SCPI Commands

CMA5000(a) Platform

Third Edition

For safety and warning information, please read this manual before attempting to use the equipment.
Keep this manual with the equipment.

ANRITSU CORPORATION

Part No.: 42541

SCPI Commands – CMA5000(a) Standard OTDR Application
10 October 2009 (Third Edition)

Copyright © 2009, ANRITSU CORPORATION.
All rights reserved. No part of this manual may be reproduced without the prior written permission of the publisher.
The contents of this manual may be changed without prior notice.

Composed in the USA

About This Manual

This operation manual describes the SCPI (Standard Commands for Programmable Instruments) commands for the CMA5000 Platform.

Table of Contents

This section details the SCPI commands for the CMA5000 platform.

About This Manual iii
1 Star (IEEE 488.2) Subsystem Commands 1
1.1 *CLS Clear Status Command 1
1.2 *ESE Standard Event Status Enable Command 1
1.3 *ESE? Standard Event Status Enable Query 1
1.4 *ESR? Standard Event Status Register Query 2
1.5 *IDN? Identification Query 2
1.6 *OPC Operation Complete Command 2
1.7 *OPC? Operation Complete Query 3
1.8 *RST Reset Command 3
1.9 *SRE Service Request Enable Command 4
1.10 *SRE? Service Request Enable Query 4
1.11 *STB? Read Status Byte Query 5
1.12 *TST? Self-Test Query 5
1.13 *WAI Wait-to-Continue Command 5
2 System Subsystem Commands 6
2.1 SYSTem:ERRor? 6
2.2 SYSTem:VERSion? 6
3 Status Subsystem Commands 7
3.1 STATus:OPERation[:EVENt]? 7
3.2 PERation:CONDition? 7
3.3 STATus:OPERation:BIT#:CONDition? 7
3.4 STATus:OPERation:BIT#:ENABle 7
3.5 STATus:OPERation:BIT#:ENABle? 8
3.6 STATus:OPERation:BIT#[:EVENt]? 8
3.7 STATus:OPERation:ENABle 8
3.8 STATus:OPERation:ENABle? 8
3.9 STATus:OPERation:INSTrument:CONDition? 9
3.10 STATus:OPERation:INSTrument:ENABle 9
3.11 STATus:OPERation:INSTrument:ENABle? 9
3.12 STATus:OPERation:INSTrument[:EVENt]? 9
3.13 STATus:OPERation:INSTrument:ISUMmary#:CONDition? 10
3.14 STATus:OPERation:INSTrument:ISUMmary#:ENABle 10
3.15 STATus:OPERation:INSTrument:ISUMmary#:ENABle? 10
3.16 STATus:OPERation:INSTrument:ISUMmary#[:EVENt]? 10
3.17 STATus:QUEStionable[:EVENt]? 11
3.18 STATus:QUEStionable:CONDition? 11
3.19 STATus:QUEStionable:BIT#:CONDition? 11
3.20 STATus:QUEStionable:BIT#:ENABle 11
3.21 STATus:QUEStionable:BIT#:ENABle? 12

3.22 STATus:QUEStionable:BIT#[:EVENt]? ... 12
3.23 STATus:QUEStionable:ENABle ... 12
3.24 STATus:QUEStionable:ENABle? ... 12
3.25 STATus:QUEStionable:INSTrument:CONDition? ... 13
3.26 STATus:QUEStionable:INSTrument:ENABle ... 13
3.27 STATus:QUEStionable:INSTrument:ENABle? ... 13
3.28 STATus:QUEStionable:INSTrument[:EVENt]? ... 13
3.29 STATus:QUEStionable:INSTrument:ISUMmary#:CONDition? ... 14
3.30 STATus:QUEStionable:INSTrument:ISUMmary#:ENABle ... 14
3.31 STATus:QUEStionable:INSTrument:ISUMmary#:ENABle? ... 14
3.32 STATus:QUEStionable:INSTrument:ISUMmary#[:EVENt]? ... 14
3.33 STATus:PRESet ... 15
4 Instrument Subsystem Commands ... 16
4.1 INSTrument:CATalog? ... 16
4.2 INSTrument:CATalog:FULL? ... 16
4.3 INSTrument:NSELect ... 16
4.4 INSTrument:NSELect? ... 16
4.5 INSTrument[:SELect] ... 17
4.6 INSTrument[:SELect]? ... 17
4.7 INSTrument:STATe ... 17
4.8 INSTrument:STATe? ... 17
5 Configuration ... 18
5.1 General Requirements ... 18
5.1.1 Client and Server ... 18
5.1.2 Connection Port ... 18
5.1.3 Startup ... 18
5.2 Connecting from Hyper Terminal ... 18
5.3 Writing Custom C Client ... 20
5.4 Writing Custom Java Client ... 25
5.5 Generic Error Codes ... 29

1 Star (IEEE 488.2) Subsystem Commands

1.1 *CLS Clear Status Command

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. The command's effect on individual logical instrument depends on the instrument implementation and is described in the instrument's documentation.
Parameters:	None
Response:	None
Example:	`*CLS`

1.2 *ESE Standard Event Status Enable Command

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**): 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`

1.3 *ESE? Standard Event Status Enable Query

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>`

1.4 *ESR? Standard Event Status Register Query

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** 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? –> 22<END>`

1.5 *IDN? Identification Query

Syntax:	*IDN?
Description:	The Identification query *IDN? gets the instrument identification over the interface.
Parameters:	None
Response:	The identification string terminated by <END>
Example:	`*IDN? –> ANRITSU, CMA5000, 1990.0 1.00<END>`

1.6 *OPC Operation Complete Command

Syntax:	*OPC
Description:	A device is in the Operation Complete Command Active State (OCAS) after it has executed *OPC. The device returns to the Operation Complete Command Idle State (OCIS) whenever the No Operation Pending flag is TRUE, at the same time setting the OPC bit of the SESR TRUE. The following events force the device into OCIS without setting the No Operation Pending flag TRUE and without setting the OPC bit of the SESR: **.** Power-on **.** the Device Clear Active State is asserted on the interface. **.** *CLS **.** *RST
Parameters:	None
Response:	None
Example:	`*OPC`

1.7 *OPC? Operation Complete Query

Syntax:	*OPC?
Description:	A device is in the Operation Complete Query Active State (OQAS) after it has executed *OPC?. The device returns to the Operation Complete Query Idle State (OQIS) whenever the No Operation Pending flag is TRUE, at the same time placing a “1" in the Output Queue. 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> or 1<END>
Example:	`*OPC? –> 1<END>`

1.8 *RST Reset Command

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)
Parameters:	None
Response:	None
Example:	`*RST`

1.9 *SRE Service Request Enable Command

Syntax:	*SRE<wsp><value>
Description:	The standard Service Request Enable command (*SRE) sets bits in the Service Request Enable register. A 1 in a bit in the enable register enables the corresponding bit in the Service Request Enable 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**): 7 Operation Status Summary 128 6 Master Summary Status (MSS) / Request Service (RQS) 64 5 Standard Event Status Summary (ESB) 32 4 Message Available (MAV) 16 3 Questionable Status Summary 8 2 Error/Event Queue Summary 4 1 Available 2 0 Available 1
Response:	None
Example:	`*SRE 64`

1.10 *SRE? Service Request Enable Query

Syntax:	*SRE?
Description:	The Service Request Enable query *SRE? returns the contents of the Service Request Enable register (see *SRE for information on this register).
Parameters:	None
Response:	The bit value for the register (a **short** value). 7 Operation Status Summary 128 6 Master Summary Status (MSS) / Request Service (RQS) 64 5 Standard Event Status Summary (ESB) 32 4 Message Available (MAV) 16 3 Questionable Status Summary 8 2 Error/Event Queue Summary 4 1 Available 2 0 Available 1
Example:	`*SRE? –> 21<END>`

1.11 *STB? Read Status Byte Query

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): 7 Operation Status Summary 128 6 Master Summary Status (MSS) / Request Service (RQS) 64 5 Standard Event Status Summary (ESB) 32 4 Message Available (MAV) 16 3 Questionable Status Summary 8 2 Error/Event Queue Summary 4 1 Available 2 0 Available 1
Example:	`*STB? –> 78<END>`

1.12 *TST? Self-Test Query

Syntax:	*TST?
Description:	The self-test query *TST? makes the currently selected logical instrument to 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>`

1.13 *WAI Wait-to-Continue Command

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`

2 System Subsystem Commands

2.1 SYSTem:ERRor?

Syntax:	SYSTem:ERRor?
Description:	Returns the contents of the SCPI 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>`

2.2 SYSTem:VERSion?

Syntax:	SYSTem:VERSion?
Description:	Returns the SCPI revision to which the system complies.
Parameters:	None
Response:	The revision year and number string.
Example:	`SYST:VERS? -> 1995.0<END>`

3 Status Subsystem Commands

3.1 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>`

3.2 PERation: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>`

3.3 STATus:OPERation:BIT#:CONDition?

Syntax:	STATus:OPERation:BIT<n>:CONDition?
Description:	This command accesses the user-definable bits in the OPERation register set. The value of <n> is restricted from 8 to 12 and represents bits 8 through 12 in the :STATus:OPERation status register.
Parameters:	None
Response:	The bit value for the operation condition register as a short value (0 .. 1)
Example:	`STAT:OPER:BIT8:COND? –> 1<END>`

3.4 STATus:OPERation:BIT#:ENABle

Syntax:	STATus:OPERation:BIT<n>:ENABle<wsp><value>
Description:	Sets the operation enable mask for the event register specified bit. The value of <n> is restricted from 8 to 12 and represents bits 8 through 12
Parameters:	The bit value for the operation enable mask as a **short** value (0 .. 1)
Response:	None
Example:	`STAT:OPER:BIT11:ENAB 1`

3.5 STATus:OPERation:BIT#:ENABle?

Syntax:	STATus:OPERation:BIT<n>:ENABle?
Description:	Returns the operation enable mask for the event register specified bit. The value of <n> is restricted from 8 to 12 and represents bits 8 through 12
Parameters:	None
Response:	The bit value for the operation enable mask as a **short** value (0 .. 1)
Example:	`STAT:OPER:BIT9:ENAB? –> 0<END>`

