parameters:	none
response:	The span as a **float** value in the current length unit.
example:	`rang:span?` → +80.4670000KM<END>
affects:	All instruments

command:	**[SOURce:]RANGe:STARt**
syntax:	[SOURce:]RANGe:STARt<wsp><value>[MM | CM | M | KM | MI | FT | KFT]
description:	Sets the starting point for the measurement.
parameters:	The start as a **float** value. You may append a length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.
response:	none
example:	`rang:star 10km`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**[SOURce:]RANGe:STARt?**
syntax:	[SOURce:]RANGe:STARt?
description:	Returns the current starting point for the measurement.
parameters:	none
response:	The start as a **float** value in the current length unit.
example:	`rang:star?` → 10.0000000KM<END>
affects:	All instruments

command:	**[SOURce:]WAVelength[1|2][:CW]**
syntax:	[SOURce:]WAVelength[1|2][:CW]<wsp><value>[NM | UM | MM | M]
description:	Sets the wavelength for the specified source: `wav` or `wav1`: internal source (default) `wav2`: Visual Light source

NOTE **`wav2` is only included for the sake of consistency. You will never want to set the Visual Light Source wavelength**

NOTE **You cannot use a submodule with an Mainframe OTDR. This means that `wav2` is unavailable.**

parameters:	The wavelength as a **float** value in nm/um/mm/m.
response:	none
example:	`wav 1550E-3um`
affects:	All instruments

command:	**[SOURce:]WAVelength[1|2][:CW]?**
syntax:	[SOURce:]WAVelength[1:2][:CW]?
description:	Inquires the wavelength for the specified source: WAVelength or WAVelength1: internal source (default) WAVelength2: Visual Light source

NOTE **You cannot use a submodule with an Mainframe OTDR. This means that `wav2` is unavailable.**

parameters:	none
response:	The wavelength as a **float** value in nm.
example:	`wav?` → +1550NM<END>
affects:	All instruments

command:	**[SOURce:]WAVelength[1|2][:CW]:AVAilable?**
syntax:	[SOURce:]WAVelength[1|2][:CW]:AVAilable?
description:	Returns the wavelengths for the specified source: `wav` or `wav1`: internal source (default) `wav2`: Visual Light source

NOTE **You cannot use a submodule with an Mainframe OTDR. This means that `wav2:ava?` is unavailable.**

parameters:	The wavelengths as **float** values separated by commas.
response:	none
example:	`wav:ava?` → 1310,1550<END>
affects:	All instruments

4.5 Trace Data Access – The TRACe Subsystem

The TRACe subsystem lets you control the traces loaded into the OTDR’s memory.

command:	**TRACe:CATalog?**
syntax:	TRACe:CATalog?
description:	Returns the names of the currently loaded traces and their positions in the trace array. There is a maximum of two loaded traces for the Mini-OTDR and Rack OTDR, and four loaded traces for the Mainframe OTDR.
parameters:	none
response:	A string terminated by <END>.
example:	`trac:cat?` → "1:TRACE1.SOR 2:TRACE2.SOR"<END>
affects:	All instruments

command:	**TRACe:DATA?**
syntax:	TRACe:DATA?
description:	Reads a complete trace data array for the current trace.
parameters:	none
response:	The data is a Binary Block containing the trace data.

NOTE

`TRAC:DATA?` returns blocks of unsigned short (16-bit) data in Intel little endian byte ordering (low byte first).

Some processor architectures (such as HP PA-Risc or Motorola) use big endian byte order (high byte first).

If your processor uses big endian byte order, you must swap the low and high byte for each 16 bit value.

If you are not sure about the byte ordering technique used by your processor, please consult your processor documentation.

example:	`trac:data?` → #48192[..8192 bytes of data..]<END>
affects:	All instruments

command:	**TRACe:DATA:CHECk:TABLe?**
syntax:	TRACe:DATA:CHECk:TABLe?
description:	Returns the Pass/Fail test Checker Table.
parameters:	none.
response:	Block containing the Pass/Fail test table. The header is the same as a binary, but the data is in ASCII format.
example:	`trac:data:chec:tabl?` → *block*<END>
affects:	Mini-OTDR and Rack OTDR

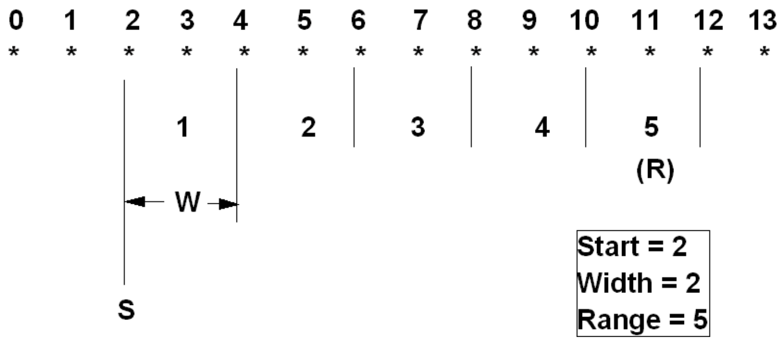
command:	**TRACe:DATA:CHECk:STATe?**
syntax:	TRACe:DATA:CHECk:STATe?
description:	Returns the current Pass/Fail test state.
parameters:	none
response:	Possible values are: INVALID, PASSED, FAILED
example:	`trac:data:chec:stat?` → PASSED<END>
affects:	Mini-OTDR and Rack OTDR

command:	**TRACe:DATA:FCRetloss?**
syntax:	TRACe:DATA:FCRetloss?
description:	Returns the Front connector Return Loss
parameters:	none
response:	Return loss in dB.
example:	`trac:data:fcr?` → -35723MDB<END>
affects:	All instruments

command:	**TRACe:DATA:LANDmark:ADD**	
syntax:	TRACe:DATA:LANDmark:ADD<wsp><value>[MM | CM | M | KM | MI | FT | KFT],<comm>	
description:	Adds a landmark.	
parameters:	<value>	The landmark position as a **float** value. You may append a length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.
	<comm>	Landmark name, given as a string in " " (max. 40 characters)
response:	none	
example:	`trac:data:land:add 2km,"Landmark A"`	
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR	

command:	**TRACe:DATA:LANDmark:DELete**
syntax:	TRACe:DATA:LANDmark:DELete<wsp><value>[MM | CM | M | KM | MI | FT | KFT]
description:	Deletes a landmark.
parameters:	The landmark position as a **float** value. You may append a length unit. Valid length units are: MM, CM, M, KM, MI, FT, KFT.
response:	none
example:	`trac:data:land:del 2km`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**TRACe:DATA:LINE?**
syntax:	TRACe:DATA:LINE?<wsp><start>,<range>,<width>,<minmax>
description:	Starting at sample `start`, examines the next `width` samples, and notes their minimum/maximum value. (`minmax` determines whether it is `MIN` or `MAX`). Repeats this for `range` samples, and stores the resulting line in a binary block.
parameters:	`start` (**int32**) - starting point from which samples are taken. `range` (**int32**) - number of separate samples analyzed, `width` (**int32**) - number of points in each sample. For an illustration of the interpretation of the parameters, see the diagram below:



NOTE

`start` + (`range*width`) must be less than the number of data points

`range` must be greater than or equal to 4

`width` must be greater than 0

	`minmax` - `MIN`: minimum value is taken; `MAX`: maximum value is taken
response:	binary block

NOTE	**`TRAC:DATA:LINE?` returns blocks of unsigned short (16-bit) data in Intel little endian byte ordering (low byte first).** **Some processor architectures (such as HP PA-Risc or Motorola) use bug endian byte order (high byte first).** **If your processor uses big endian byte order, you must swap the low and high byte for each 16 bit value.** **If you are not sure about the byte ordering technique used by your processor, please consult your processor documentation.**

example:	`trac:data:line? 2,5,2,MAX`→ *block*
affects:	All instruments

command:	**TRACe:DATA:TABLe?**
syntax:	TRACe:DATA:TABLe?
description:	Returns an event table.
parameters:	none.
response:	Block containing the event table. The header is the same as a binary, but the data is in ASCII format.
example:	`trac:data:tabl?` → *block*
affects:	All instruments

command:	**TRACe:DATA:TABLe:LOCK**
syntax:	TRACe:DATA:TABLe:LOCK<wsp><boolean>
description:	Locks/Unlocks the event table
parameters:	A **boolean** value: 0: table unlocked 1: table locked
response:	none
example:	`trac:data:tabl:lock 0`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**TRACe:DATA:TABLe:LOCK?**
syntax:	TRACe:DATA:TABLe:LOCK?
description:	Returns whether the event table is locked.
parameters:	none.
response:	A **boolean** value: 0: table unlocked 1: table locked
example:	`trac:data:tabl:lock?` → 0<END>
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**TRACe:DATA:TORL?**
syntax:	TRACe:DATA:TORL?
description:	Returns the Total Optical Return Loss
parameters:	none
response:	Return loss in dB.
example:	`trac:data:torl?` → +35.7DB<END>
affects:	All instruments

command:	**TRACe:DATA:VALue?**
syntax:	TRACe:DATA:VALue?<wsp><sample point>
description:	Returns the measured value at the specified sample point.

