MT1000A Network Master Pro
OTDR Modules
Remote Scripting Operation Manual

Second Edition

ANRITSU CORPORATION

Document No.: M-W3859AE-2.0

- This is an addendum to “Network Master Pro Operation Manual”.
- For safety and warning information, please read “MT1000A Network Master Pro OTDR Modules Operation Manual”(M-W3810AE) before attempting to use the equipment.
- Keep this manual with the equipment.

Notes On Export Management

This product and its manuals may require an Export License/Approval by the Government of the product’s country of origin for re-export from your country. Before re-exporting the product or manuals, please contact us to confirm whether they are export-controlled items or not. When you dispose of export-controlled items, the products/manuals need to be broken/shredded so as not to be unlawfully used for military purpose.

June 13, 2016 (First Edition)
September 2, 2016 (Second Edition)

Copyright ©2016 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.

About This Manual

This operation manual describes the SCPI (Standard Commands for Programmable Instruments) commands for Network Master Pro OTDR modules.

Note: SCPI commands described in this manual are supported in Network Master Pro version 7.00.

Some commands or queries in this manual may require that specific hardware or software options are installed. These options must be purchased separately.

This operation manual uses the notations described in the following standards:

- IEEE: Std 488.2-1992
- SCPI: VERSION 1999.0 (SCPI Consortium)

Contents

List of Figures

Chapter 1

Overview

The Network Master command based remote control functions support the built-in Ethernet service interface. Software specifications are in conformity with the IEEE488.2 standard based on SCPI version 1999 (Standard Commands for Programmable Instruments). Network Master becomes an automated measurement instrument when it is connected to an external controller.

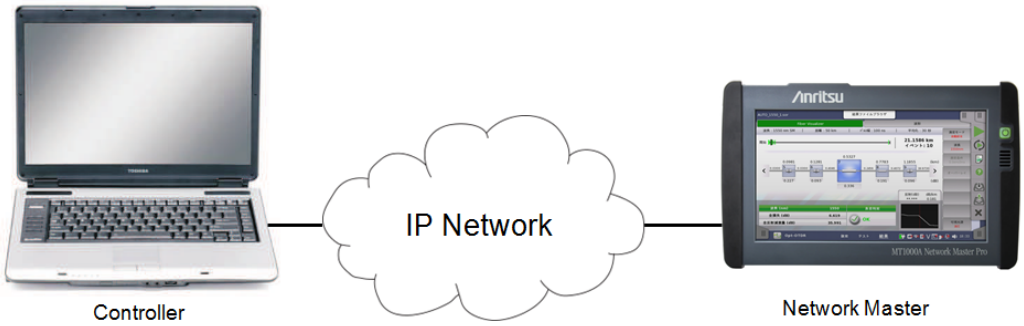

Figure 1.1: System setup using Ethernet

1.1 Ethernet Based Remote Control

1.1.1 Connecting Cable

To use remote control via the Ethernet service interface, connect an Ethernet cable to the Ethernet connector next to the power socket.

Figure 1.2: Connector panel

1.1.2 Ethernet Remote Control Settings

Port Number

To change a TCP port number (for a valid range, see Table 1.1) type the number in the **TCP Port** field (see Figure 1.3).

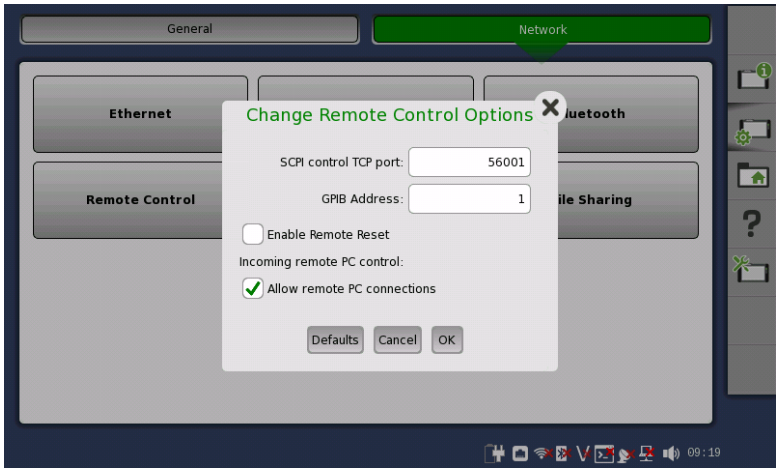

Figure 1.3: Configure TCP **Port** for Remote Control

Setup item	Description	Allowable range
Port Number	TCP Port Number	1024 to 65535 (default: 56001)

Table 1.1: Allowable TCP port range

1.1.3 Communication Buffers

The input- and output streams are buffered. Besides the TCP receive buffer (87380 bytes) and the TCP transmit buffer (16384 bytes), the two streams share a common command/response buffer of 32 entries. Each buffer entry can hold a compound program message of maximum 4 KB or a response message of maximum 64 KB.

Program data transferred as <ARBITRARY BLOCK PROGRAM DATA> does not go through the internal buffer, but is streamed directly from the TCP receive buffer to the internal file system. Similar for response data of type <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>; it is streamed directly from the internal file system to the TCP transmit buffer.

1.2 Program Messages

Program messages are the remote commands sent to Network Master as shown in Figure 1.4.

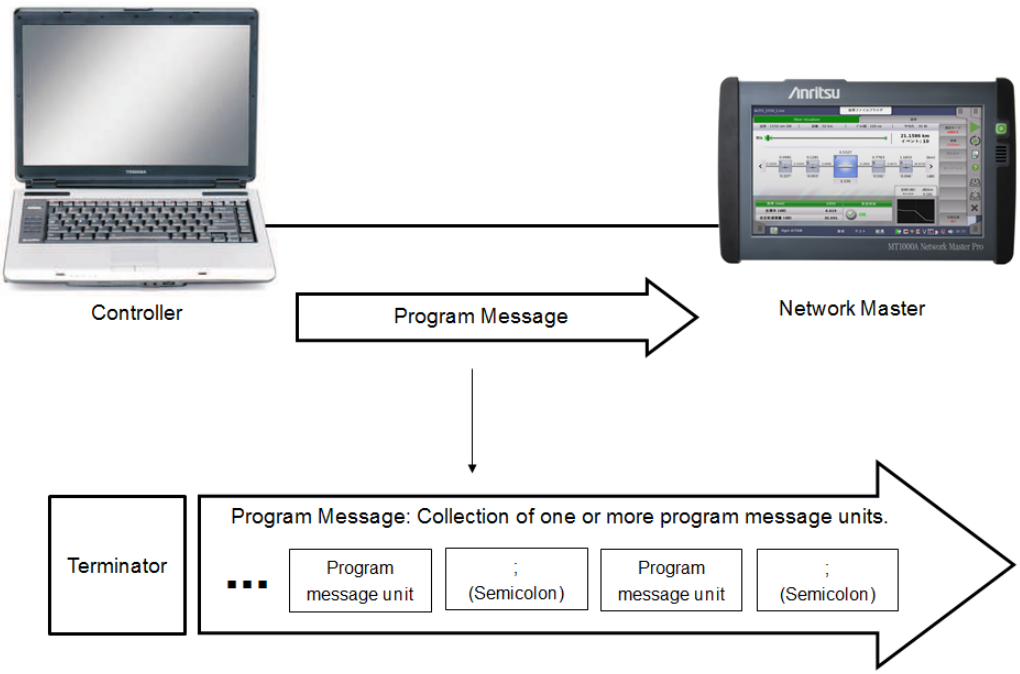

Figure 1.4: Program message structure

A program message consists of one or more program message units separated with a semicolon (;). Space(s) before or after a semicolon is ignored (space has no meaning). For more information on program message units, see section 1.2.1 Program Message Unit.