3.6 STATus:OPERation:BIT#[:EVENt]?

Syntax:	STATus:OPERation:BIT<n>[:EVENt]?
Description:	Queries the operation event register specified bit. The value of <n> is restricted from 8 to 12 and represents bits 8 through 12
Parameters:	None
Response:	The bit value for the operation event register as a **short** value (0 .. 1)
Example:	`STAT:OPER:BIT10:EVEN? –> 0<END>`

3.7 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`

3.8 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>`

3.9 STATus:OPERation:INSTrument:CONDition?

Syntax:	STATus:OPERation:INSTrument:CONDition?
Description:	Queries the instrument operation condition register.
Parameters:	None
Response:	The bit value for the operation condition register as a **short** value (0 .. +32767)
Example:	`STAT:OPER:INST:COND? –> 16<END>`

3.10 STATus:OPERation:INSTrument:ENABle

Syntax:	STATus:OPERation:INSTrument:ENABle<wsp><value>
Description:	Sets the instrument 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:INST:ENAB 128`

3.11 STATus:OPERation:INSTrument:ENABle?

Syntax:	STATus:OPERation:INSTrument:ENABle?
Description:	Returns the instrument 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:INST:ENAB? –> 128<END>`

3.12 STATus:OPERation:INSTrument[:EVENt]?

Syntax:	STATus:OPERation:INSTrument[:EVENt]?
Description:	Queries the instrument operation event register.
Parameters:	None
Response:	The bit value for the operation event register as a **short** value (0 .. +32767)
Example:	`STAT:OPER:INST? –> 0<END>`

3.13 STATus:OPERation:INSTrument:ISUMmary#:CONDition?

Syntax:	STATus:OPERation:INSTrument:ISUMmary<n>:CONDition?
Description:	Queries the instrument operation condition register of the specified instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	None
Response:	The bit value for the operation condition register as a **short** value (0 .. +32767)
Example:	`STAT:OPER:INST:ISUM4:COND? –> 16<END>`

3.14 STATus:OPERation:INSTrument:ISUMmary#:ENABle

Syntax:	STATus:OPERation:INSTrument:ISUMmary<n>:ENABle<wsp><value>
Description:	Sets the instrument operation enable mask for the event register of the specified instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	The bit value for the operation enable mask as a **short** value (0 .. +32767)
Response:	None
Example:	`STAT:OPER:INST:ISUM1:ENAB 128`

3.15 STATus:OPERation:INSTrument:ISUMmary#:ENABle?

Syntax:	STATus:OPERation:INSTrument:ISUMmary<n>ENABle?
Description:	Returns the instrument operation enable mask for the event register of the specified instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	None
Response:	The bit value for the operation enable mask as a **short** value (0 .. +32767)
Example:	`STAT:OPER:INST:ISUM4:ENAB? –> 128<END>`

3.16 STATus:OPERation:INSTrument:ISUMmary#[:EVENt]?

Syntax:	STATus:OPERation:INSTrument:ISUMmary<n>[:EVENt]?
Description:	Queries the instrument operation event register of the specified instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	None
Response:	The bit value for the operation event register as a **short** value (0 .. +32767)
Example:	`STAT:OPER:INST:ISUM3? –> 0<END>`

3.17 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>`

3.18 STATus:QUEStionable:CONDition?

Syntax:	STATus:QUEStionable:CONDition?
Description:	Queries the questionable 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>`

3.19 STATus:QUEStionable:BIT#:CONDition?

Syntax:	STATus:QUEStionable:BIT<n> :CONDition?
Description:	Queries the questionable condition register specified bit. The value of <n> is restricted from 9 to 12 and represents bits 9 through 12
Parameters:	None
Response:	The bit value for the questionable condition register as a **short** value (0 .. 1)
Example:	`STAT:QUES:BIT9:COND? –> 0<END>`

3.20 STATus:QUEStionable:BIT#:ENABle

Syntax:	STATus:QUEStionable:BIT<n>:ENABle<wsp><value>
Description:	Sets the questionable enable mask for the event register specified bit. The value of <n> is restricted from 9 to 12 and represents bits 9 through 12
Parameters:	The bit value for the questionable enable mask as a **short** value (0 .. 1)
Response:	None
Example:	`STAT:QUES:BIT11:ENAB 1`

3.21 STATus:QUEStionable:BIT#:ENABle?

Syntax:	STATus:QUEStionable:BIT<n>:ENABle?
Description:	Returns the questionable enable mask for the event register specified bit. The value of <n> is restricted from 9 to 12 and represents bits 9 through 12
Parameters:	None
Response:	The bit value for the questionable enable mask as a **short** value (0 .. 1)
Example:	`STAT:QUES:BIT10:ENAB? –> 1<END>`

3.22 STATus:QUEStionable:BIT#[:EVENt]?

Syntax:	STATus:QUEStionable:BIT<n>[:EVENt]?
Description:	Queries the questionable event register specified bit. The value of <n> is restricted from 9 to 12 and represents bits 9 through 12
Parameters:	None
Response:	The bit value for the questionable event register as a **short** value (0 .. 1)
Example:	`STAT:QUES:BIT9:EVEN? –> 0<END>`

3.23 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`

3.24 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>`

3.25 STATus:QUEStionable:INSTrument:CONDition?

Syntax:	STATus:QUEStionable:INSTrument :CONDition?
Description:	Queries the questionable instrument condition register.
Parameters:	None
Response:	The bit value for the questionable condition register as a **short** value (0 .. +32767)
Example:	`STAT:QUES:INST:COND? –> 8<END>`

3.26 STATus:QUEStionable:INSTrument:ENABle

Syntax:	STATus:QUEStionable:INSTrument:ENABle<wsp><value>
Description:	Sets the questionable instrument 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:INST:ENAB 128`

3.27 STATus:QUEStionable:INSTrument:ENABle?

Syntax:	STATus:QUEStionable:INSTrument:ENABle?
Description:	Returns the questionable instrument 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:INST:ENAB? –> 128<END>`

3.28 STATus:QUEStionable:INSTrument[:EVENt]?

Syntax:	STATus:QUEStionable:INSTrument[:EVENt]?
Description:	Queries the questionable instrument event register
Parameters:	None
Response:	The bit value for the questionable event register as a **short** value (0 .. +32767)
Example:	`STAT:QUES:INST:EVEN? –> 0<END>`

3.29 STATus:QUEStionable:INSTrument:ISUMmary#:CONDition?

Syntax:	STATus:QUEStionable:INSTrument:ISUMmary<n> :CONDition?
Description:	Queries the specified logical instrument questionable instrument condition register. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	None
Response:	The bit value for the questionable condition register as a **short** value (0 .. +32767)
Example:	`STAT:QUES:INST:ISUM2:COND? –> 0<END>`

3.30 STATus:QUEStionable:INSTrument:ISUMmary#:ENABle

Syntax:	STATus:QUEStionable:INSTrument:ISUMmary<n>:ENABle<wsp><value>
Description:	Sets the questionable instrument enable mask for the event register of the specified logical instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	The bit value for the questionable enable mask as a **short** value (0 .. +32767)
Response:	None
Example:	`STAT:QUES:INST:ISUM3:ENAB 128`

3.31 STATus:QUEStionable:INSTrument:ISUMmary#:ENABle?

Syntax:	STATus:QUEStionable:INSTrument:ISUMmary<n>:ENABle?
Description:	Returns the questionable instrument enable mask for the event register of the specified logical instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	None
Response:	The bit value for the questionable enable mask as a **short** value (0 .. +32767)
Example:	`STAT:QUES:INST:ISUM3:ENAB? –> 136<END>`

3.32 STATus:QUEStionable:INSTrument:ISUMmary#[:EVENt]?

Syntax:	STATus:QUEStionable:INSTrument:ISUMmary[:EVENt]?
Description:	Queries the questionable instrument event register of the specified logical instrument. The value of <n> is restricted from 1 to 14 and represents the logical instruments id assigned to the SCPI controlled instrument by the INSTrument subsystem.
Parameters:	None
Response:	The bit value for the questionable event register as a **short** value (0 .. +32767)
Example:	`STAT:QUES:INST:ISUM4:EVEN? –> 0<END>`

3.33 STATus:PRESet

Syntax:	STATus:PRESet
Description:	Resets both the operation enable mask and questionable enable mask to 0
Parameters:	None
Response:	None
Example:	STAT:PRES

4 Instrument Subsystem Commands

4.1 INSTrument:CATalog?

Syntax:	INSTrument:CATalog?
Description:	Returns the list of SCPI controllable instruments on CMA5000 that are identified as SCPI controllable and have available plug-ins to run them
Parameters:	None
Response:	comma-separated list of string identifiers of all SCPI controllable logical instruments
Example:	`INST:CAT? –> STATUS1, OTDR_STD1, ATX_SONET1<END>`

4.2 INSTrument:CATalog:FULL?

Syntax:	INSTrument:CATalog:FULL?
Description:	Returns the list of SCPI controllable instruments on CMA5000 that are identified as SCPI controllable and have available plug-ins to run them
Parameters:	None
Response:	Returns a list of string - number pairs. The string contains the name identifier of the logical instrument. The immediately following NR1-formatted number is its associated logical instrument number. All response data elements are comma separated.
Example:	`INSTrument:CATalog:FULL? –> STATUS1,1, OTDR_STD1,2<END>`

4.3 INSTrument:NSELect

Syntax:	INSTrument:NSELect<wsp><num_id>
Description:	Sets the specified logical instrument to be currently selected instrument.
Parameters:	The numeric value identifier assigned by the Instrument subsystem for the instrument to be selected as a **short** value
Response:	None
Example:	`INSTrument:NSELect 2`

4.4 INSTrument:NSELect?

Syntax:	INSTrument:NSELect?
Description:	Returns the numeric value identifier of the currently selected logical instrument
Parameters:	None
Response:	Returns a numeric value of the currently selected logical instrument
Example:	`INSTrument:NSELect? –> 2<END>`

4.5 INSTrument[:SELect]

Syntax:	INSTrument:SELect<wsp><string_id>
Description:	Sets the specified logical instrument to be currently selected instrument
Parameters:	The string instrument identifier assigned by the Instrument subsystem for the instrument to be selected as a **string** value
Response:	None
Example:	`INSTrument:SELect OTDR_STD1`

4.6 INSTrument[:SELect]?

Syntax:	INSTrument:SELect?
Description:	Returns the string value identifier of the currently selected logical instrument
Parameters:	None
Response:	Returns the string value identifier of the currently selected logical instrument
Example:	`INSTrument:SELect? –> OTDR_STD1<END>`

4.7 INSTrument:STATe

Syntax:	INSTrument:STATe<wsp><boolean>
Description:	Turns the currently selected logical instrument ON or OFF
Parameters:	The Boolean data (ON or 1) to turn instrument on and (OFF or 0) to turn instrument off
Response:	None
Example:	`INSTrument: STATe ON`

4.8 INSTrument:STATe?

Syntax:	INSTrument:STATe?
Description:	Returns the state of the currently selected logical instrument
Parameters:	None
Response:	Returns a boolean value of the state of the currently selected logical instrument (1 if the instrument is active, 0 if the instrument is inactive)
Example:	`INSTrument: STATe? –> 1<END>`

5 Configuration

5.1 General Requirements

5.1.1 Client and Server

CMA5000 represents server which executes SCPI service. It allows remote clients connecting to it one at the time.