NOTE **The maximum value of <sample point> is determined by `trac:poin?`**

parameters:	The sample point.
response:	The measured value in mdB.
example:	`trac:data:val? 1999` → +31800<END>
affects:	All instruments

command:	**TRACe:DELete**
syntax:	TRACe:DELete
description:	Closes the current trace.
parameters:	none
response:	none
example:	`trac:del`
affects:	All instruments

command:	**TRACe:DELete:ALL**
syntax:	TRACe:DELete:ALL
description:	Closes all loaded traces.
parameters:	none
response:	none
example:	`trac:del:all`
affects:	All instruments

command:	**TRACe:FEED:CONTrol**
syntax:	TRACe:FEED:CONTrol<wsp><trace>
description:	Specifies the current trace.

NOTE **The current trace receives all measurement data and therefore will be overwritten with every new measurement**

parameters:	Valid values are:	FIRSt
		SECond
		THIRd (Mainframe OTDR only)
		FOURth (Mainframe OTDR only)
response:	none	
example:	`trac:feed:cont sec`	
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR	

command:	**TRACe:FEED:CONTrol?**
syntax:	TRACe:FEED:CONTrol?
description:	Returns the current trace.
parameters:	none
response:	Possible values are: FIRSt SECond THIRd (Mainframe OTDR only) FOURth (Mainframe OTDR only)
example:	`trac:feed:cont?` → SEC<END>
affects:	All instruments

command:	**TRACe:FREE?**
syntax:	TRACe:FREE?
description:	Returns the number of unused trace array fields.
parameters:	none
response:	A **short** value: 0...2.(0...4 for the Mainframe OTDR)
example:	`trac:free?` → `+2<END>`
affects:	All instruments

command:	**TRACe:POINts**
syntax:	TRACe:POINts
description:	Sets the maximum number of samples for the current trace.
parameters:	The number of data points (a **short** value). Valid arguments are 4000, 8000, and 16000.
response:	none
example:	`trac:poin 8000`
affects:	Mini-OTDR and Rack OTDR

command:	**TRACe:POINts?**
syntax:	TRACe:POINts?
description:	Returns the number of trace data points for the current trace.
parameters:	none
response:	The number of data points (a **short** value).
example:	`trac:poin?` $\rightarrow$ +8192<END>
affects:	All instruments

5

Mass Storage, Display, and Print Functions

Mass Storage, Display, and Print Functions

This chapter gives descriptions of commands that you can use when storing and printing traces from your OTDR. The commands are split into the following separate subsystems:

- **:DISPLAY**: commands which relate to what you see on the OTDR display.
- **:HCOPY**: commands which relate to printing operations.
- **:MMEMORY**: commands which relate to the OTDR memory.

Other commands are described in Chapter 3 “Instrument Setup and Status”, and Chapter 4 “Operations on Traces and Measurements”.

5.1 Display Operations – The DISPlay Subsystem

The DISPlay subsystem lets you control what you see on the OTDR's display.

command:	**DISPlay:BRIGhtness**
syntax:	DISPlay:BRIGhtness<wsp><value>
description:	Controls the brightness for the display.
parameters:	0 .. 100 (0 ..64 on the Mainframe OTDR)
response:	none
example:	`disp:brig 32`
affects:	All instruments

command:	**DISPlay:BRIGhtness?**
syntax:	DISPlay:BRIGhtness?
description:	Requests the brightness for the display.
parameters:	none
response:	0 .. 100 (0 ..64 on the Mainframe OTDR)
example:	`disp:brig?` → 32<END>
affects:	All instruments

command:	**DISPlay:CONTrast**
syntax:	DISPlay:CONTrast<wsp><value>
description:	Controls the contrast for the display.
parameters:	0 .. 100
response:	none
example:	`disp:cont 50`
affects:	Mini-OTDR and Mini-FBL

command:	**DISPlay:CONTrast?**
syntax:	DISPlay:CONTrast?
description:	Requests the contrast for the display.
parameters:	none
response:	0 .. 100
example:	`disp:cont?` → 50<END>
affects:	Mini-OTDR and Mini-FBL

command:	**DISPlay:ENABle**
syntax:	DISPlay:ENABle<wsp><boolean>
description:	Enables or disables the LCD.
parameters:	A **boolean** value: 0 – switch off the LCD 1 – switch on the LCD
response:	none
example:	`disp:enab 1`
affects:	All instruments

command:	**DISPlay:ENABle?**
syntax:	DISPlay:ENABle?
description:	Queries the state of the LCD.
parameters:	none
response:	A b**oolean** value: 0 – the LCD is turned off 1 – the LCD is turned on
example:	`disp:enab?` → 1<END>
affects:	All instruments

command:	**DISPlay[:WINDow]:GRAPhics:COLor**
syntax:	DISPlay[:WINDow]:GRAPhics:COLor<wsp><color>
description:	Changes the color of the current trace.
parameters:	The new trace color (a **short** value): BLACk, RED, BLUE, GREen, GREY, WHITe
response:	none
example:	`disp:grap:col blac`
affects:	Mainframe OTDR only

command:	**DISPlay[:WINDow]:GRAPhics:COLor?**
syntax:	DISPlay[:WINDow]:GRAPhics:COLor?
description:	Queries the color of the current trace.
parameters:	none
response:	The current trace color (a **short** value): BLAC, RED, BLUE, GRE, GREY, WHIT
example:	**DISPlay[:WINDow]:GRAPhics:COLor?**
affects:	Mainframe OTDR only

command:	**DISPlay[:WINDow]:GRAPhics:LTYPe**
syntax:	DISPlay[:WINDow]:GRAPhics:LTYPe<wsp><boolean>
description:	Changes the linestyle of the current trace.
parameters:	A **boolean** value: 0 – new linestyle is dotted 1 – new linestyle is solid
response:	none
example:	`disp:grap:ltyp 0`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**DISPlay[:WINDow]:GRAPhics:LTYPe?**
syntax:	DISPlay[:WINDow]:GRAPhics:LTYPe?
description:	Queries the linestyle of the current trace.
parameters:	none
response:	A **boolean** value: 0 – current linestyle is dotted
	1 – current linestyle is solid
example:	`disp:grap:ltyp?` → 0<END>
affects:	All instruments

command:	**DISPlay[:WINDow]:TEXT:DATA**
syntax:	DISPlay[:WINDow]:TEXT:DATA<wsp><c-no>,<comm>
description:	Sets a comment in the trace.
parameters:	<c-no> 0 .. 4 - comment number
	<comm> Comment, given as a string in " " (max. 40 characters)
response:	none
example:	`disp:text:data 0,"This is a Comment"`
affects:	All instruments

command:	**DISPlay[:WINDow]:TEXT:DATA?**
syntax:	DISPlay[:WINDow]:TEXT:DATA? <wsp><c-no>
description:	Requests an individual comment
parameters:	0 .. 4 - comment number
response:	Comment, given as a string, terminated by <END>
example:	`disp:text:data? 0` → "This is a Comment"<END>
affects:	All instruments

command:	**DISPlay[:WINDow]:X:SCALe**
syntax:	DISPlay[:WINDow]:X:SCALe:<wsp><type>
description:	Controls whether the display is in full trace mode or zoomed.

NOTE **You must send this command before you perform any zooming operations.**

The `DISP ... :PDIV/?` commands described below only work in AROund mode.

parameters:	FULLtrace or AROund.
response:	none
example:	`disp:x:scal full`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**DISPlay[:WINDow]:X:SCALe?**
syntax:	DISPlay[:WINDow]:X:SCALe?
description:	Queries whether the display is in full trace mode or zoomed.
parameters:	none
response:	FULLtrace or AROund
example:	`disp:x:scal?` $\rightarrow$ FULL<END>
affects:	All instruments

command:	**DISPlay[:WINDow]:X[:SCALe]:PDIVision**
syntax:	DISPlay[:WINDow]:X[:SCALe]:PDIVision<wsp><value>
description:	Determines the scaling of the X-axis.

NOTE **This command only works in AROund mode (see `DISP:X:SCAL`).**

parameters:	Valid values for the scaling: 0...15 (a **short** value): 0 – full trace ... 15– 1 m/DIV
response:	none
example:	`disp:x:pdiv 3`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**DISPlay[:WINDow]:X[:SCALe]:PDIVision?**
syntax:	DISPlay[:WINDow]:X[:SCALe]:PDIVision?
description:	Queries the current scaling of the X-axis.