When a program message is sent to Network Master, a terminator is appended after it. Network Master receives the program message by detecting the terminator. For a description of the terminator, see section 1.2.4 Program Message Terminator.

The Network Master is able to handle program messages with a maximum length of 4096 characters including the message terminator.

1.2.1 Program Message Unit

A program message unit consists of a program header and a program data, see Figure 1.5.

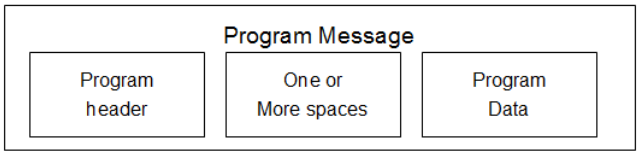

Figure 1.5: Program message unit

There must be one or more spaces between a program header and a program data. Network Master recognizes the program header and program data using the space(s). One or more spaces before a program header are ignored.

1.2.2 Program Headers

The program header specifies the function of the command message unit sent from the controller to Network Master. There are two types of program headers:

- Program headers for command message units.
- Program headers for query message units. Similar to headers for command message unit, but are always followed by a question mark "?".

The Network Master supports some of the common commands defined in the IEEE488.2 standard. These common commands are special in the way that they are always preceded by an asterisk "*"; e.g. `*IDN?`. All other commands are referred to as "device specific commands". Device specific commands consists of two or more <program mnemonic>'s (hereinafter called "mnemonic") separated with a colon "(:)".

[:]<program mnemonic>[:<program mnemonic>]... e.g. `SYSTem:TIME`

A mnemonic is a character string, which consists of capital and small letters. The capital part of the mnemonic is also referred to as the short form of the mnemonic.

- Long form program header: `INSTrument:STARt`
- Short form program header: `INST:STAR`

The Network Master recognizes a mnemonic even if only the short form is sent. For example, mnemonic `SYSTem` is recognized as a normal mnemonic when `SYST` is sent.

In this way, capital and small letters are used for recognizing long and short forms of a mnemonic, The Network Master does not distinguish between capital and small letters when reading the program header. However, the Network Master only accepts the short form or the complete long form of a mnemonic. Hence `SYSTe` is **not** a valid mnemonic. The following program headers are all acceptable and assumed to be the same:

- `SYSTEM:POWER:SOURCE?`
- `system:power:source?`
- `SySteM:PoWeR:SoUr?`
- `syst:POW:sour?`

1.2.3 Program Data

Program data is sent following the program header as parameters specified in the command message unit. This operation manual uses the notations given below in Table 1.2 for indicating the program data format. Most of them are defined in the IEEE488.2 standard.

Program data type	Description
<BOOLEAN PROGRAM DATA>	Defined in IEEE488.2 Indicates On/Off, Enable/Disable, or Yes/No. To specify On/Enable state, set {ON|1}. To specify Off/Disable state, set {OFF|0}.
<NUMERIC PROGRAM DATA>	Comprises <DECIMAL NUMERIC PROGRAM DATA> and <NON-DECIMAL NUMERIC PROGRAM DATA> as defined in IEEE488.2 The Network Master accepts both decimal and non-decimal entries for the <NUMERIC PROGRAM DATA>.
<DECIMAL NUMERIC PROGRAM DATA>	Defined in IEEE488.2 Comprises <NR1>,<NR2> and <NR3> decimal values, where <NR1> indicates an integer value. <NR2> indicates a numeric value in fixed point format. <NR3> indicates a numeric value in floating point format. Examples: <NR1>: 123 <NR2>: -123.456 <NR3>: 1.23E-3
<NONDECIMAL NUMERIC PROGRAM DATA>	Defined in IEEE488.2 Comprises <HEXADECIMAL>, <OCTAL> or <BINARY> program data. See below for further details.

<HEXADECIMAL>	Conforms to the hexadecimal format defined in IEEE488.2 as follows:
	`#{H|h}{A|a|B|b|C|c|D|d|E|e|F|f|<digit>}...`
	<digit> is an ASCII character with a value in the range of 0x30 to 0x39 (48 to 57 in decimal), that is, a numeric 0 to 9.
	Examples:
	`#h1234ABCD`
	`#Hfe1a9`

<OCTAL>	Conforms to the octal format defined in IEEE488.2 as follows:
	`#{Q|q}{0|1|2|3|4|5|6|7}...`
	Examples:
	`#q12345670`
	`#Q77`

<BINARY>	Conforms to the binary format defined in IEEE488.2 as follows:
	`#{B|b}{0|1}...`
	Examples:
	`#b10101010`
	`#B110`

<STRING PROGRAM DATA>	Defined in IEEE488.2
	A character string in a pair of single quotation marks (') or double quotation marks (").
	Examples:
	`"Network Master"`
	`'Testing the network'`

continued on next page...

...continued from previous page

Program data type	Description
<CHARACTER PROGRAM DATA>	Defined in IEEE488.2 Indicates two or more mnemonics for selections. Like program header mnemonics, <CHARACTER PROGRAM DATA> mnemonics can have a short and a long form. The syntax used in the Network Master additionally allows a digit as the first character of a mnemonic and also allows a dash (-) inside a mnemonic.

Table 1.2: Acceptable program data

1.2.4 Program Message Terminator

A program message terminator indicates the end of the program message. Upon reception of a terminator, the Network Master assumes that the program message is complete and starts processing the message. A terminator must always be added to the end of a program message. For Network Master the program message terminator is:

[<WHITE SPACE>]{NL} for Ethernet based remote control

<WHITE SPACE> is one or more ASCII characters with a value in the range of 0x00 to 0x09 or 0x0B to 0x20 (0 to 9 or 11 to 32 in decimal). These ranges include the ASCII control characters and space, except NL (newline). Since <WHITE SPACE> includes CR (0x0D) (13 in decimal), {CR}{NL} is also interpreted as a terminator by Network Master in Ethernet based remote control - to keep compatibility with conventional models.

1.2.5 Compound Program Messages

Compound headers are supported by the Network Master. Examples of the use of the compound headers are shown below.
The three program message units:

```
SYSTem:TIME?
SYSTem:DATE?
SYSTem:POWer:SOURce?
```

can be combined in one program message as follows:

```
SYSTem:TIME?; :SYSTem:DATE?; :SYSTem:POWer:SOURce?
```

or just:

```
SYSTem:TIME?; DATE?; POWer:SOURce?
```

(SYST: mnemonic can be omitted in the second and third program data units)
For further information on compound headers, see Appendix A of the IEEE488.2 standard.

1.2.6 Sequential Execution

The Network Master processes one program message unit at a time and in the same order in which they are arranged within the program message. The Network Master will not start processing a new program message until the processing of the current program message is finished.

1.3 Response Messages

Response messages are messages sent from a Network Master to a controller as reply to queries, see Figure 1.6.