Client can be implemented in many different ways. It can be properly configured Windows Hyper Terminal, small user application or large control system that manages many units simultaneously.

CMA5000 SCPI server uses an UNIX socket configured to listen at port 2288. It receives processes and executes SCPI commands. While CMA5000 responds to the SCPI client requests, its graphical user interface (GUI) is simultaneously available as well.

5.1.2 Connection Port

For outside connections, CMA5000 SCPI server is listening at port 2288.

5.1.3 Startup

Before powering CMA5000 on for remote connection, it needs to be connected to the local area network (LAN). This is important, as during the boot process, the unit establishes it's network connection, making it's services available to the network. Connecting CMA5000 to the LAN after the boot will not allow outside connections.

Here is the list of initial commands required for operation of any instrument via SCPI interface:

INSTrument:CATalog?

Identifies the list of available remote instruments. The response may look like: "STATUS1,OSA1". STATUS1 is the CMA5000's default status instrument. OSA1 is plugged in OSA instrument. It could show any other instrument which represents the module inserted into CMA5000.

INSTrument:SELect OSA1

Makes OSA instrument current.

INSTrument:STATe ON

Instantiates the current instrument by setting its state to "ON". At this point OSA instrument should be ready to receive commands.

5.2 Connecting from Hyper Terminal

Windows Hyper Terminal may be used to quickly verify the SCPI operation. It can not handle binary responses, but works good with the text and most commands have textual response.

Figure 5.2.1 illustrates the connection screen of the Hyper Terminal. The Host Address is the IP Address of the CMA5000 unit connected to the LAN. Figures 5.2.2 and 5.2.3 show other important settings. Figure 5.2.4 shows examples of commands executed through Hyper Terminal.

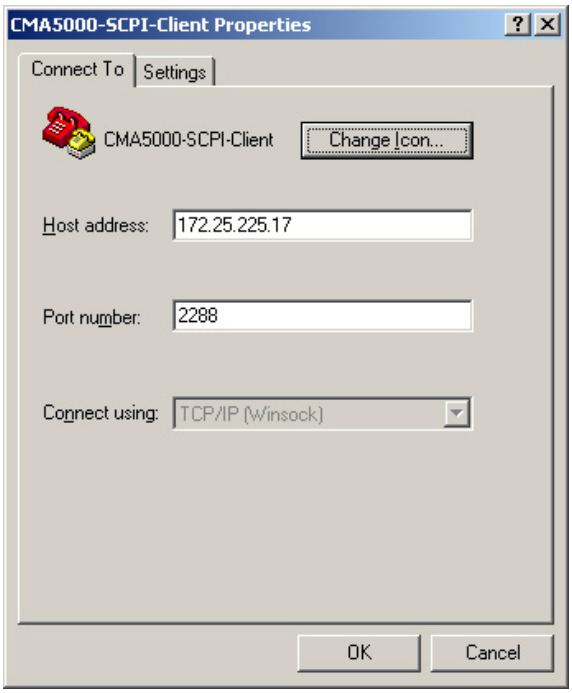

Figure 5.2.1 Connection screen

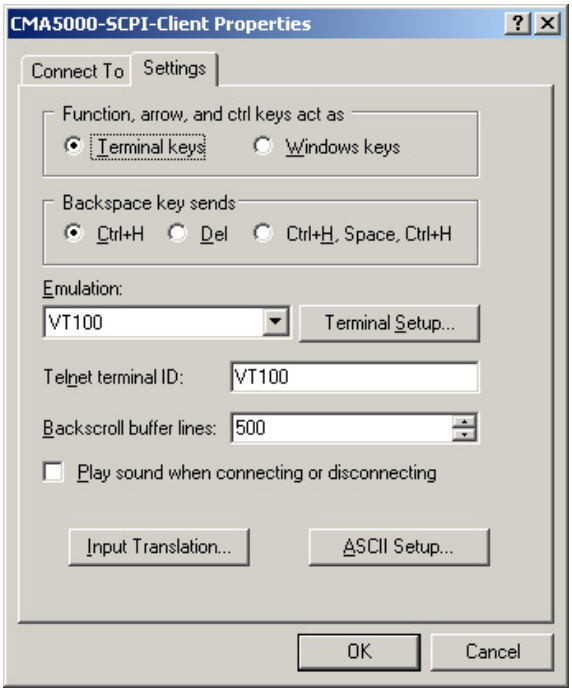

Figure 5.2.2 Settings

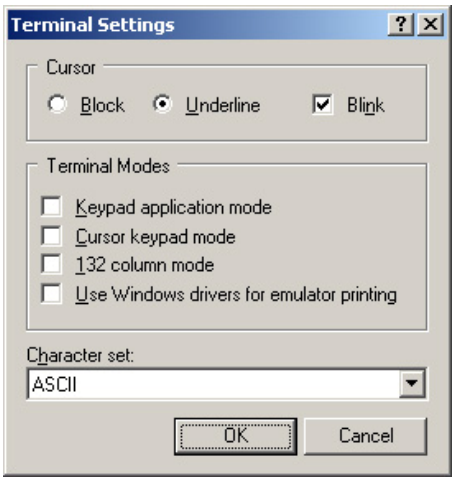

Figure 5.2.3 ASCII Setup

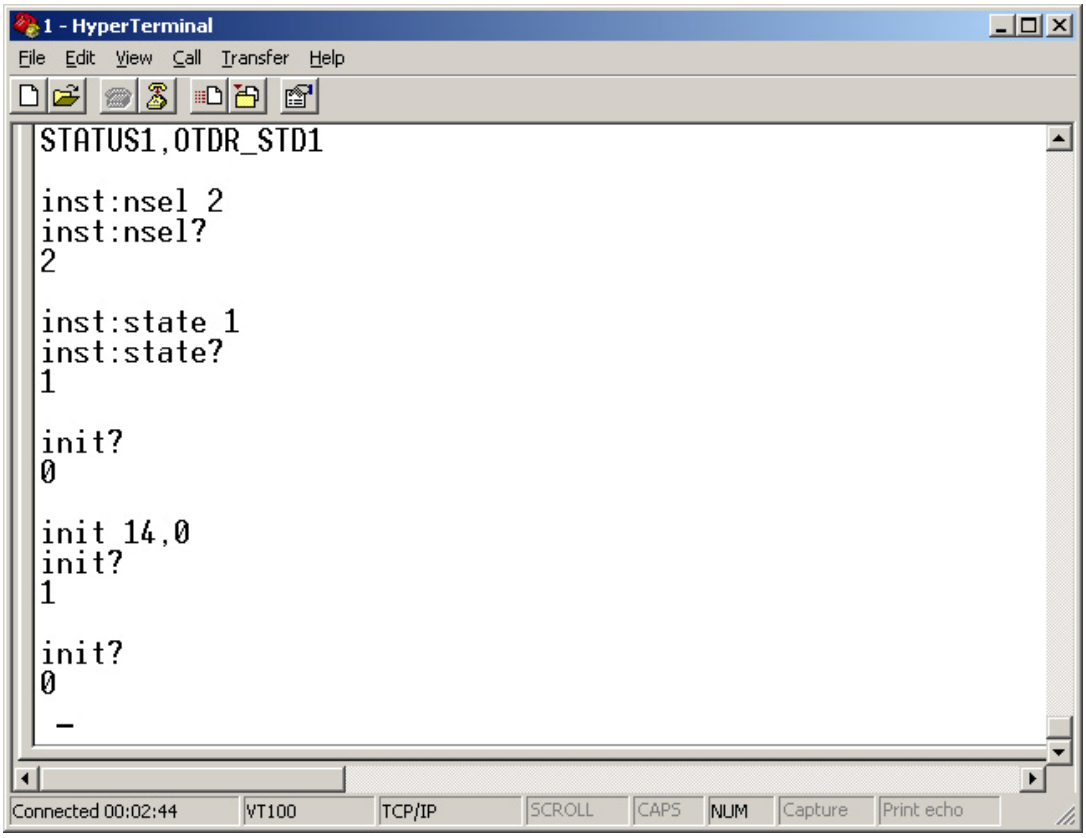

Figure 5.2.4 Hyper Terminal window

5.3 Writing Custom C Client

Below is an example of a simple C programming language client that demonstrates that can be used as a starting point of program that is customized for a particular application. Sample is developed using Microsoft Visual C++ version 6.0. Sample can controls OTDR module hosted by