NOTE **This command only works in AROund mode (see `DISP:X:SCAL`).**

parameters:	none
response:	Possible values for the scaling: 0...15 (a **short** value): 0 – full trace .... 15 – 1 m/DIV
example:	`disp:x:pdiv?` → +3<END>
affects:	All instruments

command:	**DISPlay[:WINDow]:Y[:SCALe]:PDIVision**
syntax:	DISPlay[:WINDow]:Y[:SCALe]:PDIVision<wsp><value>
description:	Determines the scaling of the Y-axis.

NOTE **This command only works in AROund mode (see `DISP:X:SCAL`).**

parameters:	Valid values for the scaling: 1...7 (a **short** value): 1 –> 5 dB/DIV .... 7 –> 0.1 dB/DIV
response:	none
example:	`disp:y:pdiv 3`
affects:	Mini-OTDR, Rack OTDR, and Mainframe OTDR

command:	**DISPlay[:WINDow]:Y[:SCALe]:PDIVision?**
syntax:	DISPlay[:WINDow]:Y[:SCALe]:PDIVision?
description:	Queries the current scaling of the Y-axis.

NOTE **This command only works in AROund mode (see `DISP:X:SCAL`).**

parameters:	none
response:	Possible values for the scaling: 1...7(a **short** value): 1 –> 5 dB/DIV .... 7 –> 0.1 dB/DIV
example:	`disp:y:pdiv?` → +3<END>
affects:	All instruments

5.2 Print Operations – The HCOPy Subsystem

The HCOPy subsystem lets you select the print layout and control the printing.

command: **HCOPy:ABORt**
syntax: HCOPy:ABORt
description: Cancels the current print job.
parameters: none
response: none
example: `hcop:abor`
affects: All instruments

command: **HCOPy:DESTination**
syntax: HCOPy:DESTination<wsp><printer>
description: changes the current printing device.
parameters: The printer's name as a string.

Valid names for the Mini-OTDR and Rack OTDR are:

`PCL100DPI`: Standard HP-PCL printer (for example, HP LaserJet or HP DeskJet) @ 100 dots per inch
`PCL150DPI`: Standard HP-PCL printer (for example, HP LaserJet or HP DeskJet) @ 150 dots per inch
`EPSONPIN`: Epson 8-Pin printer
`SEIKODPU`: Seiko DPU-411/414

Valid name for the Mainframe OTDR are:

the name of a specific printer, for example `HP-LASERJET`
`INTernal`: internal printer
`EXTernal`: external printer

response: none
example: `hcop:dest "PCL100DPI"`
affects: All instruments

command:	**HCOPy:DESTination?**
syntax:	HCOPy:DESTination?
description:	Queries the current printing device.
parameters:	none
response:	The printer's name as a string terminated by <END>. Valid names for the Mini-OTDR and Rack OTDR are: `PCL100DPI`: Standard HP-PCL printer (for example, HP LaserJet or HP DeskJet) @ 100 dots per inch `PCL150DPI`: Standard HP-PCL printer (for example, HP LaserJet or HP DeskJet) @ 150 dots per inch `EPSONPIN`: Epson Pin printer `SEIKODPU`: Seiko DPU-411/414 `NONE`: no printer configured Valid names for the Mainframe OTDR are: *the name of a specific printer*, for example `HP-LASERJET` `INTernal`: internal printer `EXTernal`: external printer
example:	`hcop:dest?` $\rightarrow$ "PCL100DPI"<END>
affects:	All instruments

command:	**HCOPy[:IMMediate]**
syntax:	HCOPy[:IMMediate]
description:	Immediately starts printing everything that has been selected before.
parameters:	none
response:	none
example:	`hcop`
affects:	All instruments

command:	**HCOPy:ITEM:ALL[:IMMediate]**
syntax:	HCOPy:ITEM:ALL[:IMMediate]
description:	Immediately starts printing everything.
parameters:	none
response:	none
example:	`hcop:item:all`
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow][:IMMediate]**
syntax:	HCOPy:ITEM[:WINDow][:IMMediate]
description:	Immediately starts printing the parameter window.
parameters:	none
response:	none
example:	`hcop:item`
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:STATe**
syntax:	HCOPy:ITEM[:WINDow]:STATe<wsp><boolean>
description:	Enables or disables printing the parameter window.
parameters:	A **boolean** value: 0 – disable 1 – enable
response:	none
example:	`hcop:item:stat 1`
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:STATe?**
syntax:	HCOPy:ITEM[:WINDow]:STATe?
description:	Queries printing the parameter window.
parameters:	none
response:	A **boolean** value: 0 – parameter window will not be printed 1 – parameter window will be printed
example:	`hcop:item:stat?` → 1<END>
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:TEXT[:IMMediate]**
syntax:	HCOPy:ITEM[:WINDow]:TEXT[:IMMediate]
description:	Immediately starts printing the event table.
parameters:	none
response:	none
example:	`hcop:item:text`
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:TEXT:STATe**
syntax:	HCOPy:ITEM[:WINDow]:TEXT:STATe<wsp><boolean>
description:	Enables or disables printing the event table.
parameters:	A **boolean** value: 0 – disable 1 – enable
response:	none
example:	`hcop:item:text:stat 1`
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:TEXT:STATe?**	
syntax:	HCOPy:ITEM[:WINDow]:TEXT:STATe?	
description:	Queries whether the event table will be printed.	
parameters:	none	
response:	A **boolean** value:	0 – event table will not be printed
		1 – event table will be printed
example:	`hcop:item:text:stat?` → 1<END>	
affects:	All instruments	

command:	**HCOPy:ITEM[:WINDow]:TRACe[:IMMediate]**	
syntax:	HCOPy:ITEM[:WINDow]:TRACe[:IMMediate]	
description:	Immediately starts printing the trace.	
parameters:	none	
response:	none	
example:	`hcop:item:trac`	
affects:	All instruments	

command:	**HCOPy:ITEM[:WINDow]:TRACe:STATe**	
syntax:	HCOPy:ITEM[:WINDow]:TRACe:STATe<wsp><boolean>	
description:	Enables or disables printing the trace window.	
parameters:	A **boolean** value:	0 – disable
		1 – enable
response:	none	
example:	`hcop:item:trac:stat 1`	
affects:	All instruments	

command:	**HCOPy:ITEM[:WINDow]:TRACe:STATe?**
syntax:	HCOPy:ITEM[:WINDow]:TRACe:STATe?
description:	Queries whether the trace window will be printed.
parameters:	none
response:	A **boolean** value: 0 – trace window will not be printed; 1 – trace window will be printed
example:	`hcop:item:trac:stat?` → 1<END>
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:TRACe:GRATicule:STATe**
syntax:	HCOPy:ITEM[:WINDow]:TRACe:GRATicule:STATe<wsp> <boolean>
description:	Enables or disables printing the trace window grid.
parameters:	A **boolean** value: 0 – disable; 1 – enable
response:	none
example:	`hcop:item:trac:grat:stat 1`
affects:	All instruments

command:	**HCOPy:ITEM[:WINDow]:TRACe:GRATicule:STATe?**
syntax:	HCOPy:ITEM[:WINDow]:TRACe:GRATicule:STATe?
description:	Queries printing the trace window grid.
parameters:	none
response:	A **boolean** value: 0 – trace window grid will not be printed; 1 – trace window grid will be printed
example:	`hcop:item:trac:grat:stat?` → 1<END>
affects:	All instruments

command:	**HCOPy:PAGE:SIZE**
syntax:	HCOPy:PAGE:SIZE<wsp><size>
description:	Controls the paper size of the printout.
parameters:	Valid parameters are LETTer, A or A4. Please note that LETTer and A are the same page size.
response:	none
example:	`hcop:page:size A4`
affects:	All instruments

command:	**HCOPy:PAGE:SIZE?**
syntax:	HCOPy:PAGE:SIZE?
description:	Queries the current paper size of the printout.
parameters:	none
response:	A value containing A or A4, terminated by <END>
example:	`hcop:page:size?` → A4<END>
affects:	All instruments

5.3 File Operations – The MMEMory Subsystem

The MMEMory subsystem gives you access to the OTDR’s memory and to the storage devices.