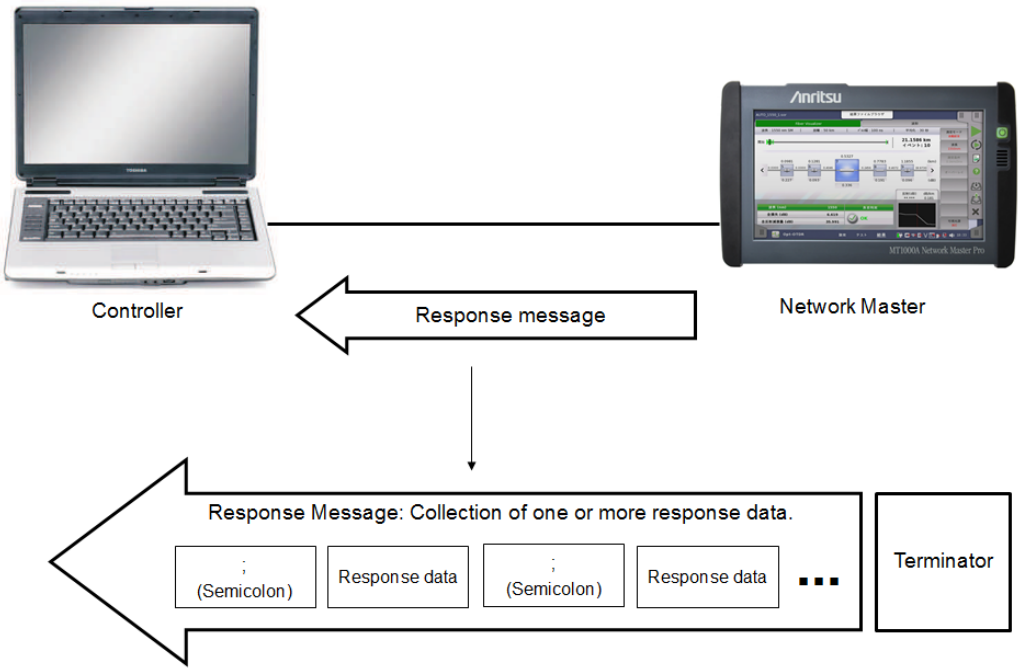

Figure 1.6: Response message structure

A response message consists of one or more response data separated with a semicolon (;). The response message is terminated with the response message terminator.

1.3.1 Response Data

Response data is a data returned by Network Master as reply to a query received from the controller. Table 1.3 shows examples of the response data format used in this manual.

Response data type	Description
<BOOLEAN RESPONSE DATA>	Defined in SCPI-99 Indicates On/Off, Enable/Disable, or Yes/No. When "1" is returned, it indicates an On/Enable state. When "0" is returned, it indicates an Off/Disable state.

<NR1 NUMERIC RESPONSE DATA>	Defined in IEEE488.2
	Indicates an decimal integer value.
	Examples:
	`123`
	`-500`

<NR2 NUMERIC RESPONSE DATA>	Defined in IEEE488.2
	Indicates a numeric value in fixed point format.
	Examples:
	`123.45`
	`-500.0`

continued on next page...

. . . continued from previous page

Response data type	Description

<NR3 NUMERIC RESPONSE DATA>	Defined in IEEE488.2
	Indicates a numeric value in floating point format.
	Examples:
	`1.23E3`
	`-5.67E-4`

<HEXADECIMAL NUMERIC RESPONSE DATA>	Conforms to the hexadecimal format defined in IEEE488.2 as follows:
	`#H{A|B|C|D|E|F|<digit>}...`
	<digit> is an ASCII character with a value in the range of 0x30 to 0x39 (48 to 57 in decimal), a numeric 0 to 9.
	Example:
	`#H0011EEFF`

<BINARY NUMERIC RESPONSE DATA>	Conforms to the binary format defined in IEEE488.2 as follows:
	`#B{0|1}...`
	Example:
	`#B10101010`

<STRING RESPONSE DATA>	Defined in IEEE488.2 A character string enclosed in a pair of double quotation marks ("). Example: `"Network Master - Testing the network."`
<CHARACTER RESPONSE DATA>	Defined in IEEE488.2 Indicates two or more mnemonics for selections. Like program header mnemonics, <CHARACTER RESPONSE DATA> mnemonics can have a short and a long form. The Network Master always returns the short form. The syntax used in the Network Master additionally allows a digit as the first character of a mnemonic and also allows a dash (-) inside a mnemonic.
<EXPRESSION RESPONSE DATA>	Defined in IEEE488.2 A Network Master-defined set of <RESPONSE DATA> elements separated by a comma (,) and enclosed by a set of parenthesis. Example: `(2,0.5),(3,0.25),(4,1.75)` For further details refer to the detailed description of the Network Master specific commands.
<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>	Defined in IEEE488.2 This response data type is used when the instrument streams binary data (typically PDF files) to the controller. It is defined as #<nonzero digit><digits><8 bit data bytes> , where: <nonzero digit> is a single ASCII character in the range of '1'-'9'. It represents the length of <digits> in number of bytes. <digits> is a number of ASCII characters in the range of '0'-'9', which together are a decimal representation of the number of succeeding data bytes. Example: `#49137<9137 bytes of binary data>`

Table 1.3: Network Master response data

1.3.2 Response Messages Terminator

A response message terminator indicates the end of the response message. Network Master appends the terminator to the end of a response message to indicate the end of the message. For Network Master the response message terminator is {NL} .

1.3.3 Prompt

For Ethernet based remote control a prompt can optionally be returned by the Network Master when all commands in a program message has completed. The prompt is inserted after the response message if any. It can be useful to enable the prompt when manually typing commands on the command line of the remote control interface. The prompt inserted is:

```
SCPI:>
```

1.4 Status

1.4.1 IEEE488.2 Standard Status and SCPI-defined Registers

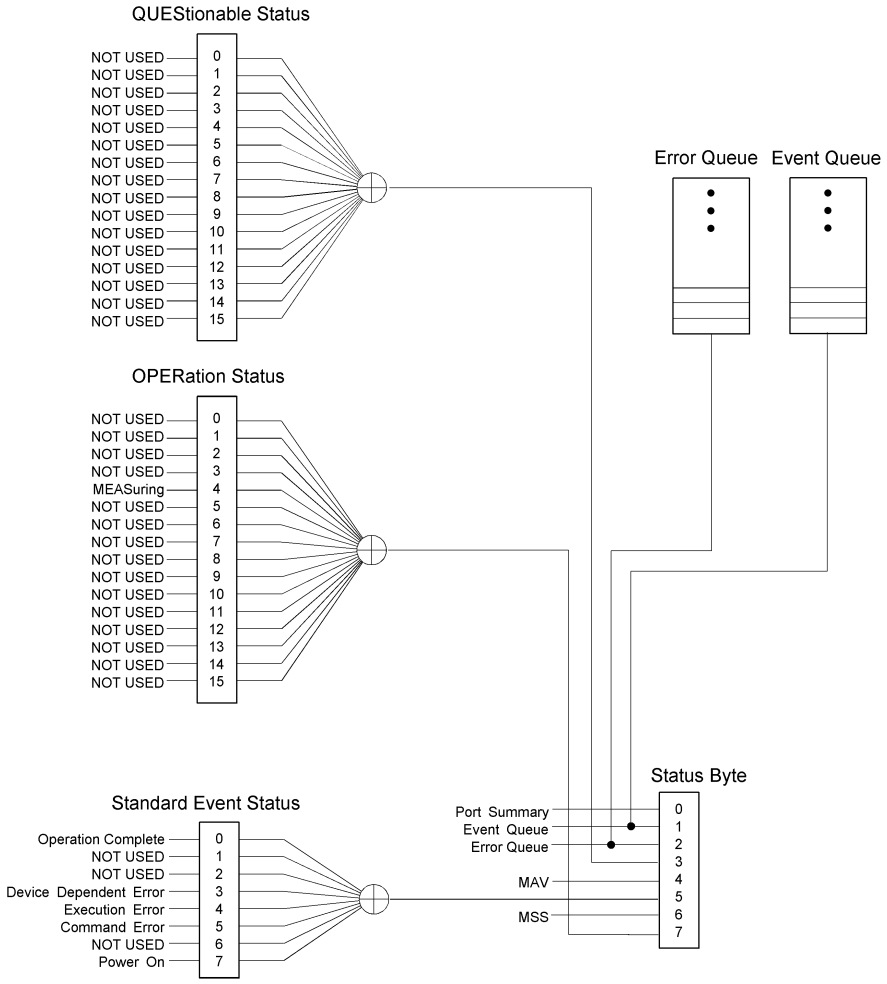

Figure 1.7: IEEE488.2 standard status and SCPI-defined registers/queues. $\oplus$ means logical OR.