the CMA5000 controller. The IP address of the CMA5000 unit, has to be passed as a command line argument

```
#include "stdafx.h"
#include "winsock2.h"
#include <stdio.h>
#include <stdlib.h>

#define PORT 2288

void Send(SOCKET s, char* pszData)
{
  WORD wTx;

  WORD wRemaining = strlen(pszData);
  WORD wIndex = 0;

  while (wRemaining != 0)
  {
       wTx = send(s, &pszData[wIndex], wRemaining, 0);
       if (wTx == -1)
       {
              perror("send");
              return;
       }

       wRemaining -= wTx;
       wIndex += wTx;
  }

  // Flush command ... send new line
  wRemaining = 1;
  while (wRemaining != 0)
  {
       wTx = send(s, "\n", wRemaining, 0);
       if (wTx == -1)
       {
              perror("send");
              return;
       }

       wRemaining -= wTx;
  }
}
```

```
void Receive(SOCKET s, char* pszData, WORD wRemaining)
{
  WORD wIndex = 0;

  char* pEOT = NULL;

  while (pEOT == NULL)
  {
       WORD wRx = recv(s, &pszData[wIndex], wRemaining, 0);
       if (wRx == -1)
       {
              perror("Receive");
              return;
       }

       wRemaining -= wRx;
       wIndex += wRx;

       pEOT = strstr(pszData, "\n");
  }

  // remove terminator
  if (pEOT != NULL)
       *pEOT = 0;
}

void SendCmd(SOCKET s, char* pszCmd, char* pszReply, WORD wRemaining)
{
  Send(s, pszCmd);
  Receive(s, pszReply, wRemaining);
  printf("Command [%s] received [%s]\n", pszCmd, pszReply);
}

void main(int argc, const char* argv[])
{
  char hostIp[256];
  if (argc == 2)
  {
       sprintf(hostIp, "%s", argv[1]);
       printf("Connecting to: %s\n", hostIp);
  }
  else
  {
```

```
     printf("Wrong number of arguments! Please pass IP address.\n");
     return;
}

char szReply[512];
char szCmd[512];

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

//----------------------
// Create a SOCKET for connecting to server
SOCKET ConnectSocket;
ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (ConnectSocket == INVALID_SOCKET)
{
     printf("Error at socket(): %ld\n", WSAGetLastError());
     WSACleanup();
     return;
}

//----------------------
// The sockaddr_in structure specifies the address family,
// IP address, and port of the server to be connected to.
sockaddr_in clientService;
clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = inet_addr( hostIp );
clientService.sin_port = htons( PORT );

//----------------------
// Connect to server.
if    (    connect(    ConnectSocket,    (SOCKADDR*)    &clientService,
sizeof(clientService) ) == SOCKET_ERROR)
{
     printf( "Failed to connect!\n" );
     WSACleanup();
     return;
}

printf("Connected to server.\n");
```

```
// Retrieve Version from CMA
SendCmd(ConnectSocket, "SYST:ERR?", szReply, sizeof(szReply));
SendCmd(ConnectSocket, "*IDN?", szReply, sizeof(szReply));
SendCmd(ConnectSocket, "SYST:VERS?", szReply, sizeof(szReply));
SendCmd(ConnectSocket, "INST:CAT:FULL?", szReply, sizeof(szReply));

// Select OTDR
sprintf(szCmd, "inst:sel  OTDR_STD1");
Send(ConnectSocket, szCmd);
sprintf(szCmd, "inst:sel?");
SendCmd(ConnectSocket, szCmd, szReply, sizeof(szReply));

// Start OTDR Standard
sprintf(szCmd, "%s 1;%s?", "inst:stat", "inst:stat");
SendCmd(ConnectSocket, szCmd, szReply, sizeof(szReply));

// Start Scan
sprintf(szCmd, "init 14,0");
Send(ConnectSocket, szCmd);

// Check Status
sprintf(szCmd, "init?");
SendCmd(ConnectSocket, szCmd, szReply, sizeof(szReply));

while (atoi(szReply) == 1)
{
     // Check Status
     sprintf(szCmd, "init?");
     Sleep(1000);
     SendCmd(ConnectSocket, szCmd, szReply, sizeof(szReply));
}

// Check Status
sprintf(szCmd, "init?");
SendCmd(ConnectSocket, szCmd, szReply, sizeof(szReply));

// Stop OTDR Standard
sprintf(szCmd, "%s 0;%s?", "inst:stat", "inst:stat");
SendCmd(ConnectSocket, szCmd, szReply, sizeof(szReply));

WSACleanup();

printf("Press any key to close ...\n");
```

```
  while (!kbhit())
       Sleep(100);

  return;
}
```

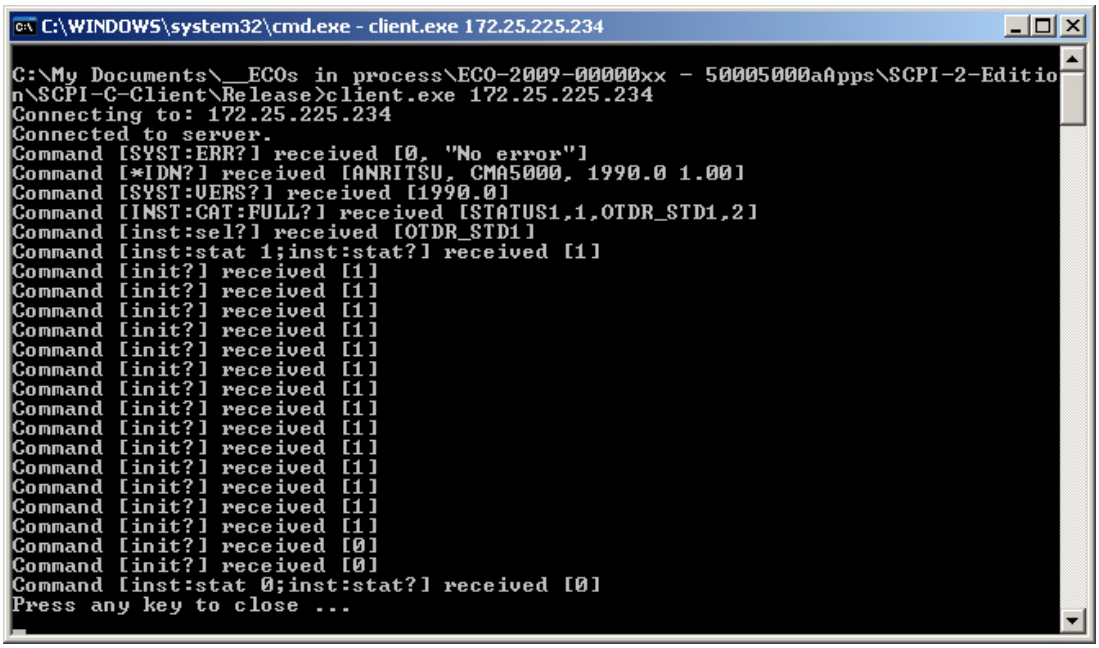

Figure 5.3.1 Program output

Figure 5.3.1 displays the screen shot of the program output. SCPI commands can be subdivided into two different categories: query commands which return requested information and set commands which do not return any information. Function SendCmd is suitable for query commands as it returns and prints the result of the executed query command. Function Send executes commands of the second type and does not return anything.

Worth noting, that every query command is terminated by the sender with new line character. When processing such a command, read operation must execute until the new line character is received. It is practically important to observe such behavior when working with buffered input stream, as there is no guarantee that the entire response will come in one read operation. Another possibility is that the string returned from read operation, may contain some data past the new line character. This is an acceptable behavior and client must interpret the data past new line as the return from the next command.

5.4 Writing Custom Java Client

For added convenience an example Java program is included below. It implements simple SCPI client. Sample is developed using Java SDK version 1.4. It controls OTDR module hosted by the CMA5000 controller. The IP address of the CMA5000 unit has to be passed as a command line argument. Figure 5.4.1 displays the screen shot of the program output.

```
package scpi;

import java.io.*;
import java.net.*;

public class Client
{
  static final int PORT = 2288;

  private Socket socket_ = null;
  private BufferedReader reader_ = null;
  private PrintWriter writer_ = null;

  public Client(String hostIp)
  {
    try
    {
      socket_ = new Socket(hostIp, PORT);
      reader_     =     new     BufferedReader(new     InputStreamReader(new
   DataInputStream(socket_.getInputStream())));
      writer_ = new PrintWriter(socket_.getOutputStream(), true /* autoFlush
   */);
    }
    catch (Exception e)
    {
      System.out.println("Failed to connect!");
    }

    String szReply;

    System.out.println("Connected to server.");

    // Retrieve Version from CMA
    szReply = Send(writer_, reader_, "SYST:ERR?", true);

    szReply = Send(writer_, reader_, "*IDN?", true);

    szReply = Send(writer_, reader_, "SYST:VERS?", true);

    szReply = Send(writer_, reader_, "INST:CAT:FULL?", true);

    // Select OTDR
    Send(writer_, reader_, "inst:sel  OTDR_STD1", false);
```

```
szReply = Send(writer_, reader_, "inst:sel?", true);

// Start OTDR Standard
szReply = Send(writer_, reader_, "inst:stat 1;inst:stat?", true);

// Start Scan
Send(writer_, reader_, "init 14,0", false);

// Check Status
szReply = Send(writer_, reader_, "init?", true);

while (szReply.equals("1"))
{
  // Check Status
  try { Thread.sleep(1000); }
  catch (Exception e) { e.printStackTrace(); }
  szReply = Send(writer_, reader_, "init?", true);
}

// Check Status
szReply = Send(writer_, reader_, "init?", true);

// Stop OTDR Standard
szReply = Send(writer_, reader_, "inst:stat 0;inst:stat?", true);

try
{
  if (socket_ != null)
    socket_.close();
}
catch (Exception e)
{
  e.printStackTrace();
}

System.out.println("Press any key to close ...\n");

try
{
  while (System.in.available() <= 0)
  {
    Thread.sleep(100);
  }
```

```
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  String Send(PrintWriter writer, BufferedReader reader, String cmd, boolean
   readReply)
  {
    if (writer != null && reader != null)
    {
      try
      {
        writer.println(cmd);
        if (readReply)
        {
          String reply = reader.readLine();
          System.out.println("Command [" + cmd + "] received [" + reply + "]");
          return reply;
        }
        System.out.println("Command [" + cmd + "]");
      }
      catch (IOException ioe)
      {
        ioe.printStackTrace();
      }
    }
    return "";
  }

  public static void main(String[] args)
  {
    String hostIp;
    if (args.length == 1)
    {
      hostIp = args[0];
      System.out.println("Connecting to: " + hostIp);
      Client client = new Client(hostIp);
    }
    else
    {
      System.out.println("Wrong number of arguments! Please pass IP address.");
      return;
```

```
    }
  }
}
```