command:	**MMEMory:CATalog?**
syntax:	MMEMory:CATalog?
description:	Returns the contents of the current directory.
parameters:	none
response:	A binary Block containing the contents of the directory as ASCII text, separated by CR/LF. The first digit states the number of digits following. The digits following give the total number of characters in the list of filenames.
example:	`mmem:cat?` → #229. .. DEMO1.SOR DEMO2.SOR <END>
affects:	All instruments

command:	**MMEMory:CDIRectory**
syntax:	MMEMory:CDIRectory<wsp><directory>
description:	Changes the current directory.
parameters:	The directory given as a string in " ".
response:	none
example:	`mmem:cdir "TRACES"`
affects:	All instruments

command:	**MMEMory:CDIRectory?**	
syntax:	MMEMory:CDIRectory?	
description:	Queries the current directory.	
parameters:	none	
response:	The directory given as a string terminated by <END>.	
example:	`mmem:cdir?` → "TRACES"<END>	
affects:	All instruments	

command:	**MMEMory:COPY:FILE**	
syntax:	MMEMory:COPY:FILE?<wsp><file>,<newfile>,<device>	
description:	Copies the specified Bellcore binary file from the current device.	
parameters:	The file name given as a string in " ".	
	The name of the new file given as a string in " ".	
	Device where new file is located:	FLASh - internal memory
		FLOPpy – diskette
		PCMCia - memory card
response:	none	
example:	mmem:copy:file "t0721_01.sor"`,"\abc\test.sor",flop`	
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR	

command:	**MMEMory:DELete**	
syntax:	MMEMory:DELete<wsp><file>	
description:	Deletes the specified file from the current directory.	
parameters:	The file name given as a string in " ".	
response:	none	
example:	`mmem:del "t0721_01.sor"`	
affects:	All instruments	

command:	**MMEMory:FREE**
syntax:	MMEMory:FREE
description:	Performs garbage collection on internal memory to reclaim free space.
parameters:	none
response:	none
example:	`mmem:free`
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR

command:	**MMEMory:FREE?**
syntax:	MMEMory:FREE?
description:	returns the free and used disk space.
parameters:	none
response:	<free-space> - the amount of free space
	<used-space> - the amount of used space
example:	`mmem:free?` → 125384, 1354789
affects:	All instruments

command:	**MMEMory:INITialize**	
syntax:	MMEMory:INITialize<wsp><device>	
description:	Formats the specified storage device.	
parameters:	Valid devices are:	FLASh - internal memory
		FLOPpy – diskette
		PCMCia - memory card
response:	none	
example:	`mmem:init flop`	
affects:	Mini-OTDR, Mini-FBL, and Rack OTDR	

command:	**MMEMory:LOAD:STATe, :LOAD:TRACe**
syntax:	for example: MMEM:LOAD:STATe<wsp><file>
description:	Loads a settings file or a trace file.
parameters:	The file name given as a string in " ".
response:	none
example:	`mmem:load:trac "t0721_01.sor"`
affects:	MMEM:LOAD:TRACE - All instruments
	MMEM:LOAD:STAT does not affect the Mini-FBL.

command:	**MMEMory:LOAD:FILE?**
syntax:	MMEMory:LOAD:FILE?<wsp><file>
description:	Uploads the specified Bellcore binary file from the OTDR.
parameters:	The file name given as a string in " ".
response:	binblock (Bellcore binary)
example:	`mmem:load:file? "t0721_01.sor"` → *binblock*
affects:	All instruments

command:	**MMEMory:MDIRectory**
syntax:	MMEMory:MDIRectory<wsp><directory>
description:	Creates a directory on the current storage device.
parameters:	The directory given as a string in " ".
response:	none
example:	`mmem:mdir "TRACES"`
affects:	All instruments

command:	**MMEMory:MSIS**
syntax:	MMEMory:MSIS<wsp><device>
description:	Changes the current storage device.
parameters:	Valid devices are: FLASh - internal memory (not Mainframe OTDR) FLOPpy – diskette HARDdisk (Mainframe OTDR only) PCMCia - memory card (not Mainframe OTDR)
response:	none
example:	`mmem:msis flop`
affects:	All instruments

command:	**MMEMory:MSIS?**
syntax:	MMEMory:MSIS?
description:	Queries the current storage device.
parameters:	none
response:	Possible devices are: FLAS - internal memory (not Mainframe OTDR) FLOP – diskette HARD (Mainframe OTDR only) PCMC - memory card (not Mainframe OTDR)
example:	`mmem:msis?` $\rightarrow$ FLOP<END>
affects:	All instruments

command:	**MMEMory:NAME**
syntax:	MMEMory:NAME<wsp><name>
description:	Changes the name of the current trace.
parameters:	The name given as a string.
response:	none
example:	`mmem:name "t0711_01.sor"`
affects:	All instruments

command:	**MMEMory:NAME?**
syntax:	MMEMory:NAME?
description:	Queries the name of the current trace.
parameters:	none
response:	The name given as a string.
example:	`mmem:name?` → "T0711_01.SOR"<END>
affects:	All instruments

command:	**MMEMory:SAVE:FILE**
syntax:	MMEMory:SAVE:FILE<wsp><file>,<binblock>
description:	Downloads the specified file to the OTDR.
parameters:	The file name given as a string in " ". binblock (Bellcore binary)
response:	none
example:	`mmem:save:file "t0721_01.sor",`*binblock*
affects:	All instruments

command:	**MMEMory:STORe:STATe, :STORe:TRACe**
syntax:	for example: MMEMory:STORe:STATe<wsp><file>
description:	Saves a setting or a trace under the specified name.
parameters:	The file name given as a string in " ".
response:	none
example:	`mmem:stor:trac "t0721_01.sor"`
affects:	All instruments

command:	**MMEMory:STORe:TRACe:REVision**
syntax:	MMEMory:STORe:TRACe:REVision<wsp><value>
description:	Sets the Bellcore revision number used to store Bellcore files.

NOTE **Bellcore revision 1.1 conforms to standards, but you may need to use Bellcore revision 1.0 for backward compatibility.**

parameters:	Valid values: (a **short** value): 10: Bellcore revision 1.0 11: Bellcore revision 1.1
response:	none
example:	`mmem:stor:trac:rev 11`
affects:	All instruments

command:	**MMEMory:STORe:TRACe:REVision?**
syntax:	MMEMory:STORe:TRACe:REVision?
description:	Queries the Bellcore revision number according to which Bellcore files are stored on your OTDR.
parameters:	none
response:	Possible values: (a **short** value): 10: Bellcore revision 1.0 11: Bellcore revision 1.1
example:	`mmem:stor:trac:rev?` → +11<END>
affects:	All instruments

6

Programming Examples

Programming Examples

This section contains some example programs that you can use to run an OTDR.

This programming examples do not cover the full command set for the instrument. They are intended only as an introduction to the method of programming the instrument.

We recommend that you send commands via a program, examples of which are contained in this chapter. However, for testing processes you can enter individual commands (for example, `*idn?`) from your terminal program (see “How to Send Commands and Queries” on page 152).

6.1 How to Connect your OTDR to a PC

This section explains the processes needed to connect your OTDR to a PC, and set up a serial interface,

This section contains extracts from a demo program. You can see the program in full in “SCPI data transfer between PC and OTDR” on page 159.

1 Connect the OTDR serial port to the serial interface of the PC. Use an Agilent 5180-2477 RS232 cable or an equivalent.

NOTE **For more information about attaching cables, consult the appropriate User’s Guide:**

***Mini-OTDR User’s Guide* (E6000-91031), Mini-FBL User’s Guide (E6020-91011), *OTDR User’s Guide* (E4310-91011), or *Rack OTDR User’s Guide* (E6050-91011).**

2 If you have no available cable, you can configure your own, according to the specifications listed in Table 6-1.

Table 6-1 **Cable configuration for connection to a PC**

Mini-OTDR signal	Pin	PC-Host signal	(9 pin standard) Pin
DCD	1	RTS	7
RxD	2	TxD	3
TxD	3	RxD	2
DTR	4	DSR, CTS	6, 8 (connected)
GND	5	GND	5
DSR	[6	DTR	4
RTS	7	DCD	1
CTS	8	DTR	4
RI	9	RI	9

How to set the Instrument Configuration

3 If the instrument is not also configured at your PC's serial interface, set the following configuration:

- baud rate of 19200
- hardware handshaking
- 8 data bits
- no parity
- 1 stop bit

NOTE **This is the default configuration, so you should only need to send these commands if the instrument configuration has been altered.**

```
HANDLE InitSerial( int baudrate )
    {
    static HANDLE hSer = CreateFile(
                         INTERFACE,     // use COM1 / Serial A
                         GENERIC_READ | GENERIC_WRITE,
                                        // open for read & write access
                         0, NULL,
                         OPEN_EXISTING,
                                        // well, hopefully ... :-)
                         0, NULL );

    if(!hSer)
        {
        return NULL;
        }

    // configure the interface ...
    DCB dcb;
    COMMTIMEOUTS commtimeout;
    GetCommTimeouts(hSer, &commtimeout);
    commtimeout.ReadIntervalTimeout = 3000;
    commtimeout.ReadTotalTimeoutMultiplier = 200;
    commtimeout.WriteTotalTimeoutMultiplier = 200;
    commtimeout.WriteTotalTimeoutConstant = 3000;
    GetCommState(hSer, &dcb);
    dcb.DCBlength = sizeof(dcb);
    dcb.BaudRate = baudrate;
    dcb.ByteSize = 8;
    dcb.Parity = 0;
    dcb.StopBits = 1;
    dcb.fBinary =  1;
    dcb.fParity = 0 ;
    dcb.fOutX = 0;
    dcb.fInX = 0;
    dcb.fDtrControl = DTR_CONTROL_DISABLE;

   dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;  // RTS flow control

    SetCommState(hSer, &dcb);
    SetCommTimeouts(hSer, &commtimeout);
    ClearCommBreak(hSer);
    PurgeComm(hSer,
              PURGE_TXABORT|PURGE_RXABORT|PURGE_TXCLEAR|PURGE_RXCLEAR);

    return hSer;
    }
```

Figure 6-1 **Instrument configuration - example**

6.2 How to Connect with a Terminal Program

1 Start a terminal program on the PC, for example *terminal.exe* (Win 3.11 or Windows NT), or *hypertrm.exe* (Windows 95 or Hyperterminal).