Status Byte

Bit	Name	Description
0	Port 1	Summary-message bit for the Port Event Summary register. Use the `STATus:PRESet` command described in section 2.5.16 and the `STATus:PORT:ENABle` command described in section 2.5.13 to enable generation of this summary-message.
1	Event Queue	Summary-message bit for the Event Queue for the currently selected application server. Use the `SYSTem:ERRor[:NEXT]?` command described in section 2.3.2 to retrieve the messages.
2	Error Queue	Summary-message bit for the Error queue for all connected application servers. Use the `INSTrument:ERRor[:NEXT]?` command described in section 2.4.18 to retrieve the messages.
3	QUEStionable	Summary-message bit for the Questionable Status register. Use the `STATus:QUEStionable:ENABle` command described in section 2.5.8 to enable generation of this summary-message.
4	Output Queue	Summary-message bit for the Output Queue.
5	Standard Event	Summary-message bit for the Standard Event Status register. Use the `*ESE` command described in section 2.2.2 to enable generation of this summary-message.
6	Master Summary	The Master Summary Status message. Use the `*SRE` command described in section 2.2.7 to enable generation of this summary-message.
7	OPERation	Summary-message bit for the Operation Status register. Use the `STATus:OPERation:ENABle` command described in section 2.5.2 to enable generation of this summary-message.

Table 1.4: Bits in the Status Byte register (unused bits are not listed)

For more information about the Status Byte register, see section 2.2.8 on page 35.

Standard Event Status

All condition bits are immediately changed back to 0 after they are set. This means that the only way to check the bits is to read the Event register. For more information on what triggers the Device Dependent, Execution and Command Errors see the Error/Event Queue section on page page 19.

For more information on the Standard Event Status register see section 2.2.2 on page 32.

Error/Event Queue

When an unexpected error or event occurs, an entry is added to the Error/Event queue. This queue can hold 4 errors or events. If the queue overflows, the most recent events are discarded. A summary-message in bit 2 of the Status Byte is 1 when the queue is not empty. Table 1.6 gives an overview of the different errors and events inserted in the queue.

For more information about the Event queue, see section 2.4.18 on page 46.
For more information about the Error queue, see section 2.3.2 on page 36.

Bit	Name	Description
0	Operation Complete	The condition bit changes to 1 when *OPC command is received.
3	Device Dependent Error	The condition bit changes to 1 when a required SW or HW options is missing or the Error/Event queue is full.
4	Execution Error	The condition bit changes to 1 when a command fail to execute properly.
5	Command Error	The condition bit changes to 1 when a unknown or errored command is received.
7	Power On	The condition bit changes to 1 when the external power supply is connected.

Table 1.5: Bits in the Standard Event Status register (unused bits are not listed)

Event Number	Error Description
0	No Error (when queue is empty)
Command errors (Command Error bit is simultaneously set)	
-100	Command error
-102	Syntax error
-104	Data type error
-115	Unexpected number of parameters
-130	Suffix error
-131	Invalid suffix
-138	Suffix not allowed
Execution errors (Execution Error bit is simultaneously set)	
-200	Execution error
-220	Parameter error
-221	Settings conflict
-222	Data out of range
-224	Illegal parameter value
-250	Mass storage error
Device Dependent errors (Device Dependent Error bit is simultaneously set)	
1	Options Missing
-350	Queue overflow

Table 1.6: Errors and events that can occur in the Error/Event queue

Questionable Status

Bit	Name	Description
No bits in this register are currently in use.)		

Table 1.7: Bit in the Questionable Status register (unused bits are not listed)

For more information about the Questionable Status register, see section 2.5.6 on page 50.

Operation Status

For more information about the Operation Status register, see section 2.5.1 on page 49.

Bit	Name	Description
4	Measuring	The measuring condition bit changes to 1 when the an Application Server is running a measurement or a test. It returns to 0 when the measurement or test is stopped.

Table 1.8: Bit in the Operation Status register (unused bits are not listed)

1.4.2 Network Master Unique Status Registers

Figure 1.8 shows the structure of the Network Master Unique Status registers.

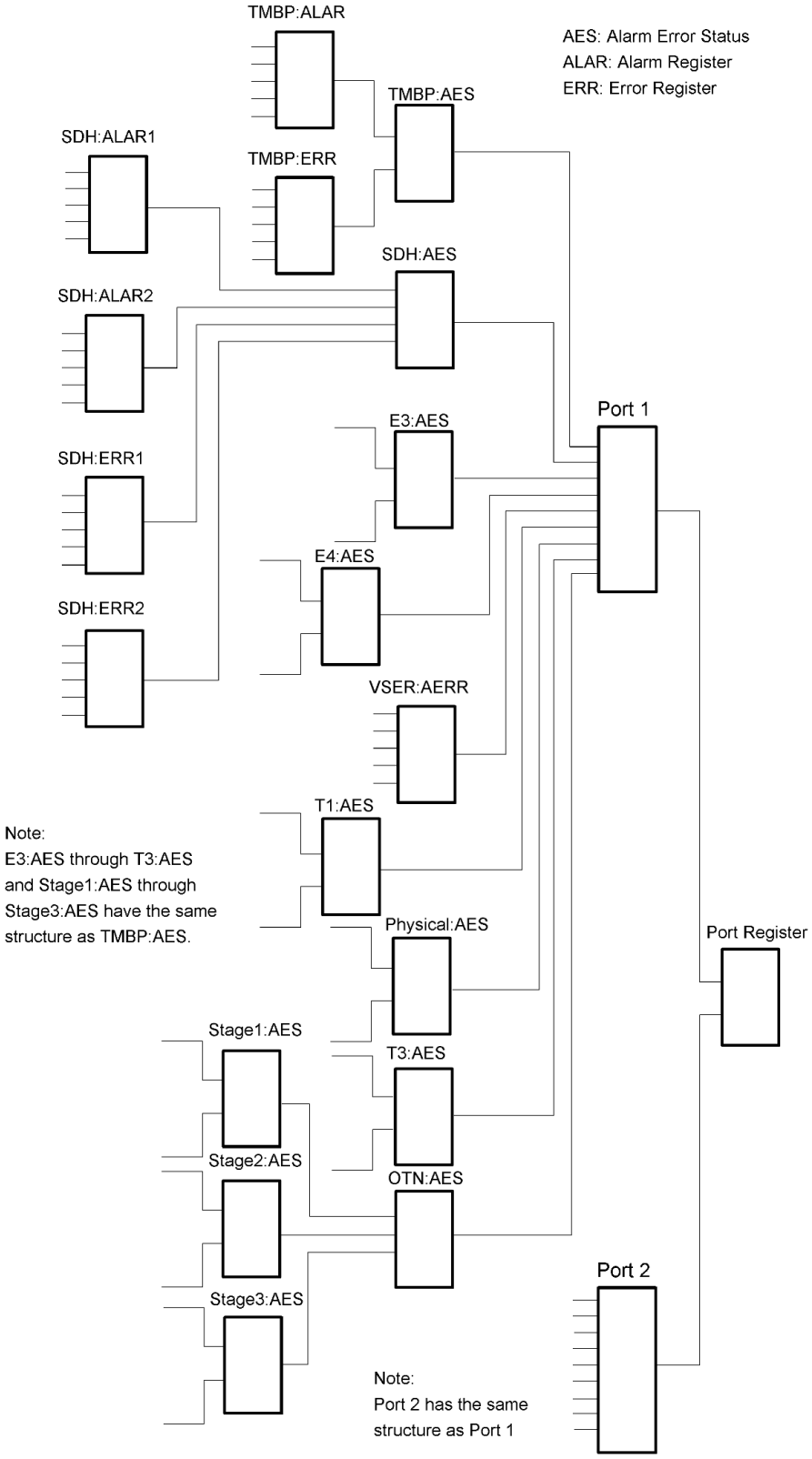

Figure 1.8: The Network Master Unique Status registers for some of the supported interfaces. Similar registers exist for T1, OTN, Physical, and T3 interfaces