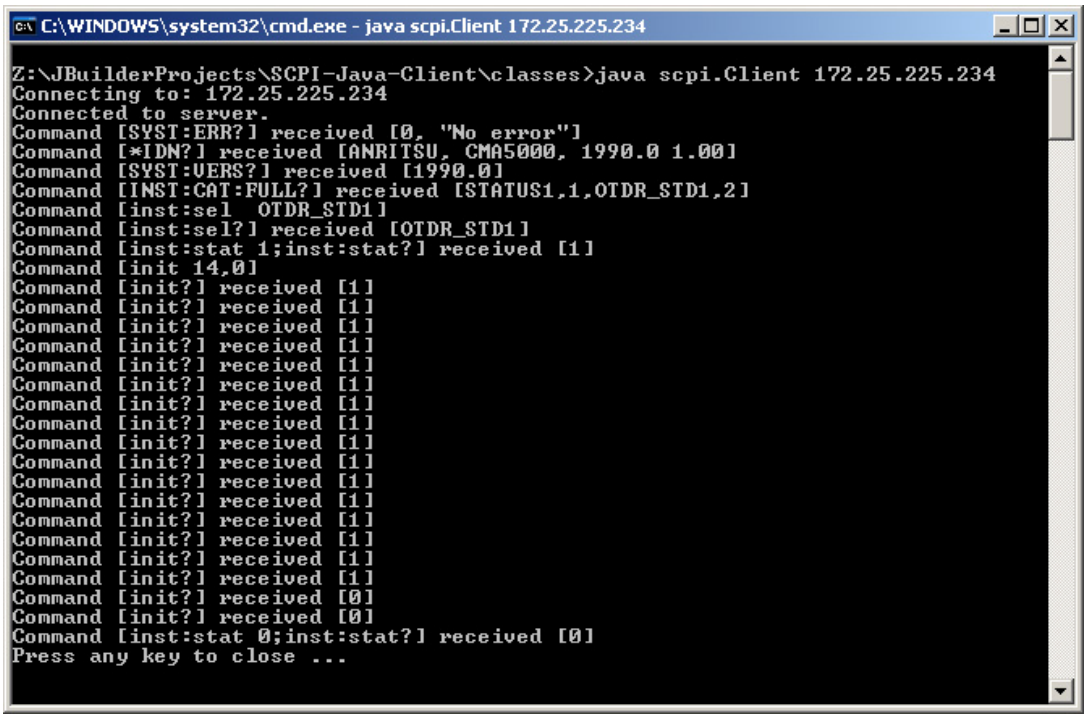

Figure 5.4.1 Program output

5.5 Generic Error Codes

This section details the list of Generic Error Codes that can be returned by SCPI Remote Control "SYSTem:ERRor?" command.

(0, "No error") No errors in the queue
(-350,"Queue overflow") Error queue is full (more then 12 errors generated and not queried)
(-100, "Command error") Command is not recognized by the SCPI Remote Control
(-102, "Syntax error") Command syntax error
(-104, "Data type error") Wrong parameter type passed to the command
(-108, "Parameter not allowed") Extra parameter passed to the command
(-109, "Missing parameter") Missing parameter
(-200, "Error INITIALIZING") Instrument subsystem could not initialize

Note, that particular Instrument Subsystem commands have list of error codes under each command description section in the provided document.

SCPI Commands

CMA5000(a)
Standard OTDR Application

Third Edition

For safety and warning information, please read this manual before attempting to use the equipment.
Keep this manual with the equipment.

ANRITSU CORPORATION

Part No.: 42469

SCPI Commands – CMA5000(a) Standard OTDR Application
10 October 2009 (Third Edition)

Copyright © 2009, ANRITSU CORPORATION.
All rights reserved. No part of this manual may be reproduced without the prior written permission of the publisher.
The contents of this manual may be changed without prior notice.

Composed in the USA

Table of Contents

This section details the SCPI commands for the CMA5000 Standard OTDR application. The Command Summary section presents a brief summary of each command, while each command is detailed in the subsequent sections.

Table of Contents iii
1 Command Summary 7
2 Root Level Commands 12
2.1 ABORt Command 12
2.2 INITiate Command 12
2.3 INITiate AUTo Command 13
2.4 INITiate? Command 13
3 Sensing Commands 14
3.1 SENSe:AVERages:COMPleted? Command 14
3.2 SENSe:FIBer:IOR Command 14
3.3 SENSe:FIBer:IOR? Command 14
3.4 SENSe:FIBer:BSC Command 15
3.5 SENSe:FIBer:BSC? Command 15
3.6 SENSe:TRACE:READY? Command 15
4 Signal Generation – The SOURce Subsystem 16
4.1 SOURce:HOFFset Command 16
4.2 SOURce:HOFFset? Command 16
4.3 SOURce:VOFFset Command 16
4.4 SOURce:VOFFset? Command 17
4.5 SOURce:WAVelength Command 17
4.6 SOURce:WAVelength? Command 17
4.7 SOURce:ACURsor:POINt Command 17
4.8 SOURce:ACURsor:POINt? Command 18
4.9 SOURce:BCURsor:POINt Command 18
4.10 SOURce:BCURsor:POINt? Command 18
4.11 SOURce:PULSe:WIDTh Command 19
4.12 SOURce:PULSe:WIDTh? Command 19
4.13 SOURce:RANge:RESo Command 20
4.14 SOURce:RANge:RESo? Command 20
4.15 SOURce:RANge:RESo:ALL? Command 21
4.16 SOURce:WAVelength:AVAilable? Command 21
4.17 SOURce:LSALeft Command 22
4.18 SOURce:LSALeft? Command 22
4.19 SOURce:LSARight Command 22
4.20 SOURce:LSARight? Command 23
4.21 SOURce:Loss:Mode Command 23
4.22 SOURce:Loss:Mode? Command 23
4.23 SOURce:ANALyze:ON Command 24

4.24 SOURce:ANALyze:ON? Command 24
4.25 SOURce:CONTinuous:Laser:Fire Command 24
4.26 SOURce:CONTinuous:Laser:Fire? Command 25
4.27 SOURce:PARameters:CURRent:TRACE? Command 25
5 Data Calculations – CALCulate Subsystem 26
5.1 CALCulate:MATH:EXPRession:Loss? Command 26
5.2 CALCulate:MATH:EXPRession:EELoss? Command 26
6 File Operations – The MMEMory Subsystem 27
6.1 MMEMory:LOAD:T6? Command 27
6.2 MMEMory:LOAD:T5? Command 27
6.3 MMEMory:LOAD:SOR? Command 28
6.4 MMEMory:LOAD:MODule? Command 28
6.5 MMEMory:SAVE:File Command 29
6.6 MMEMory:LOAD:T6Text? Command 29
7 Display Operations 32
7.1 DISPLay:TAB Command 32
7.2 DISPLay:TAB? Command 32
7.3 DISPLay:DISTance:UNits Command 33
7.4 DISPLay:DISTance:Units? Command 33
7.5 DISPLay:Format Command 34
7.6 DISPLay:Format? Command 34
7.7 DISPLay:Full:Zoom Command (Deprecated) 34
7.8 DISPLay:Zoom:Full Command 35
7.9 DISPLay:Zoom:Horizontal Command 35
7.10 DISPLay:Zoom:Horizontal? Command 35
7.11 DISPLay:Zoom:Vertical Command 36
7.12 DISPLay:Zoom:Vertical? Command 36
7.13 DISPLay:Module:INFO? Command 36

1 Command Summary

The Command Summary section provides a list of the SCPI commands for the CMA5000 Standard OTDR application, along with a brief definition for each.

ABORt
Stop test.

INITiate
Start test. This command requires average count or time as parameter.

INITiate:AUTo
Start auto-test.

INITiate?
Queries if test is active.

SENSe:AVERages:COMPLeted?
Queries number of averages that have been completed since the start of the test.

SENSe:FIBer:IOR
Sets index of refraction. This value will be used for the next test.

SENSe:FIBer:IOR?
Queries index of refraction.

SENSe:FIBer:BSC
Sets backscatter coefficient. This value will be used for the next test.

SENSe:FIBer:BSC?
Queries backscatter coefficient.

SENSe:TRACE:READY?
Queries if trace data is ready.

SOURce:HOFFset
Set horizontal offset for the displayed trace(s).

SOURce:HOFFset?
Query horizontal offset for the displayed trace(s).

SOURce:VOFFset
Set vertical offset for the displayed trace(s).

SOURce:VOFFset?
Query vertical offset for the displayed trace(s).

SOURce:WAVelength
Set current wavelength.

SOURce:WAVelength?
Query current wavelength.

SOURce:AMARKer:POINt
Set A marker position.

SOURce:AMARKer:POINt?
Query current A marker position.

SOURce:BMARKer:POINt
Set B marker position.

SOURce:BMARKer:POINt?
Query current B marker position.

SOURce:PULSe:WIDTh
Set current pulse width.

SOURce:PULSe:WIDTh?
Query current pulse width.

SOURce:RANGEReso
Set current range and resolution.

SOURce:RANGEReso?
Query current range and resolution.

SOURce:RANge:RESo:ALL?
Query the entire range and resolution table.

SOURce:WAVelength:AVAilable?
Query the list of available wavelengths.

SOURce:LSALeft
Set left LSA value.

SOURce:LSALeft?

Query left LSA value.

SOURce:LSARight
Set right LSA value.

SOURce:LSARight?
Query right LSA value.

SOURce:Loss:Mode
Set current Loss Mode.

SOURce:Loss:Mode?
Query current Loss Mode.

SOURce:ANALyze:ON
Set if analysis to be performed after the test.

SOURce:ANALyze:ON?
Query if analysis is set on.

SOURce:CONTinuous:Laser:Fire
Set continuous laser fire.

SOURce:CONTinuous:Laser:Fire?
Check if continuous laser fire.

SOURce:PARameters:CURRent:TRACE?
Query current trace parameters.

CALCulate:MATH:EXPRession:Loss?
Calculate loss.

CALCulate:MATH:EXPRession:EELoss?
Calculate end-to-end loss.

MMEMory:LOAD:T5?
Get T5 trace object.

MMEMory:LOAD:T6?
Get T6 trace object.

MMEMory:LOAD:SOR?
Get SOR trace object.

MMEMory:LOAD:T6Text?
Load T6 trace text data.

MMEMory:LOAD:MODule?
Get module data.

MMEMory:SAVE:File
Save current file.

DISPLay:TAB
Set instrument GUI tab.
DISPLay:TAB?
Queries current instrument GUI tab.

DISPLay:DISTance:Units
Set current distance units.

DISPLay:DISTance:Units?
Query current distance units.

DISPLay:Format
Set display format state.

DISPLay:Format?
Query display format

DISPLay:Full:Zoom (Deprecated)
Render full zoom on the trace screen.

DISPLayl:Zoom:Full
Render full zoom on the trace screen.

DISPLay:Zoom:Horizontal
Set current horizontal zoom level.

DISPLay:Zoom:Horizontal?
Query current horizontal zoom level.

DISPLay:Zoom:Vertical
Set current vertical zoom level.

DISPLay:Zoom:Vertical?
Query current vertical zoom level.