2 Set the transmission parameters in the terminal program as listed in Table 6-2:

Table 6-2 **Transmission parameters**

Speed:	19200 bps (Baud)
Code, databits:	8 bit
Communication:	Full duplex
Parity:	no parity
Startbits:	1 (not configurable)
Stopbits:	1
Flow control:	RTS-CTS (Hardware)

3 Send a test command in terminal mode to the OTDR:

type `*IDN?`

4 You should see a response, telling you the identity of your OTDR.

For example, a Mini-OTDR should respond:

```
HP E6000 Mini Optical Time Domain
Reflectometer.....
```

If you see this message, or its equivalent, the basic connection works.

5 Close the terminal program on the PC.

Closing the terminal program is important, as it avoids later conflicts with the PC and the interface control.

6.3 Using a Program to Connect to the OTDR

1. Send a new line (`"\n"`)
2. Send `*idn?` to check the identity of the OTDR
3. Check the response to the `*idn?` query.

   The response should be `HP ...<END>` and give details of the type of OTDR, and the modules used.

   The following responses are possible (depending on you OTDR type):

   - `HP E6000 Mini Optical Time Domain Reflectometer...`
   - `Agilent E6020A Fiber Break Locator Instrument...`
   - `HP E60`*`xx`*`A Rack Optical Time Domain Reflectometer...`
   - `HP 8147 Optical Time Domain Reflectometer...`
4. If you do not receive an appropriate response, repeat steps 1 to 3

until you receive the correct response or you give up.

```
// write query
sprintf (txtbuffer, "\n");
numbytes = strlen(txtbuffer);
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
sprintf(txtbuffer,"*IDN?\n");
numbytes = strlen(txtbuffer);
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);

// read response
ReadFile(hSerial, txtbuffer, MAXNUMBYTES, &cnt, 0);
if(cnt == 0 || strlen(txtbuffer) == 0)
    {
    printf("SCPI query failed, exiting!\n");
    CloseHandle(hSerial);
    return;
    }

// print result (in txtbuffer)
printf("Connected to: %s\n", txtbuffer);
```

Figure 6-2 **Connection check - example**

5 If the response is still incorrect, make the following checks:

How to check the connection

6 Send a break

This resets the instruments and RS232 to the values given in step 3.

7 Close the device and reopen it.

8 Repeat steps 1 to 4.

6.4 How to Send Commands and Queries

There are two types of SCPI commands: queries which end with a question mark (?), and commands which do not. Only queries expect a response.

Commands and queries are discussed below.

NOTE **For more information about SCPI, please consult Chapter 1 "Introduction to Programming".**

The SCPI commands specific to OTDRs are listed in Chapter 2 "Specific Commands", and explained in subsequent chapters.

Commands

Commands must be followed by a newline (`"\n"`).

For example, the abort command `abor` should be formatted as:

```
sprintf(txtbuffer,"ABOR\n");
```

There is no response.

You can check that a command has been sent correctly by sending the query `SYST:ERR?`, which returns the contents of the OTDR's error queue.

Queries

A query produces a response from the instrument.

If the response is short, you can read the line. Otherwise, you should read the response one character at a time until you find an `<END>` (see Figure 6-3).

```
sprintf(txtbuffer,"*IDN?\n");
numbytes = strlen(txtbuffer);
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
// read response
ReadFile(hSerial, txtbuffer, MAXNUMBYTES, &cnt, 0);
```

Figure 6-3 **Query - example**

Blocks transfer

Larger blocks of data are given as **Binary Blocks**, preceded by "#*HLen*Numbytes", terminated by <END>; *HLen* represents the length of the Numbytes block. For example: #16TRACES<END>.

For more examples, see Figure 6-4 and “How to Upload a Bellcore File from the current trace” on page 156

```
    // read the trace data ...
    sprintf(txtbuffer,"TRACE:DATA?\n");
    numbytes = strlen(txtbuffer);
    WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);

    // now comes the data: e.g. #48000.... which means:
    //                          | 4 digits following to tell the number
    //                                 of bytes
    //                           |||| 8000 bytes following, containing
    //                                 4000 trace pts
    cnt=0;
    while(!cnt) ReadFile(hSerial, header, 1, &cnt, 0);  // read "#"
    ReadFile(hSerial, header, 1, &cnt, 0);  // read number of digits
    header[1]=0;
    numbytes = atoi(header);
    ReadFile(hSerial, header, numbytes, &cnt, 0); // read digits
    header[cnt] = 0;
    numbytes = atoi(header);
    printf("Reading %d points of trace data ...\n", numbytes/2);
            // 1 point = 16 bit unsigned short
    ReadFile(hSerial, tracebuf, numbytes, &cnt, 0);  // read trace data
    ReadFile(hSerial, header, 15, &cnt, 0);          // read rest:
<END>\n
```

Figure 6-4 **Blocks transfer - example**

6.5 Common Tasks

This section gives some programming examples for common OTDR tasks. The examples do not cover all SCPI commands, but are just a general example.

For a full program containing some of these, and other, commands, see “SCPI data transfer between PC and OTDR” on page 159.

How to Initialize the Instrument

1 Connect to the instrument,

See “How to Connect your OTDR to a PC” on page 147.

2 Clear the error queue.
Send the command `*CLS`.

3 Check the instrument id
Send the query `*IDN?`

For example, sending: `*idn?` may return:

```
HP E6000 Mini Optical Time Domain
Reflectometer
Mainframe: 3502G00056 , Module: 3525G00056
SW-Rev.: 5.3<END>
```

How to Set Up an OTDR Measurement

4 Set up the measurement parameters.
For example, send the following commands:

```
source:puls:width 3us
source:range:start 0km
source:range:span 60km
source:wav 1310nm
sens:det:func:opt dyn
sens:aver:coun 180
sens:fib:refr 1.462
```

This sets a pulsewidth of 3 us, a start and span of 0 km - 60 km, a wavelength of 1310nm, dynamic optimize modem an averaging time of 3 minutes, and a refractive index of 1.462

5 Select the OTDR screen (this is only possible with the Mini-OTDR and Rack OTDR):
Send the command `SENS:DET:MODE OTDR`.

How to Run a Measurement

6 Start the measurement
Send the command `init`.

You can stop the measurement with the `abor` command, or wait until the Averaging Time is complete,

7 Check whether the measurement is still running `*opc?` returns `0` if the measurement is still running, `and 1` if the measurement is finished.

The measurement has now stopped, and you can check the results

How to Scan a Trace

8 Send the command `prog:expl:exec "scan"`

When the scan is complete, `*opc?` returns 1 (see note 7, above).

How to Process a Trace

9 Print the Trace

Send the command `hcop:item:all`

10 Save the Trace

Send the command `mmem:stor:trac "newtrace.sor"`

How to Upload a Bellcore File from the current trace

11 Upload the file from the OTDR

Send the query `MMEM:LOAD:FILE? ""`

12 Read in the first character

This character should be a hash (#).

13 Read in the next character

This character should be an integer, *m*, giving the number of digits you should now read.

14 Read in the next *m* characters

This series of characters should form an integer, *n*, giving the number of data bytes that follow.