The Network Master Unique Status registers are used to report alarms and errors for all interfaces. Each interface has one or more registers to represent the current alarm and error status. Each of these Alarm and Error registers are summarized in a General Interface Summery register (AESummery), see Figure 1.9. The

exact layout of each register is found under the Status section for each interface. There are two Port Status

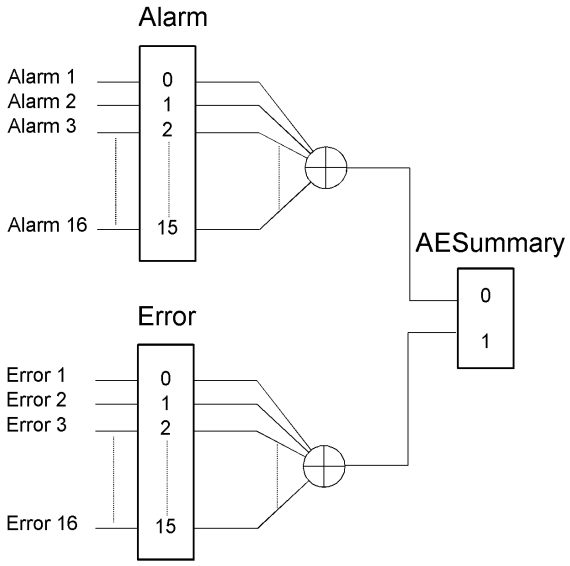

Figure 1.9: The general structure for the Alarms and Errors status register for the interfaces

registers, one for each port on the Network Master. The Port Status registers summarize the AESummary registers from the active interfaces. The Port Status registers are again summarized in bit 0 and 1 of the Status Byte, see Figure 1.10.

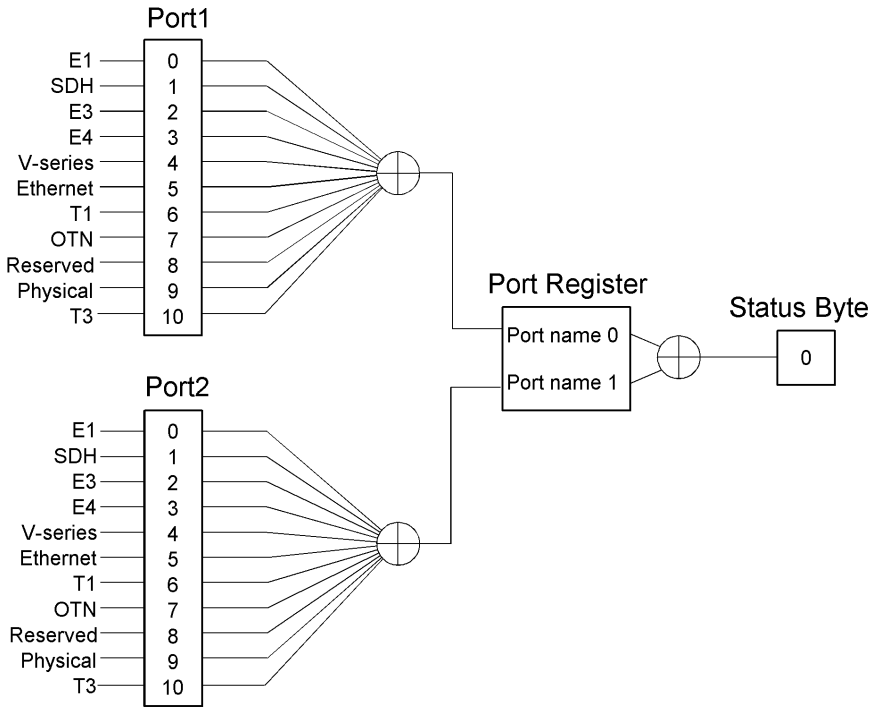

Figure 1.10: The structure for the Port Status register

All Network Master Unique Status registers follow the register model defined in section 11.4.2 of IEEE488.2 The register model is shown in Figure 1.11.

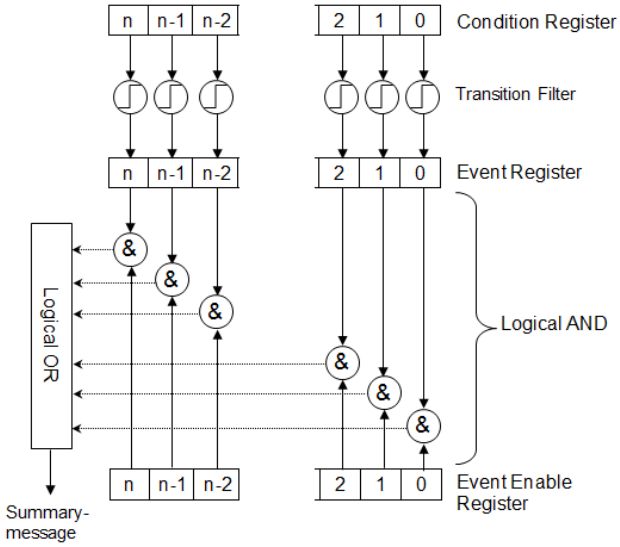

Figure 1.11: The register model for the Network Master Unique Status registers.

Condition Registers

The Condition registers reflect the real-time status of the instrument or summary-message bits of other status registers.

Transition Filters

The transition filters for the Network Master's Unique Status registers are locked to the "Positive transition" criteria. This means that events will be generated when the condition bits goes from 0 to 1. It is not possible to change the transition criteria.

Event Registers

The Event registers store the transition filter output. These registers are cleared when read.

Event Enable Registers

The Event Enable registers cannot be read or written and on power-on they are all set to zero. To enable the Event Enable registers and hence generation of summary-messages, the STATus:PRESet command must be send. The STATus:PRESet command changes all the bits in the Event Enable registers to 1.

Summary Register

The summarized status of the following registers is stored in the Summary Register (Refer to Table 1.9).

- Alarm event registers
- Error event registers
- Event registers of lower layer summary registers

Like other status registers, the Summary register consists of the Condition register and the Event register, according to the register model defined in section 11.4.2 of IEEE488.2. The Summary register is locked to positive transition criteria because it consists of lower layer Event registers.

1.4.3 Reading, Writing and Clearing Status Registers

The following two tables list the possibilities for reading and writing the various status registers and queues. They also show when and how registers are cleared or enabled.