DISPLay:Module:INFO?

Display model name and serial number of the OTDR module.

2 Root Level Commands

2.1 ABORt Command

Syntax:	ABORt
Parameters:	None
Description:	Aborts the active test.
Example:	`abor`
Response:	None
Errors:	-200, “std_execGen, State is already IDLE!”

2.2 INITiate Command

Syntax:	INITiate<wsp><number of averages>,<is timed averaging>
Parameters:	<number of averages> Integer format Range: If <is timed averaging> parameter is 0, then <number of averages> parameter should be 8 – 21. These values are power of 2 exponents. If <is timed averaging> parameter is 1, then <number of averages> parameter should be 5 – 5995 seconds. If <number of averages> is 0 (zero), then the real-time rest will be initiated. In this case, second parameter is still required but will not be parsed. <is timed averaging> Integer format Range: 0|1 0 = <number of averages> parameter is used as power of 2 exponent in this case. 1 = <number of averages> parameter is used as number of seconds for the timed scan.
Description:	Starts test. This command requires average count or time as parameter. This command is an overlapping command, except if the test if real-time (<number of averages> parameter is equal to zero).
Example:	`init 20,1`
Response:	None
Error:	-200, “std_execGen, Test is already active!” -224, “std_illegalParmValue, Parameters are out of range!”

2.3 INITiate AUTo Command

Syntax:	INITiate:AUTo
Parameters:	None
Description:	Stars auto-test. All test parameters are automatically obtained during the pre-scan. This command is an overlapping command.
Example:	`init:aut`
Response:	None
Errors:	-200, “std_execGen, Test is already active!”

2.4 INITiate? Command

Syntax:	INITiate?
Parameters:	None
Description:	Queries if test is active.
Example:	`Init? --- 0<END>`
Response:	Possible response range: 0|1 0 = Test is NOT active. 1 = Test is currently active.
Errors:	None

3 Sensing Commands

The SENSe subsystem lets you control measurement parameters.

3.1 SENSe:AVERages:COMPleted? Command

Syntax:	SENSe:AVERages:COMPleted?
Parameters:	None
Description:	Queries number of averages that have been completed since the start of the test. Number of averages is equal to 128 during the Real Time test.
Example:	`sens:aver:comp? --- 4096<END>`
Response:	Number of averages
Errors:	-200, “std_execGen, No primary trace!”

3.2 SENSe:FIBer:IOR Command

Syntax:	SENSe:FIBer:IOR<wsp><value>
Parameters:	<value> Floating point format Range: 1.300000 - 1.700000
Description:	Sets index of refraction. This value will be used for the next test.
Example:	`sens:fib:ior 1.45`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameter is out of range!”

3.3 SENSe:FIBer:IOR? Command

Syntax:	SENSe:FIBer:IOR?
Parameters:	None
Description:	Queries index of refraction.
Example:	`sens:fib:ior? --- 1.45<END>`
Response:	Possible response is value in range: 1.300000 - 1.700000
Errors:	None

3.4 SENSe:FIBer:BSC Command

Syntax:	SENSe:FIBer:BSC<wsp><value>
Parameters:	<value> Floating point format Range: -90.0 - -40.0
Description:	Sets backscatter coefficient. This value will be used for the next test.
Example:	`sens:fib:bsc -83.0`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameter is out of range!”

3.5 SENSe:FIBer:BSC? Command

Syntax:	SENSe:FIBer:BSC?
Parameters:	None
Description:	Queries backscatter coefficient.
Example:	`sens:fib:bsc? --- -83.0<END>`
Response:	Possible response is value in range: -90.0 - -40.0
Errors:	None

3.6 SENSe:TRACE:READY? Command

Syntax:	SENSe:TRACE:READY?
Parameters:	None
Description:	Queries if trace data is ready.
Example:	`sens:trace:ready? --- true<END>`
Response:	Possible response values: true = trace data is ready and can be transferred or saved. false = no trace data available in the memory.
Errors:	None

4 Signal Generation – The SOURce Subsystem

The SOURce subsystem allows controlling of the OTDR's optical source. It also controls of the cursors.

4.1 SOURce:HOFFset Command

Syntax:	SOURce:HOFFset<wsp><value>
Parameters:	<value> Floating point format Range: Offset value can be set plus/minus current distance range.
Description:	Set horizontal offset for the displayed trace(s). Offset value units – KM.
Example:	`sour:hoff 10.0`
Response:	None
Errors:	None

4.2 SOURce:HOFFset? Command

Syntax:	SOURce:HOFFset?
Parameters:	None
Description:	Queries horizontal offset for the displayed trace(s).
Example:	`sour:hoff? --- 0<END>`
Response:	Current horizontal offset value.
Errors:	None

4.3 SOURce:VOFFset Command

Syntax:	SOURce:VOFFset<wsp><value>
Parameters:	<value> Floating point format Range: Offset value can be set plus/minus current distance range.
Description:	Set vertical offset for the displayed trace(s). Offset value units – DB..
Example:	`sour:voff -5.0`
Response:	None
Errors:	None

4.4 SOURce:VOFFset? Command

Syntax:	SOURce:VOFFset?
Parameters:	None
Description:	Queries vertical offset for the displayed trace(s).
Example:	`sour:voff? --- 0<END>`
Response:	Current vertical offset value.
Errors:	None

4.5 SOURce:WAVelength Command

Syntax:	SOURce:WAVelength<wsp><value>
Parameters:	<value> Floating point format Range: Offset value can be set plus/minus current dynamic range.
Description:	Set current wavelength. Wavelength value units – NM. Wavelength will be set only if the new value matches the one in the list of available once.
Example:	`sour:wav 1310`
Response:	None
Errors:	-224, “std_illegalParmValue, Invalid parameter value!”

4.6 SOURce:WAVelength? Command

Syntax:	SOURce:WAVelength?
Parameters:	None
Description:	Queries current wavelength.
Example:	`sour:wav? --- 1310<wsp>nm<END>`
Response:	Current wavelength value.
Errors:	None

4.7 SOURce:ACURsor:POINt Command

Syntax:	SOURce: ACURsor:POINt<wsp><value>
Parameters:	<value> Floating point format Range: 0.0 - 273.8043.
Description:	Set A cursor position. Cursor position units – KM.
Example:	`sour:acur:poin 20.5`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameter is out of range!”

4.8 SOURce:ACURsor:POINt? Command

Syntax:	SOURce: ACURsor:POINt?
Parameters:	None.
Description:	Queries current A cursor position.
Example:	`sour:acur:poin? --- 20.5<END>`
Response:	Current A cursor position value.
Errors:	None

4.9 SOURce:BCURsor:POINt Command

Syntax:	SOURce: BCURsor:POINt<wsp><value>
Parameters:	<value> Floating point format Range: 0.0 - 273.8043.
Description:	Set B cursor position. Cursor position units – KM.
Example:	`sour:bcur:poin 20.5`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameter is out of range!”

4.10 SOURce:BCURsor:POINt? Command

Syntax:	SOURce: BCURsor:POINt?
Parameters:	None.
Description:	Queries current B cursor position.
Example:	`sour:bcur:poin? --- 20.5<END>`
Response:	Current B cursor position value.
Errors:	None

4.11 SOURce:PULSe:WIDTh Command

Syntax:	SOURce:PULSe:WIDTh<wsp>< pulse width >,<mode>
Parameters:	<pulse width> Integer format Range: 5 – 30000 nanoseconds (NS). Available pulse width values are dependant on current range/resolution. Pulse width value will be ignored if it doesn't mach any value in the list of available once. <mode> Integer format Range: 0 - 7 1 = Sets Long Haul to be on 2 = Sets Gain Splice to be on 4 = Sets Box Car Filter to be on To enable more than one of these options, use sum of their corresponding values. Setting Long Haul and Gain Splice for example would require <mode>=3 (1 + 2). Setting Box Car Filter and Long Haul would require <mode>=5 (4 + 1) etc..
Description:	Set current pulse width.
Example:	`sour:puls:widt 1000,4`
Response:	None
Errors:	-224, "std_illegalParmValue, Parameter is out of range!"

4.12 SOURce:PULSe:WIDTh? Command

Syntax:	SOURce:PULSe:WIDTh?
Parameters:	None
Description:	Query current pulse width.
Example:	`sour:puls:widt? --- 1000,4<END>`
Response:	First value is current pulse width in nanoseconds (NS), the second value is pulse width mode 0 – 7. 1 = Sets Long Haul to be on 2 = Sets Gain Splice to be on 4 = Sets Box Car Filter to be on For example 3 would mean Gain Splice and Long Haul (2 + 1).
Errors:	None

4.13 SOURce:RANge:RESo Command

Syntax:	SOURce:RANge:RESo<wsp><range>,<resolution>
Parameters:	<range> Integer format Range: 5 – 300 kilometers (KM). <resolution> Floating point format Range: 0.125 – 16.0 meters (M).
Description:	Set current range and resolution.
Example:	`sour:ran:res 1000,4`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameter is out of range!”