15 Read in the next *n* data bytes, and store them.

16 Read until the final `<END>`.

17 Check that there have been no errors.

```
    // now comes the data: e.g. #48000.... which means:
    //                          | 4 digits following to tell the number
    //                                 of bytes
    //                           |||| 8000 bytes following, containing
    //                                 4000 trace pts
    cnt=0;
    while(!cnt) ReadFile(hSerial, header, 1, &cnt, 0);  // read "#"
    ReadFile(hSerial, header, 1, &cnt, 0);  // read number of digits
    header[1]=0;
    numbytes = atoi(header);
    ReadFile(hSerial, header, numbytes, &cnt, 0); // read digits
    header[cnt] = 0;
    numbytes = atoi(header);
    printf("Reading %d points of trace data ...\n", numbytes/2);
            // 1 point = 16 bit unsigned short
    ReadFile(hSerial, tracebuf, numbytes, &cnt, 0);  // read trace data
    ReadFile(hSerial, header, 15, &cnt, 0);          // read rest:
<END>\n

    // write the data to the console ...
    for(unsigned int i=0; i<numbytes/2; i++)
        {
        printf("idx: %d, value: %d\n", i, tracebuf[i]);
        }
```

Figure 6-5 **Uploading a Bellcore file - example**

6.6 Advanced Topics

This section gives some further examples of SCPI commands that you may wish to use when programming your OTDR.

How to Download a Bellcore File

1 Download a specified file to the OTDR

Send the command

`mmem:save:file "newtrace.sor"`*#Asss....*

Where *`#Assss...`* is a binary block containing the Bellcore file.

How to Use the Power Meter and Source Mode

These examples show you how to user the Power Meter options on the Mini-OTDR and Rack OTDR. They are not valid for the 8147A Mainframe OTDR.

1. Select source mode

   Send the command `SENS:DET:MODE SOUR`.

2. Reset the reference power

   Send the command `SENS:POW:REF 0`.

3. Set the power meter display to absolute power level readout

   Send the command `SENS:POW:REF:STAT 0`.

4. Select Watts (W) as the readout unit.

   Send the command `SENS:POW:UNIT W`

5. Start a measurement on the power meter.

   Send the command `INIT2:CONT 0`.

6. Read the detected wavelength and power.

   Send the queries `SENS:POW:WAV?` and `READ:POW?`

   These return, for example, `1310NM<END>` and `1.07898NW<END>`.

These queries respectively return the current power meter wavelength (in nm), and the current power reading (in dBm, W, or dB).

How to Store Traces on Other Devices

1. Select a new storage device.

   For example, send the command `MMEM:MSIS FLOP` to change to the floppy disk drive.

2. Check that the device has been changed correctly.

   Send the query `MMEM:MSIS?`

You should receive a string corresponding to the device that you have just set, in this case `FLOP`.

3. Check that there is enough free disk space.

Send the query MMEM:FREE?.

You receive a response giving 2 values. The first value gives the amount of free space.

4 Reclaim extra disk space, if required (this is not possible with the Mainframe OTDR).

Send the command MMEM:FREE.

NOTE **MMEM:FREE replaces internal disk space only (not, for example, for the Flash Disk or Floppy disk).**

6.7 SCPI data transfer between PC and OTDR

This C program transfers data between the Mini-OTDR and a PC.

Before you run this program connect the PC and the OTDR with an RS232 cable (see the *Mini-OTDR User's Guide*)

The program sets the measurement parameters, starts the measurement, stops the measurement 15 seconds later, and transfers the trace data to the PC.

NOTE **TRAC:DATA? and TRAC:DATA:LINE? returns blocks of unsigned short (16-bit) data in Intel little endian byte ordering (low byte first).**

Some processor architectures (such as HP PA-Risc or Motorola) use big endian byte order (high byte first).

If your processor uses big endian byte order, you must swap the low and high byte for each 16 bit value.

If you are not sure about the byte ordering technique used by your processor, please consult your processor documentation.

```
/* ---------------------------------------------------------------------------
```

```
*  Module:        demoapp.cpp                                                    *
*  Description:   application to demonstrate a SCPI data transfer between PC<->OTDR   *
*  Copyright:     12/02/1996 Hewlett-Packard GmbH                                *
*  NOTE:          This application is not supported by HP/Agilent!         *
*                 HP/Agilent cannot be held responsible                    *
*                 for any problems/damages caused by this program!          *
*                                                                                *
*  Compile:       Compile this program as a 32Bit Console Application under Win95/NT. *
*                 We recommend a struct member byte alignment of 2 bytes.           *
* -------------------------------------------------------------------------------*/

#include <windows.h>
#include <stdio.h>
#include <string.h>

#define INTERFACE "COM1"
#define MAXNUMBYTES 255
#define TRLEN 16512

HANDLE InitSerial( int baudrate )
    {
    static HANDLE hSer = CreateFile(
                         INTERFACE,                      // use COM1 / Serial A
                         GENERIC_READ | GENERIC_WRITE, // open for read & write access
                         0, NULL,
                         OPEN_EXISTING,                // well, hopefully ... :-)
                         0, NULL );

    if(!hSer)
        {
        return NULL;
        }

    // configure the interface ...
    DCB dcb;
    COMMTIMEOUTS commtimeout;
    GetCommTimeouts(hSer, &commtimeout);
    commtimeout.ReadIntervalTimeout = 3000;
    commtimeout.ReadTotalTimeoutMultiplier = 200;
    commtimeout.WriteTotalTimeoutMultiplier = 200;
    commtimeout.WriteTotalTimeoutConstant = 3000;
    GetCommState(hSer, &dcb);
    dcb.DCBlength = sizeof(dcb);
    dcb.BaudRate = baudrate;
    dcb.ByteSize = 8;
    dcb.Parity = 0;
    dcb.StopBits = 1;
    dcb.fBinary =  1;
    dcb.fParity = 0 ;
    dcb.fOutX = 0;
    dcb.fInX = 0;
    dcb.fDtrControl = DTR_CONTROL_DISABLE;
    dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;   // RTS flow control

    SetCommState(hSer, &dcb);
    SetCommTimeouts(hSer, &commtimeout);
    ClearCommBreak(hSer);
    PurgeComm(hSer, PURGE_TXABORT|PURGE_RXABORT|PURGE_TXCLEAR|PURGE_RXCLEAR);
    return hSer;
    }

void main(int argc, char** argv)
    {
    int baudrate=19200;               // default value for baudrate
    HANDLE hSerial=NULL;              // windows handle for interface
    char txtbuffer[MAXNUMBYTES+1];    // ascii buffer for commands/ascii queries
    char header[16];                  // buffer to read the binary header into
    unsigned short tracebuf[TRLEN];   // binary buffer for trac:data? query
    unsigned long cnt;                // number of bytes actually written/read
    unsigned long numbytes;           // number of bytes to write/read

    // if argc>1, take argv[1] as the current baudrate
    if(argc>1)
        {
        baudrate = atoi(argv[1]);
        if(baudrate < 1200 || baudrate > 115200) baudrate = 19200;
        }

    // initialize the interface ...
    printf("Setting baudrate to %d!\n", baudrate);
    hSerial = InitSerial(baudrate);

    if(!hSerial)
        {
        printf("Failed to open %s, exiting!\n", INTERFACE);
        return;
        }

    // now start communicating ...
    sprintf(txtbuffer,"*CLS\n");
    numbytes = strlen(txtbuffer);
    WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
    sprintf(txtbuffer,"*IDN?\n");
    numbytes = strlen(txtbuffer);
    WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
    ReadFile(hSerial, txtbuffer, MAXNUMBYTES, &cnt, 0);
    if(cnt == 0 || strlen(txtbuffer) == 0)
        {
        printf("SCPI query failed, exiting!\n");
        CloseHandle(hSerial);
        return;
        }

    printf("Connected to: %s\n", txtbuffer);

    // setting measurement parameters ...
    sprintf(txtbuffer,"SOURCE:RANGE:START 0\n");          // measurement start
    numbytes = strlen(txtbuffer);
    WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
    sprintf(txtbuffer,"SOURCE:RANGE:SPAN 10km\n");        // measurement span
    numbytes = strlen(txtbuffer);
    WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
    sprintf(txtbuffer,"SOURCE:PULSE:WIDTH 100ns\n");      // pulsewidth
    numbytes = strlen(txtbuffer);
    WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
    sprintf(txtbuffer,"SOURCE:WAVELENGTH 1310nm\n");      // wavelength
    numbytes = strlen(txtbuffer);
```

```
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);

// start the measurement ...
printf("Starting measurement ...\n");
sprintf(txtbuffer,"INIT\n");
numbytes = strlen(txtbuffer);
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);

Sleep(15000); // give it 10s to run + 5s for init ...

// stop the measurement ...
printf("Stopping measurement ...\n");
sprintf(txtbuffer,"ABORT\n");
numbytes = strlen(txtbuffer);
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);
Sleep(1000); // wait a little for things to settle ...

// read the trace data ...
sprintf(txtbuffer,"TRACE:DATA?\n");
numbytes = strlen(txtbuffer);
WriteFile(hSerial, txtbuffer, numbytes, &cnt, 0);

// now comes the data: e.g. #48000.... which means:
//                            | 4 digits following to tell the number of bytes
//                             |||| 8000 bytes following, containing 4000 trace pts
cnt=0;
while(!cnt) ReadFile(hSerial, header, 1, &cnt, 0);  // read "#"
ReadFile(hSerial, header, 1, &cnt, 0);  // read number of digits
header[1]=0;
numbytes = atoi(header);
ReadFile(hSerial, header, numbytes, &cnt, 0); // read digits
header[cnt] = 0;
numbytes = atoi(header);
printf("Reading %d points of trace data ...\n", numbytes/2);
ReadFile(hSerial, tracebuf, numbytes, &cnt, 0);  // read trace data
ReadFile(hSerial, header, 15, &cnt, 0);          // read rest: <END>\n

// write the data to the console ...
for(unsigned int i=0; i<numbytes/2; i++)
    {
    printf("idx: %d, value: %d\n", i, tracebuf[i]);
    }

// close the interface
CloseHandle(hSerial);
return;
}
```

A

The VEE Driver

The VEE Driver

This appendix gives you extra information about using HP/Agilent OTDRs with the HP VEE VXI-plug&play driver.