Registers		**Reading**	**Writing**
IEEE488.2 standard status registers	Status Byte	`*STB?`	Not possible
	Service Request Enable	`*SRE?`	`*SRE`
	Standard Event Status	`*ESR?` After reading, the register content is cleared.	Not possible
	Standard Event Status Enable	`*ESE?`	`*ESE`
SCPI defined status registers	Error/Event Queue	`SYST:ERR?` After reading, the error/event is removed from the queue.	Not possible
	Operation Event	`STAT:OPER?` After reading, the register content is cleared.	Not possible
	Operation Enable	`STAT:OPER:ENAB?`	`STAT:OPER:ENAB`
	Questionable Event	`STAT:QUES?` After reading, the register content is cleared.	Not possible
	Questionable Enable	`STAT:QUES:ENAB?`	`STAT:QUES:ENAB`
Network Master unique status registers	Condition	`<Interface>:STAT:<Port>:<Register>:COND?`	Not possible
	Transition Filter	Not possible	Not possible
	Event	`<Interface>:STAT:<Port>:<Register>?` After reading, the register content is cleared.	Not possible
	Enable	Not possible	Not possible

Table 1.9: Reading and writing of Status registers

Registers		*RST	*CLS	PowerOn	STAT:PRES
IEEE488.2 standard status registers	Status Byte	No Changes	Cleared	Cleared	No Changes
	Service Request Enable	No Changes	No Changes	Cleared	No Changes
	Standard Event Status	No Changes	Cleared	Cleared	No Changes
	Standard Event Status Enable	No Changes	No Changes	Cleared	No Changes
SCPI defined status registers	Error/Event Queue	No Changes	Cleared	Cleared	No Changes
	Operation Event	No Changes	Cleared	Cleared	No Changes
	Operation Enable	No Changes	No Changes	Cleared	No Changes
	Questionable Event	No Changes	Cleared	Cleared	No Changes
	Questionable Enable	No Changes	No Changes	Cleared	No Changes
Network Master Unique Status registers	Condition	No Changes	No Changes	Cleared	No Changes
	Transition Filter	No Changes	No Changes	No Changes	No Changes
	Event	No Changes	Cleared	Cleared	No Changes
	Enable	No Changes	No Changes	Cleared	Enabled (all 1's)

Table 1.10: Status registers behaviour for different commands/events

Notes

The Condition resister of the Summary register is locked to positive transition criteria. Therefore, if clearing the register (*CLS) while an alarm or error is occurring, the register bits stay in "0" (cleared) in spite of the alarm or error occurrence.

1.5 Controller Example

One example of how to connect a controller to the Network Master instrument is described in this section.

1.5.1 PuTTY

PuTTY is a free Telnet/SSH client which supports raw TCP connections. With PuTTY it is possible to get terminal emulation access to the instrument. It is recommended to enable the prompt when using PuTTY. PuTTY does not support file streaming like the MEAS:EXP command.
PuTTY can be downloaded from http://www.chiark.greenend.org.uk/~sgtatham/putty/

Setup

1. Install PuTTY.
2. Start PuTTY.
3. In the PuTTY Configuration enable **Implicit CR in every LF** at **Category:→Terminal**.

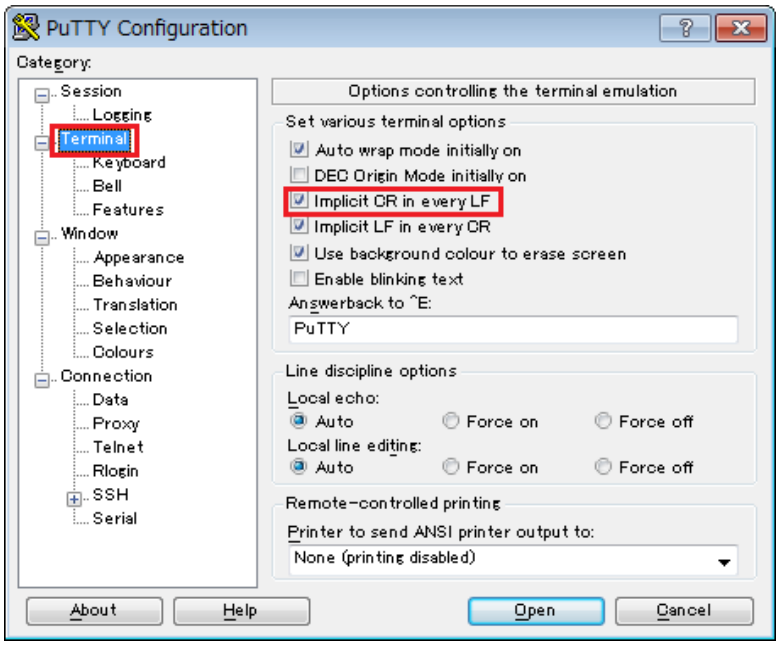

Figure 1.12: Enable **Implicit CR in every LF** in PuTTY

4. In the instrument GUI, find the instrument's **IP Address** information, see Figure 1.13. Then type it in PuTTY, see Figure 1.14.

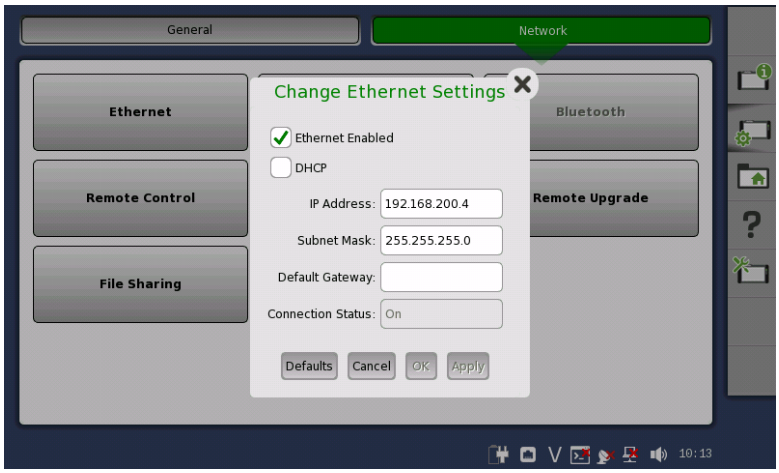

Figure 1.13: **IP Address** on the instrument

5. In PuTTY, type 56001 in the **Port** field, select the **Raw** radio button in the **Connection type** field, and click the **Open** button. see Figure 1.14.