4.14 SOURce:RANge:RESo? Command

Syntax:	SOURce:RANge:RESo?
Parameters:	None
Description:	Query current range and resolution.
Example:	`sour:ran:res? --- 50,4.0 <END>`
Response:	First value is current range in kilometers (KM). Second value is current resolution in meters (M).
Errors:	None

4.15 SOURce:RANge:RESo:ALL? Command

Syntax:	SOURce:RANge:RESo:ALL?
Parameters:	None
Description:	Query the entire range and resolution table for every wavelength.
Example:	`sour:ran:res:all? ---` `1310.0,5.0,0.125,1310.0,5.0,0.5,1310.0,5.0,2.0,1310.0,20.0,0.125,1310.0,20.0,1.0,1310.0,20.0,4.0,1310.0,50.0,0.25,1310.0,50.0,1.0,1310.0,50.0,4.0,1310.0,75.0,0.5,1310.0,75.0,2.0,1310.0,75.0,8.0,1310.0,125.0,0.5,1310.0,125.0,2.0,1310.0,125.0,8.0,1310.0,250.0,1.0,1310.0,250.0,4.0,1310.0,250.0,16.0,1310.0,300.0,2.0,1310.0,300.0,4.0,1310.0,300.0,16.0,1550.0,5.0,0.125,1550.0,5.0,0.5,1550.0,5.0,2.0,1550.0,20.0,0.125,1550.0,20.0,1.0,1550.0,20.0,4.0,1550.0,50.0,0.25,1550.0,50.0,1.0,1550.0,50.0,4.0,1550.0,75.0,0.5,1550.0,75.0,2.0,1550.0,75.0,8.0,1550.0,125.0,0.5,1550.0,125.0,2.0,1550.0,125.0,8.0,1550.0,250.0,1.0,1550.0,250.0,4.0,1550.0,250.0,16.0,1550.0,300.0,2.0,1550.0,300.0,4.0,1550.0,300.0,16.0,1625.0,5.0,0.125,1625.0,5.0,0.5,1625.0,5.0,2.0,1625.0,20.0,0.125,1625.0,20.0,1.0,1625.0,20.0,4.0,1625.0,50.0,0.25,1625.0,50.0,1.0,1625.0,50.0,4.0,1625.0,75.0,0.5,1625.0,75.0,2.0,1625.0,75.0,8.0,1625.0,125.0,0.5,1625.0,125.0,2.0,1625.0,125.0,8.0,1625.0,250.0,1.0,1625.0,250.0,4.0,1625.0,250.0,16.0,1625.0,300.0,2.0,1625.0,300.0,4.0,1625.0,300.0,16.0 <END>`
Response:	Comma separated floating point values. Wavelength in nanometers (NM) followed by range in kilometers (KM), followed by resolution in meters (M).
Errors:	None

4.16 SOURce:WAVelength:AVAilable? Command

Syntax:	SOURce:WAVelength:AVAilable?
Parameters:	None
Description:	Queries the list of available wavelengths.
Example:	`sour:wav:ava? --- 1310,1550,<END>`
Response:	List of available wavelength in nanometers (NM).
Errors:	None

4.17 SOURce:LSALeft Command

Syntax:	SOURce:LSALeft<wsp><value>
Parameters:	<left> Floating point format Range: -273.8043 - 273.8043. <right> Floating point format Range: -273.8043 - 273.8043.
Description:	Set start and stop for left LSA marker. Start and stop units – kilometers (KM).
Example:	`sour:lsal 0.0,0.5`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameters are out of range!”

4.18 SOURce:LSALeft? Command

Syntax:	SOURceLSALeft?
Parameters:	None
Description:	Query left LSA value.
Example:	`sour:lsal? --- 0.0,0.5<END>`
Response:	Start and stop for left LSA marker. Start and stop units – kilometers (KM).
Errors:	None

4.19 SOURce:LSARight Command

Syntax:	SOURce:LSARight<wsp><value>
Parameters:	<left> Floating point format Range: -273.8043 - 273.8043. <right> Floating point format Range: -273.8043 - 273.8043.
Description:	Set start and stop for right LSA marker. Start and stop units – kilometers (KM).
Example:	`sour:lsar 0.0,0.5`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameters are out of range!”

4.20 SOURce:LSARight? Command

Syntax:	SOURceLSARight?
Parameters:	None
Description:	Query right LSA value.
Example:	`sour:lsar? --- 0.0,0.5<END>`
Response:	Start and stop for right LSA marker. Start and stop units – kilometers (KM).
Errors:	None

4.21 SOURce:Loss:Mode Command

Syntax:	SOURce:Loss:Mode<wsp><mode>
Parameters:	<mode> Integer format Range: 0|1|2|3|4|5|6 0 = Splice Loss Mode 1 = Two-point Loss Mode 2 = Two-point LSA 3 = dB/km Loss Mode 4 = dB/km LSA Loss Mode 5 = ORL Loss Mode 6 = Two-point Attenuation Corrected Loss Mode
Description:	Set current Loss Mode.
Example:	`sour:l:m 0`
Response:	None
Errors:	-224, “std_illegalParmValue, Parameter is out of range!

4.22 SOURce:Loss:Mode? Command

Syntax:	SOURce:Loss:Mode?
Parameters:	none
Description:	Query current Loss Mode.
Example:	`sour:l:m? --- 0<END>`
Response:	Possible responses are: 0 = Splice Loss Mode 1 = Two-point Loss Mode 2 = Two-point LSA 3 = dB/km Loss Mode 4 = dB/km LSA Loss Mode 5 = ORL Loss Mode 6 = Two-point Attenuation Corrected Loss Mode
Errors:	None

4.23 SOURce:ANALyze:ON Command

Syntax:	SOURce:ANALyze:ON<wsp><value>
Parameters:	<value> Integer format Range: 0|1 0 = Auto-analysis is OFF. 1 = Auto-analysis is ON.
Description:	Set if analysis to be performed automatically after the test.
Example:	`sour:anal:on 0`
Response:	None
Errors:	None

4.24 SOURce:ANALyze:ON? Command

Syntax:	SOURce:ANALyze:ON?
Parameters:	None
Description:	Query if auto-analysis is set on. Trace is automatically analyzed after test completion.
Example:	`sour:anal:on? --- 0<END>`
Response:	Possible responses are: 0 = Auto-analysis is OFF. 1 = Auto-analysis is ON.
Errors:	None

4.25 SOURce:CONTinuous:Laser:Fire Command

Syntax:	SOURce:CONTinuous:Laser:Fire<wsp><value>
Parameters:	<value> Integer format Range: 0|1 0 = Laser does not fire continuously. 1 = Laser fires continuously.
Description:	Set continuous laser fire.
Example:	`sour:cont:l:f 0`
Response:	None
Errors:	None

4.26 SOURce:CONTinuous:Laser:Fire? Command

Syntax:	SOURce:CONTinuous:Laser:Fire?
Parameters:	None
Description:	Check if continuous laser fire.
Example:	`sour:cont:l:f? --- 0<END>`
Response:	Possible responses are: 0 = Laser does not fire continuously. 1 = Laser fires continuously.
Errors:	None

4.27 SOURce:PARameters:CURRent:TRACE? Command

Syntax:	SOURce:PARameters:CURRent:TRACE?
Parameters:	None
Description:	Query current trace parameters.
Example:	`sour:par:curr:trace?---50.0,4.0,200,true,1550,5120,1.5,-82.111111,0.0,0.0,CMA5000,,2001-1343,Other,T6,,0.00,0.00,0.00<END>`
Response:	<range> = Integer format. Distance range in kilometers. <reso> = Floating point format. Resolution in meters. <pw> = Integer format. Pulse width in nanoseconds. <hiRes> = Boolean format. Is pulse high resolution. Boolean value. “true” – high resolution, “false” – long haul. <wl> = Integer format. Nominal wavelength in nanometers. <avg> = Integer format. Number of averages completed. <ior> = Floating point format. Index of refraction. <bsc> = Floating point format. Backscatter coefficient. <hShift> = Floating point format. Horizontal shift in kilometers. <vShift> = Floating point format. Vertical shift in decibels. <prodType> = String format. Product type. <optMod> = String format. Optical module type. <optModSN> = String format. Optical module serial number. <fibType> = String format. Fiber type. <trcType> = String format. Trace type. <trcFlags> = String format. Trace flags. <fasLossThresh> = Floating point format. FAS loss threshold in decibels. <fasReflThresh> = Floating point format. FAS reflectance threshold in decibels. <fasFibBreakThresh> = Floating point format. FAS fiber break threshold in decibels.
Errors:	-200, “std_execGen, No primary trace!”

5 Data Calculations – CALCulate Subsystem

The CALCulate subsystem allows the executing of loss calculation expressions.

5.1 CALCulate:MATH:EXPRession:Loss? Command

Syntax:	CALCulate:MATH:EXPRession:Loss?
Parameters:	None
Description:	Calculate loss. The calculated value is depends on current loss mode.
Example:	`calc:math:expr:l? --- -4.610<END>`
Response:	Calculated loss value.
Errors:	None

5.2 CALCulate:MATH:EXPRession:EELoss? Command

Syntax:	CALCulate:MATH:EXPRession:EELoss?
Parameters:	None
Description:	Calculate end-to-end loss. The calculated value is depends on current loss mode.
Example:	`calc:math:expr:eel? --- -4.610<END>`
Response:	Calculated end-to-end loss value.
Errors:	None

6 File Operations – The MMEMory Subsystem

The MMEMory subsystem provides access to the currently loaded file.

6.1 MMEMory:LOAD:T6? Command

Syntax:	MMEMory:LOAD:T6?
Parameters:	None
Description:	Get T6 trace object.
Example:	`mmem:load:t6? --- "BINARY ARRAY"<END>`
Response:	T6 as array of bytes. Following is an example of first few bytes of the "BINARY ARRAY" response: #43255… # – First byte to signify that the binary data is coming 4 – Second byte is a digit and the digit value indicates the number of size bytes that follow. 3255 – In this example, the next 4 bytes make number 3255 which is the number of data bytes to be received. … – Actual data bytes that need to be received and stores in a file.
Errors:	-200, "std_execGen, Test is active!" -200, "std_execGen, No primary trace!"

6.2 MMEMory:LOAD:T5? Command

Syntax:	MMEMory:LOAD:T5?
Parameters:	None
Description:	Get T5 trace object.
Example:	`mmem:load:t5? --- "BINARY ARRAY"<END>`
Response:	T5 as array of bytes. Following is an example of first few bytes of the "BINARY ARRAY" response: #43255… # – First byte to signify that the binary data is coming 4 – Second byte is a digit and the digit value indicates the number of size bytes that follow. 3255 – In this example, the next 4 bytes make number 3255 which is the number of data bytes to be received. … – Actual data bytes that need to be received and stores in a file.
Errors:	-200, "std_execGen, Test is active!" -200, "std_execGen, No primary trace!"

6.3 MMEMory:LOAD:SOR? Command

Syntax:	MMEMory:LOAD:SOR?
Parameters:	None
Description:	Get SOR trace object.
Example:	`mmem:load:sor? --- “BINARY ARRAY”<END>`
Response:	SOR as array of bytes. Following is an example of first few bytes of the “BINARY ARRAY” response: #43255… # – First byte to signify that the binary data is coming 4 – Second byte is a digit and the digit value indicates the number of size bytes that follow. 3255 – In this example, the next 4 bytes make number 3255 which is the number of data bytes to be received. … – Actual data bytes that need to be received and stores in a file.
Errors:	-200, “std_execGen, Test is active!” -200, “std_execGen, No primary trace!”