You will find the driver on the update CD under `vxipnp/`.

A.1 What is HP VEE ?

Hewlett-Packard Visual Engineering Environment (HP VEE) is a visual programming language optimized for instrument control applications. To develop programs in HP VEE, you connect graphical ‘objects’ instead of writing lines of code. These programs resemble easy-to-understand block diagrams with lines.

HP VEE allows you to leverage your investment in textual languages by integrating with languages such as C, C++, Visual Basic, FORTRAN, Pascal, and HP BASIC.

HP VEE controls GPIB, VXI, Serial, GPIO, PC Plug-in, and LAN instruments directly over the interfaces or by using instrument drivers.

HP VEE supports VXI*plug&play* drivers in the WIN, WIN95, WINNT, and HP-UX frameworks. In addition, versions 3.2 and above of HP VEE support the graphical Function Panel interface, providing a function tree of the hierarchy of the driver.

NOTE **This appendix assumes that you are using Windows 95. If you are using Windows NT, please replace every reference to `win95` with `winnt`.**

Windows 95 and Windows NT are registered trademarks of Microsoft corporation.

HP VEE automatically calls the *initialize* and *close* functions to perform automatic error checking.

Using the RS232 port

HP VEE supports interfacing with an instrument from the RS232 port. Before you can do this, you must do the following:

1 Select INSTRUMENT MANAGER from the IO menu.

2 Double-click on the `Add` button to bring up the Device

Configuration screen.

3 Enter the following information:

- **`Name`**: choose any name to describe the instrument.
- **`Interface`**: `HP-IB` (even if you want to use the serial port).
- **`Address`**: key in any number (it does not matter which number you enter as you will only be using one of the serial ports).
- **`Gateway`**: `This host`.

4 Press `Advanced I/O Config`, and select the hpotdr *plug&play* Driver from a drop down list.

NOTE **If you do not see this driver in the list, it has not been installed properly.**

5 If you are planning to use the COMx port in the machine, specify the address of the instrument as `ASRLx`.

6 Select whether `Reset` and `Instrument Name Check` should be performed whenever VEE opens the instrument for interaction.'

7 Return to the Instrument Manager screen, and select `OK` to save the configuration.

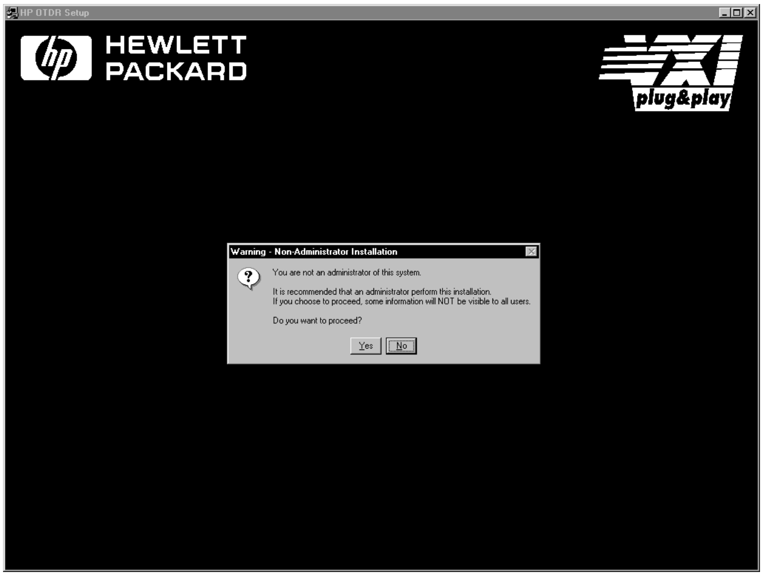
A.2 How to Install HP VEE

The HP VEE VxI*plug&play* driver comes as a self-extracting archive with an installation wizard. The installation wizard extracts all the files to preset destinations, asking you appropriate questions as it does so.

You install HP VEE by running the executable `OTDR.EXE`. When you run `OTDR.EXE`, you see a message telling you that the HP OTDR Instrument Driver will be installed.

1 Press `Yes` to continue.

You see a VXI*plug&play* window, and a message telling you that you are not an administrator (Figure A-1)



Warning - Non-Administrator Installation

You are not an administrator of this system.

It is recommended that an administrator perform this installation.
If you choose to proceed, some information will NOT be visible to all users.

Do you want to proceed?

Figure A-1 **VXI*plug&play* window**

2 Ignore this message, and press `Yes` to continue.

NOTE **If HP VEE is already installed on your system, you see a message asking you if you want to uninstall the old version.**

Press `Yes`, if required, then wait until you see a message telling you that the uninstall has been successful. Then press `OK` to continue.

You see a Welcome message, advising you to close the programs that you have running.

3 Close these programs and press `Next>` to continue.

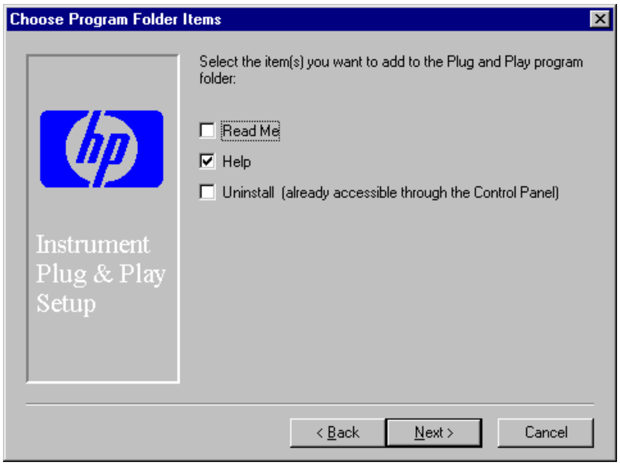
NOTE

If you do not have VISA installed, you see a message advising you to install VISA.

Press `Cancel` to temporarily exit this installation procedure; install VISA on your PC, then run `OTDR.EXE` again.

You see a window showing you what you can install (Figure A-2).

4 Select any or all of `Read Me`, `Help` and `Uninstall`, then press Next> to continue.


Choose Program Folder Items

Select the item(s) you want to add to the Plug and Play program folder:

- [ ] Read Me
- [x] Help
- [ ] Uninstall (already accessible through the Control Panel)

Figure A-2 **HP VEE - Install options**

You see a window asking you in which folder you want to install the files.

5 Select the default, `VXIPNP`, or choose a folder that you want. Press `Next>` to continue.

You see a message saying that setup is complete, giving you an option to view the Readme file.

6 Press `Finish` to complete installation, viewing the Readme file

if you wish.

A.3 Features of the HP OTDR VEE Driver

The HP OTDR VEE driver conforms to all aspects of the VXI*plug&play* driver standard which apply to conventional rack and stack instruments.

The following features are available:

- The VEE driver conforms with the VXI*plug&play* standard.

  There is one exception as the OTDR driver does not have a soft front panel or a knowledge-based file.

- The VEE driver is built on top of VISA, and uses the services provided.

  VISA supports GPIB and VXI protocols. The driver can be used with any GPIB card for which the manufacturer has provided a VISA DLL.

- The VEE driver includes a Function Panel (.FP) file.

  The .FP file allows the driver to be used with visual programming environments such as HP-VEE, LabWindows, and LabVIEW.

- The VEE driver includes a comprehensive on-line help file which complements the instrument manual.

  The help file contains application programming examples, a cross-reference between instrument commands and driver functions, and detailed documentation of each function with examples.

- The VEE driver includes a source, so that the driver can be modified if desired.

  The source conforms to VXI*plug&play* standards. You should only modify the driver if you are familiar with these standards.

- The VEE driver includes a Visual Basic (.BAS) file which contains the function calls in Visual Basic syntax, and allows the driver functions to be called from Visual Basic.

  You should only use Visual Basic with this driver if you are familiar with C/C++ function declarations. You must take particular care when working with C/C++ pointers.