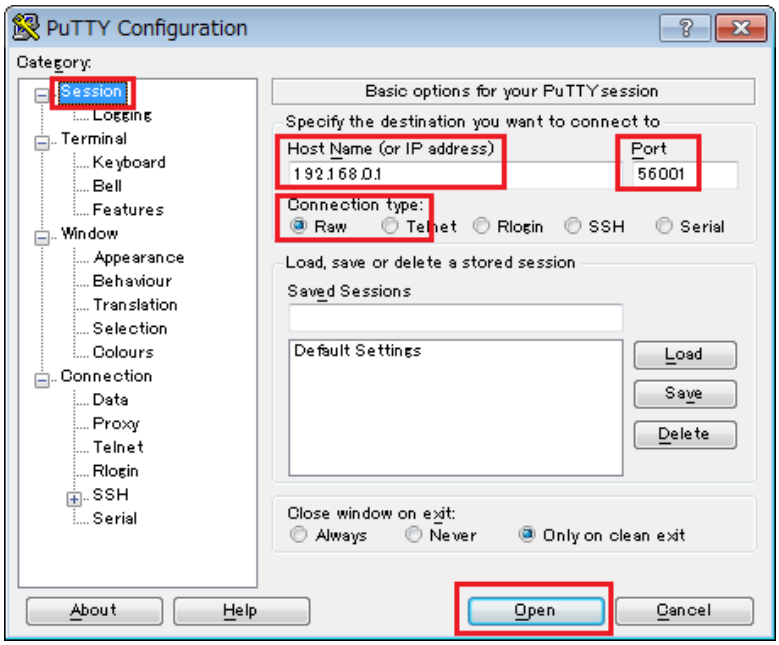

Figure 1.14: Specify the destination and **Open** the connection in PuTTY

6. A window appears, see Figure 1.15.

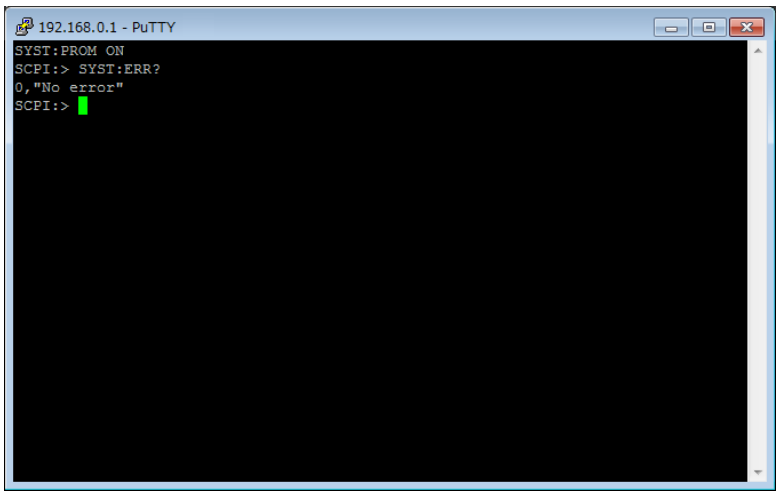

Figure 1.15: Connection established with PuTTY

1.6 Definitions

1.6.1 NaN (Not a Number)

NaN is defined in SCPI-99. NaN is represented as 9.91E37 (<NR3 NUMERIC RESPONSE DATA>) as defined in IEEE 754. NaN is also used to represent missing data.

1.6.2 → Right Arrow

The right arrow → used in this document has two meanings:

- On the left side of the arrow is a query and returned value on the right hand side.
  Example: `TMBP:RX1:PATT?` → `PRBS11`

1.6.3 Data Bit (DB)

Data bit is represented as DBx where x represents the bit index in a register. DB1 is always LSB.

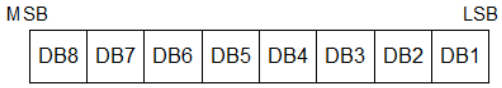

Figure 1.16: Data bit

1.6.4 Port Number (Logical Port)

Specify a logical port number assigned for each started application as a port number <Pt> in the SCPI command. The logical port numbers will be assigned in the order of Module1-Port1, Module1-Port2, Module2-Port1, and Module2-Port2. For OTDR module, specify the physical port number to "1".

Example of the OTDR application start command

- INST:STAR OTDR-OTDR,1-PORT1 (Figure 1.17)
  Physical Port : Logical Port
  1-PORT1 : PORT1

Figure 1.17: The physical and logical port number of the OTDR module

Notes

- To switch between the Single Mode port (SM) and the Multi Mode port (MM) of the OTDR module, refer to section 3.1.1 on page 59.

Chapter 2

SCPI Conformance Information

2.1 SCPI Version

The Network Master Remote Control application conforms to SCPI 1999.0

2.2 IEEE 488.2 Mandatory Commands

2.2.1 *CLS

Syntax	*CLS
Description	This command clears all the event registers summarized in the Status Byte register. The error queue is emptied. Neither the Standard Event Status Enable register, nor the Service Request Enable register are affected by this command.
Parameter	None.
Response	None.
Example	`*CLS`
Note	All active (SCPI) sessions has their own set of standard registers.

2.2.2 *ESE

Syntax	*ESE <mask>
Description	This command 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. This register is cleared at power-on. The *RST and *CLS commands do not affect this register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the enable mask: DB1 (1) = Operation Complete DB2 = NOT USED DB3 = NOT USED DB4 (8) = Device Dependent Error DB5 (16) = Execution Error DB6 (32) = Command Error DB7 = NOT USED DB8 (128) = Power On *MINimum=0, MAXimum=255*
Response	None.
Example	`*ESE 16`
Note	All active sessions has their own Standard Event Status Enable register.

Syntax	*ESE?
Description	This query returns the contents of the Standard Event Status Enable register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA> See the *ESE command for bit values for the enable mask.
Example	`*ESE?` → `16`
Note	

2.2.3 *ESR?

Syntax	*ESR?
Description	This query returns the contents of the Standard Event Status register. This register is cleared after being read.
Parameter	None.
Response	<value> = <NR1 NUMERIC RESPONSE DATA> The bits and their values for the register: DB1 (1) = Operation Complete DB2 = NOT USED DB3 = NOT USED DB4 (8) = Device Dependent Error DB5 (16) = Execution Error DB6 (32) = Command Error DB7 = NOT USED DB8 (128) = Power On
Example	`*ESR? → 49`
Note	All active sessions has their own Standard Event Status register.

2.2.4 *IDN?

Syntax	*IDN?
Description	This query returns the instrument identification over the interface.
Parameter	None.
Response	<manufacturer>,<model>,<serial>,<version> = <ARBITRARY ASCII RESPONSE DATA>
Example	`*IDN? → Anritsu,MT1000A,6123456789,1.00`
Note	

2.2.5 *OPC

Syntax	*OPC
Description	This command causes the instrument to generate the operation complete message in the Standard Event Status register when all pending selected instrument operations have been finished.
Parameter	None.
Response	None.
Example	`*OPC`
Note	All active sessions has their own Standard Event Status register.

Syntax	*OPC?
Description	This query places the ASCII character '1' into the instrument's output queue when all pending operations have been finished.
Parameter	None.
Response	<operation complete> = <NR1 NUMERIC RESPONSE DATA>
Example	`*OPC? → 1`
Note	Only application servers connected to by the current session are synchronized.

2.2.6 *RST

Syntax	*RST
Description	This command sets the instrument to reset setting (standard setting) stored in internal storage. The instrument is placed in the idle state awaiting a command. All running application/servers are closed when the *RST command is issued. The following are not changed: - Service Request Enable register (SRE) - Standard Event Status register (ESR) - Standard Event Status Enable register (ESE) - Any instrument specific Status Event or Status Event Enable registers
Parameter	None.
Response	None.
Example	`*RST`
Note	*RST is specially configured to be compatible with SCPI remote control. Only application servers connected to by the current session will set to the initial state..

2.2.7 *SRE

Syntax	*SRE <enable mask>
Description	This command sets bits in the Service Request Enable register. A 1 in a bit in the enable register enables the corresponding bit in the Status Byte, also sets the Master Summary Status bit (DB7) in the Status Byte. The register is cleared at power-on. The *RST and *CLS commands do not affect the register.
Parameter	<enable mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1 (1) = Port Event Summary DB2 (2) = Event Queue Summary for the currently selected application server. DB3 (4) = Error Queue Summary for all connected application servers. DB4 (8) = Questionable Status Summary DB5 (16) = Message Available (MAV) DB6 (32) = Standard Event Status Summary (ESB) DB7 = NOT USED DB8 (128) = Operation Status Summary *MINimum=0, MAXimum=255*
Response	None.
Example	`*SRE 255`
Note	All active sessions has their own Service Request Enable register.

Syntax	*SRE?
Description	This query returns the contents of the Service Request Enable register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA> See the *SRE command for bit values for the enable mask.
Example	`*SRE?` → `255`
Note	

2.2.8 *STB?