6.4 MMEMory:LOAD:MODule? Command

Syntax:	MMEMory:LOAD:MODule?
Parameters:	None
Description:	Get module data.
Example:	`mmem:load:mod? --- “BINARY ARRAY”<END>`
Response:	Module data as array of bytes. Following is an example of first few bytes of the “BINARY ARRAY” response: #43255… # – First byte to signify that the binary data is coming 4 – Second byte is a digit and the digit value indicates the number of size bytes that follow. 3255 – In this example, the next 4 bytes make number 3255 which is the number of data bytes to be received. … – Actual data bytes that need to be received and stores in a file.
Errors:	None

6.5 MMEMory:SAVE:File Command

Syntax:	MMEMory:SAVE:File<wsp><path><wsp><name>
Parameters:	<path> String format The parameter <path> is a legal, full Windows path to the directory which is to be set as the directory to write the file to. <path> consists of a legal Windows drive designator followed by an optional sequence of Windows directories separated by backslashes (i.e. '\'). For example a <path> consisting only of drive designator would have the form: d:\ - This <path> points to the root directory of a system's d: drive. More extensive examples of <path>'s are: d:\traces d:\traces\data\january <name> String format Filename must be in Windows compatible format.
Description:	Save current file. Both parameters have to be quoted strings.
Example:	`Mmem:save:f "d:\\mypath" "myname.trc"`
Response:	None
Errors:	-200, "std_execGen, Path does not exist!" -200, "std_execGen, Filename does already exist!" -200, "std_execGen, Error while saving file!" -200, "std_execGen, No primary trace or test is active!"

6.6 MMEMory:LOAD:T6Text? Command

Syntax:	MMEMory:LOAD:T6Text?
Parameters:	None
Description:	Get T6 trace text data.

Example:	`mmem:load:t6t? --- "BINARY ARRAY"<END>`
	
	`Following is an example of first few bytes of the "BINARY ARRAY" response: #43255…`
	
	`# – First byte to signify that the binary data is coming`
	`4 – Second byte is a digit and the digit value indicates the number of size bytes that follow.`
	`3255 – In this example, the next 4 bytes make number 3255 which is the number of data bytes to be received.`
	`… – Actual data bytes that need to be received and stores in a file.`
	
	`Here is also the example of the data bytes content when saved as the text file:`

```
   20              "// Number of header lines, including this line."
"T6TrcText - Version 04/12/02"
FN   = "AUTO13100000.SOR"
PN   = "AUTO13100000.SOR"
TYPE = "Anr SR4731"
INST = "MT9083B-053"
OPTC = ""
WL   = 1310 nm
PW   = 200 ns
HRLH = "[H]"
FBR  = "Singlemode"
AVG  = 1024
IOR  = 1.467700
BSC  = -78.50 dB
DATE = "11/06/08"
TIME = "10:23 AM"
MXDB = 64 dB
RESO = 1.001 m
DX   = 1.02200707368672 m
PTS  = 25001
 1000            "// Scale Factor."
 5783
46324
46563
46876
...
29365
29365
29365
Events  = 1
Type                = E
Location            = 4.4519 km
Loss                = >3.00 dB
Event-Event Loss    = 1.57 dB
Event-Event Loss/km = 0.353 dB
Reflectance         = -38.43  dB

Created text file contains the following lines :

   {Number of lines in the header}             "// Number of header
lines, including this line."
"T6TrcText - Version {Trace text utility version}"
FN   = "{Trace filename}"
```

	PN = "{Trace file pathname+filename}" TYPE = "{Trace type e.g. T5, T6, SR4731}" INST = "{Instrument}" OPTC = "{Optics module}" WL = {Wavelength in nm} nm PW = {PulseWidth in nsec} ns HRLH = "{HiRes/Long Haul preamp, [H]/[L]}" FBR = "{Fiber Type e.g. Singlemode or Multimode}" AVG = {Number of Averages} IOR = {IOR value} BSC = {Backscatter Coefficient in dB} dB DATE = "{Date}" TIME = "{Time}" MXDB = {Maximum display range in dB} dB RESO = {Nominal Resolution in meters} m DX = {Distance between data points in meters} m PTS = {Number of Data Points} {Scale Factor} "// Scale Factor." {Data Point Value as Integer Number. The formula for calculating the value as displayed on the screen is ((Data Point Value / Scale Factor) - MXDB)} ... {Data Point Value as an Integer Number} Events = {Number of Events} Type = {Event Type e.g. R = Reflective, N = Non-reflective, G = Grouped, E = End, ? = Questionable End} Location = {Event Location in current distance units xx} xx Loss = {Loss Value} dB Event-Event Loss = {Event to Event Loss Value} dB Event-Event Loss/km = {Event to Event Loss Value per kilometer} dB Reflectance = {Reflectance Value} dB
Response:	T6 trace text data as array of bytes.
Errors:	-200, “std_execGen, Test is active!” -200, “std_execGen, No primary trace!” -200, “std_execGen, Cannot get text data!”

7 Display Operations

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

7.1 DISPLay:TAB Command

Syntax:	DISPLay:TAB<wsp><value>
Parameters:	<value> Integer format Range: 0|1|2|3 0 = Display Trace tab 1 = Display Analysis tab 2 = Display File Manager tab 3 = Display Help tab
Description:	Switches particular instrument tab pane to be the foreground one.
Example:	`disp:tab 0`
Response:	None
Errors:	-224, "std_illegalParmValue, Parameter is out of range!"

7.2 DISPLay:TAB? Command

Syntax:	DISPLay:TAB?
Parameters:	None
Description:	Queries current foreground tab.
Example:	`displ:tab? --- 0<END>`
Response:	Possible responses are: 0 = Display Trace tab 1 = Display Analysis tab 2 = Display File Manager tab 3 = Display Help tab
Errors:	None

7.3 DISPLay:DISTance:UNits Command

Syntax:	DISPLay:DISTance:Units<wsp><value>
Parameters:	<value> Integer format Range: 0|1|2|3|4 0 = Sets units to miles (mi) 1 = Sets units to feet (ft) 2 = Sets units to kilofeet (kft) 3 = Sets units to miters (m) 4 = Sets units to kilometers (km)
Description:	Set the units used by instrument. All length measurements will be given in the units set by this command.
Example:	`dipl:dist:un 0`
Response:	None
Errors:	-224, "std_illegalParmValue, Parameter is out of range!"

7.4 DISPLay:DISTance:Units? Command

Syntax:	DISPLay:DISTance:UNits?
Parameters:	None
Description:	Queries current distance units.
Example:	`displ:dist:un? --- 0<END>`
Response:	Possible responses are: 0 = Units in miles (mi) 1 = Units in feet (ft) 2 = Units in kilofeet (kft) 3 = Units in miters (m) 4 = Units in kilometers (km)
Errors:	None

7.5 DISPLay:Format Command

Syntax:	DISPLay:Format<wsp><value>
Parameters:	<value> Integer format Range: 0|1|2|3 0 = display from Origin 1 = display from Cursor A 2 = display from cursor B 3 = display from Anywhere
Description:	Allows set up of the display format.
Example:	`displ:f 0`
Response:	None
Errors:	-224, "std_illegalParmValue, Parameter is out of range!"

7.6 DISPLay:Format? Command

Syntax:	DISPLay:Format?
Parameters:	None
Description:	Queries current display format.
Example:	`displ:f? --- 0<END>`
Response:	Possible responses are: 0 = display from Origin 1 = display from Cursor A 2 = display from cursor B 3 = display from Anywhere
Errors:	None

7.7 DISPLay:Full:Zoom Command (Deprecated)

Syntax:	DISPLay:Full:Zoom
Parameters:	None
Description:	Renders full zoom at display from anywhere OTDR trace screen. This command is deprecated do to its name. Please use DISPLay:Zoom:Full instead.
Example:	`displ:f:z`
Response:	None
Errors:	None

7.8 DISPLay:Zoom:Full Command

Syntax:	DISPLay:Zoom:Full
Parameters:	None
Description:	Renders full zoom at display from anywhere OTDR trace screen. For every combination of view (A, B, Origin, Anywhere) / tab (Trace, Analysis), there is separate zoom level.
Example:	`displ:z:f`
Response:	None
Errors:	None

7.9 DISPLay:Zoom:Horizontal Command

Syntax:	DISPLay:Zoom:Horizontal<wsp><value>
Parameters:	<value> Integer format Range: 0 – 13 The parameter range is variable and depends on current range/resolution settings with maximum value from 9 to 13.
Description:	Sets horizontal zoom level as number of expansions (number of right key presses from full zoom). For every combination of view (A, B, Origin, Anywhere) / tab (Trace, Analysis), there is separate zoom level.
Example:	`displ:z:h 0`
Response:	None
Errors:	-224, "std_illegalParmValue, Parameter is out of range!" -200, "std_execGen, Invalid Tab Selected!"

7.10 DISPLay:Zoom:Horizontal? Command

Syntax:	DISPLay: Zoom:Horizontal?
Parameters:	None
Description:	Queries current horizontal zoom level. For every combination of view (A, B, Origin, Anywhere) / tab (Trace, Analysis), there is separate zoom level.
Example:	`displ:z:h? --- 0<END>`
Response:	Value form 0 to 13.
Errors:	None

7.11 DISPLay:Zoom:Vertical Command

Syntax:	DISPLay:Zoom:Vertical<wsp><value>
Parameters:	<value> Integer format Range: 0 – 13 The parameter range is variable and depends on current range/resolution settings with maximum value from 9 to 13.
Description:	Sets vertical zoom level as number of expansions (number of right key presses from full zoom). For every combination of view (A, B, Origin, Anywhere) / tab (Trace, Analysis), there is separate zoom level.
Example:	`displ:z:v 0`
Response:	None
Errors:	-224, "std_illegalParmValue, Parameter is out of range!" -200, "std_execGen, Invalid Tab Selected!"

7.12 DISPLay:Zoom:Vertical? Command

Syntax:	DISPLay: Zoom:Vertical?
Parameters:	None
Description:	Queries current vertical zoom level. For every combination of view (A, B, Origin, Anywhere) / tab (Trace, Analysis), there is separate zoom level.
Example:	`displ:z:v? --- 0<END>`
Response:	Value form 0 to 13.
Errors:	None

7.13 DISPLay:Module:INFO? Command

Syntax:	DISPLay:MODule:INFO?
Parameters:	None
Description:	Displays model name and serial number of the OTDR module.
Example:	`displ:mod:info? --- 5236-111-OTDR-1,N/A`
Response:	Two comma separated strings. If either of those are invalid, N/A will be displayed.
Errors:	None