A.4 Directory Structure

The setup program which installs the HP OTDR instrument driver creates the `VXIPNP` directory if it does not already exist. Windows 95 files are in `VXIPNP\WIN95`; Windows NT files are in `VXIPNP\WINNT`.

A.5 Opening an Instrument Session

To control an instrument from a program, you must open a communication path between the computer/controller and the instrument. This path is known as an instrument session, and is opened with the function

```
ViStatus hpotdr_init( ViRsrc InstrDesc,
ViBoolean id_query, ViBoolean reset,
ViPSession instrumentHandle );
```

Instruments are assigned a handle when the instrument session is opened. The handle, which is a pointer to the instrument, is the first parameter passed in all subsequent calls to driver functions.

The parameters of the function `hpotdr_init` include:

- **`ViRsrc InstrDesc`**: the address of the instrument

- **`ViBoolean id_query`**: a Boolean flag which indicates if in-system verification should be performed.
  Passing `VI_TRUE` (1) will perform an in-system verification; passing `VI_FALSE` (0) will not.
  If you set `id_query` to false, you can use the generic functions of the instrument driver with other instruments.
- **`ViBoolean reset`**: a Boolean flag which indicates if the instrument should be reset when it is opened.
  Passing `VI_TRUE` (1) will perform a reset when the session is opened; passing `VI_FALSE` (0) will not perform a reset,
- **`ViPSession instrumentHandle`**: a pointer to an instrument session.
  `InstrumentHandle` is the handle which addresses the instrument, and is the first parameter passed in all driver functions.

  Successful completion of this function returns `VI_SUCCESS`

A.6 Closing an Instrument Session

Sessions (instrumentHandle) opened with the `hpotdr_init()` function are closed with the function:

```
hpotdr_close( ViSession instrumentHandle);
```

When no further communication with an instrument is required, the session must be explicitly closed (`hpotdr_close()` function).

VISA does not remove sessions unless they are explicitly closed. Closing the instrument session frees all data structures and system resources allocated to that session.

A.7 VISA Data Types and Selected Constant Definitions

The driver functions use VISA data types. VISA data types are identified by the `Vi` prefix in the data type name (for example, `ViInt16`, `ViUInt16`, `ViChar`).

The file `visatype.h` contains a complete listing of the VISA data types, function call casts and some of the common constants.

NOTE **You can find a partial list of the type definitions and constant definitions for the `visatype.h` in the HP OTDR Instrument Driver Online Help.**

A.8 Error Handling

Events and errors within a instrument control program can be detected by polling (querying) the instrument. Polling is used in application development environments (ADEs) that do not support asynchronous activities where callbacks can be used.

Programs can set up and use polling as shown below.

1 Declare a variable to contain the function completion code.

```
ViStatus errStatus;
```

Every driver function returns the completion code `ViStatus`.

If the function executes with no I/O errors, driver errors, or instrument errors, `ViStatus` is 0 (`VI_SUCCESS`).

If an error occurs, `ViStatus` is a negative error code.

Warnings are positive error codes, and indicate the operation succeeded but special conditions exist.

2 Enable automatic instrument error checking following each function call.

```
hpotdr_errorQueryDetect
  (instrumentHandle, VI_TRUE);
```

When enabled, the driver queries the instrument for an error condition before returning from the function.

If an error occurred, `errStatus` (Step 1) will contain a code indicating that an error was detected (`hpotdr_INSTR_ERROR_DETECTED`).

3 Check for an error (or event) after each function.

```
errStatus = hpotdr_cmd(instrumentHandle,
  "MEAS:FREQ");
check(instrumentHandle, errStatus);
```

After the function executes, `errStatus` contains the completion code.

The completion code and instrument ID are passed to an error checking routine. In the above statement, the routine is called 'check'.

4 Create a routine to respond to the error or event.

A.9 Introduction to Programming

Selecting Functions

The functions in each category are identified below.

Application Functions

These functions do application level tasks. They are designed to allow quick and easy access to common instrument measurement sequences.

Application functions are instrument-specific, and can be used for common instrument measurement tasks.

Subsystem Functions

These functions combine multiple SCPI commands into a single, functional operation. They are designed to allow quick and easy access to common instrument command sequences.

Subsystem functions are instrument-specific, and cab be used for functional tasks.

Passthrough Functions

Passthrough functions pass SCPI commands directly to the instrument. These functions are used when there is not a driver function available to set or perform a particular operation.

Utility Functions

Utility functions perform a variety of standard tasks.

Example Programs

See the Online Help and Chapter 6 “Programming Examples”.

LabView

The 32-bit HP/Agilent OTDR driver can be used with LabVIEW 4.0 and above. LabVIEW 4.0 is a 32-bit version of LabVIEW which runs on Windows 95 and Windows NT.

To access the functions of the HP/Agilent OTDR instrument driver from within LabVIEW 4.0, select FILE from the main menu, then select the CONVERT CVI FP FILE submenu item.

In the file selection dialog box which appears, select `hpotdr.fp` and click on the `OK` button.

LabVIEW will create a series of VI's, one per driver function. It will create a file called `hpotdr.llb` which contains these VI's. This library of VI's can then be accessed like any other VI library in LabVIEW.

NOTE **You must use the 32-bit version of the HP/Agilent OTDR driver with LabVIEW 4.0.**

NOTE **LabView is a trademark of National Instruments Corporation**

LabWindows

The 32-bit HP/Agilent OTDR driver can be used with LabWindows 4.0 and above. LabWindows 4.0 is a 32-bit version of LabWindows which runs on Windows 95 and Windows NT.

To access the functions of the HP/Agilent OTDR driver from within LabWindows, select INSTRUMENT from the main menu, and then select the LOAD... submenu item.

In the file selection dialog box which appears, select `hpotdr.fp` and click on the `OK` button. LabWindows loads the function panel and instrument driver.

The driver now appears as a selection on the Instrument menu, and can be treated like any LabWindows driver.

NOTE **LabWindows is a trademark of National Instruments Corporation**

A.10 VISA-specific information

The following information is useful if you are using the driver with a version of VISA.

Instrument Addresses

When you are using HP VXI*plug&play* instrument drivers, you should enter the instrument addresses using only upper case letters. This is to ensure maximum portability.

For example, use `GPIB0::22` rather than `gpib0::22`.

Callbacks

Callbacks are not supported by this driver.

A.11 Using the HP OTDR VEE Driver in Application Development Environments

The sections contains suggestions as to how you can use `hpotdr_32.dll` within various application development environments.

Microsoft Visual C++ 4.0 (or higher) and Borland C++ 4.5 (or higher)

Please refer to your Microsoft Visual C++ or Borland C++ manuals for information on linking and calling DLLS.

The driver uses Pascal calling conventions.

You should rebuild the driver DLL in a different directory to the directory in which the driver was installed. This helps you to differentiate the changes.

Microsoft Visual Basic 4.0 (or higher)

Please refer to your Microsoft Visual Basic manual for information on calling DLLs.

The BASIC include file is `hpotdr.bas`. You can find this file in the directory `~vxipnp\win95\include`, where ~ is the directory in the VXIPNP variable.

By default, ~ is equivalent to `C:\`. This means that the file is in `C:\vxipnp\win95\include`.

You may also need to include the file `visa.bas`. `visa.bas` is provided with your VISA DLL.

HP VEE 3.2 (or higher)

Your copy of HP VEE for Windows contains a document titled *Using VXIplug&play drivers with HP VEE for Windows*. This document contains the detailed information you need for HP VEE applications.

LabWindows CVI/ (R) 4.0 (or higher)

The HP OTDR VEE driver is supplied as both a source code file, and as a Dynamic Link Library (.DLL) file.

There are several advantages to using the .DLL form of the driver, including those listed below:

- transportability across different computer platforms,
- a higher level of support for the compiled driver from Hewlett-Packard,
- a faster load time for your project.

LabWindows/CVI (R) will attempt by default to load the source version of the instrument driver. To load the DLL, you must include the file `HPOTDR.FP` in your project. `HPOTDR.FP` can be found in the directory `vxipnp\win95\hpotdr`.

Do not include `HPOTDR.C` in your project.

You must provide an include file for `HPOTDR.H`. You do this by ensuring that the directory `~vxipnp\win95\include` is added to the include paths (CVI Project Option menu).
~ is the directory in the VXIPNP variable. By default, ~ is equivalent to `C:\`. This means that the file is in `C:\vxipnp\win95\include`.

A.12 Online information

The latest copy of this driver and other HP VXI*plug&play* drivers can be obtained from the World Wide Web at:

```
http://www.agilent.com/find/inst_drivers
```

If you do not have web access, please contact your Agilent supplier, or use the version of `OTDR.EXE` on your installation CD.

Index

Index

Index