Syntax	*STB?
Description	This query returns the contents of the Status Byte register. The Master Summary Status (MSS) bit is true when any bit of the STB register is set and a matching bit in the Service Request Enable Register is set, see `*SRE`. The Status Byte register including the MSS is not altered by this query.
Parameter	None.
Response	<value> = <NR1 NUMERIC RESPONSE DATA> The bits and their values for the register: DB1 (1) = Port Event Summary DB2 (2) = Event Queue Summary for the currently selected application server. DB3 (4) = Error Queue Summary for all connected application servers. DB4 (8) = Questionable Status Summary DB5 (16) = Message Available (MAV) DB6 (32) = Standard Event Status Summary (ESB) DB7 (64) = Master Summary Status (MSS) DB8 (128) = Operation Status Summary
Example	`*STB? → 7`
Note	

2.2.9 *TST?

Syntax	*TST?
Description	This query returns whether or not the instrument completed the self-test without any detected errors.
Parameter	None.
Response	<result> = <NR1 NUMERIC RESPONSE DATA> 0: No self-test errors detected 1: Self-test error detected
Example	`*TST? → 0`
Note	Self-test is performed automatically at the time of power up.

2.2.10 *WAI

Syntax	*WAI
Description	This command prevents the instrument from executing any further commands until the current command has been finished. All pending operations are completed during the wait period.
Parameter	None.
Response	None.
Example	`*WAI`
Note	*WAI functions for commands called ”overlap command”. For now Network Master does not have any overlap commands. So *WAI does not work on the current Network Master.

2.3 SCPI System Subsystem Commands

2.3.1 SYSTem:VERSion?

Syntax	SYSTem:VERSion?
Description	This query returns the SCPI revision to which the system complies.
Parameter	None.
Response	<version> = <NR2 NUMERIC RESPONSE DATA>
Example	`SYST:VERS?` → `1999.0`
Note	

2.3.2 SYSTem:ERRor[:NEXT]?

Syntax	SYSTem:ERRor[:NEXT]?
Description	This query returns the oldest entry of the error queue and removes the returned entry from the queue.
Parameter	None.
Response	<error number> = <NR1 NUMERIC RESPONSE DATA>
	<description> = <STRING RESPONSE DATA>
Example	`SYST:ERR?` → `-222,"Data out of range"`
Note	All active sessions has their own error queue. Application server ID is added to each error message when the additional message is selected TEST or BOTH. Application server ID is -1 for system errors. Application server ID is fixed to 0 if no error message is in error queue. Error command is added to each error message when the additional message is selected COMMand or BOTH.

2.3.3 SYSTem:ERRor:ADDitional[:MESSage]

Syntax	SYSTem:ERRor:ADDitional[:MESSage] <message>
Description	This command select additional message in the error message.
Parameter	<message> = <CHARACTER PROGRAM DATA> NONE TEST COMMand BOTH
Response	None.
Example	`SYST:ERR:ADD BOTH` `SYST:ERR?` → `-115,"Unexpected number of parameters:-1:INST:TERM"`
Note	This setting is applied only for the current session and defaulted to NONE when session closed. See also SYSTem:ERRor[:NEXT]?

Syntax	SYSTem:ERRor:ADDitional[:MESSage]?
Description	This query returns additional message in the error message.
Parameter	None.
Response	<message> = <CHARACTER RESPONSE DATA>
Example	`SYST:ERR:ADD?` → `NON`
Note	

2.3.4 SYSTem:DATE

Syntax	SYSTem:DATE <year>,<month>,<day>
Description	This command sets the date of the internal calender.
Parameters	<year> = <NUMERIC PROGRAM DATA> *MINimum = 1997, MAXimum = 2036*
	<month> = <NUMERIC PROGRAM DATA> *MINimum = 1, MAXimum = 12*
	<day> = <NUMERIC PROGRAM DATA> *MINimum = 1, MAXimum = 31*
Response	None.
Example	`SYST:DATE 2009,12,31`
Note	

Syntax	SYSTem:DATE?
Description	This query returns the date of the internal calender.
Parameter	None.
Response	<year>,<month>,<day> = <NR1 NUMERIC RESPONSE DATA>
Example	`SYST:DATE?` → `2009,07,04`
Note	

2.3.5 SYSTem:TIME

Syntax	SYSTem:TIME <hour>,<minute>,<second>
Description	This command sets the time of the internal clock.
Parameters	<hour> = <NUMERIC PROGRAM DATA> *MINimum = 0, MAXimum = 23*
	<minute> = <NUMERIC PROGRAM DATA> *MINimum = 0, MAXimum = 59*
	<second> = <NUMERIC PROGRAM DATA> *MINimum = 0, MAXimum = 59*
Response	None.
Example	`SYST:TIME 23,59,59`
Note	

Syntax	SYSTem:TIME?
Description	This query gets the time of the internal clock.
Parameter	None.
Response	<hour>,<minute>,<second> = <NR1 NUMERIC RESPONSE DATA>
Example	`SYST:TIME?` → `15,45,03`
Note	

2.3.6 SYSTem:REBoot

Syntax	SYSTem:REBoot
Description	This command will force a reboot of the instrument. A TCP remote connection to the instrument will be lost.
Parameter	None.
Response	None.
Example	`SYST:REB`
Note	

2.3.7 SYSTem:GPS:NSATellites?

Syntax	SYSTem:GPS:NSATellites?
Description	This query returns the number of satellites found by GPS.
Parameter	None.
Response	<count> = <NR1 NUMERIC RESPONSE DATA>
Example	`SYST:GPS:NSAT?` → `5`
Note	Return "0" if GPS is not available.

2.3.8 SYSTem:GPS:TIME?

Syntax	SYSTem:GPS:TIME?
Description	This query returns the GPS time.
Parameter	None.
Response	<time> = <CHARACTER RESPONSE DATA>
Example	`SYST:GPS:TIME?` → `2014-01-01T12:34:56`
Note	Return "0" if GPS is not available.

2.3.9 SYSTem:GPS:LOCation?

Syntax	SYSTem:GPS:LOCation?
Description	This query returns the location.
Parameter	None.
Response	<location> = <NR2 NUMERIC RESPONSE DATA>
Example	`SYST:GPS:LOC?` → `85 26.8444N, 22 20.4508E`
Note	Return "0" if GPS is not available.

2.3.10 SYSTem:COMMunicate:TERMinator

Syntax	SYSTem:COMMunicate:TERMinator <terminator>
Description	This command sets the terminator code which is appended to the query response.
Parameter	<terminator> = <CHARACTER PROGRAM DATA> NONE(only GPIB) LF CRLF
Response	None.
Example	`SYST:COMM:TERM LF`
Note	This setting is applied only for the current session and defaulted to CRLF when session closed.

Syntax	SYSTem:COMMunicate:TERMinator?
Description	This query returns the terminator code which is appended to the query response.
Parameter	None.
Response	<terminator> = <CHARACTER RESPONSE DATA>
Example	`SYST:COMM:TERM?` → `LF`
Note	

2.3.11 SYSTem:PROMpt

Syntax	SYSTem:PROMpt <enable>
Description	This command enables/disables appending of prompt to all replies from Remote Control interface.
Parameter	<enable> = <BOOLEAN PROGRAM DATA>
Response	None.
Example	`SYST:PROM 1`
Note	The prompt string is "SCPI:> " This setting is applied only for the current session and forgets when session closed.

Syntax	SYSTem:PROMpt?
Description	This query returns status of the prompt.
Parameter	None.
Response	<enable> = <BOOLEAN PROGRAM DATA>
Example	`SYST:PROM?` → `SCPI:>1`
Note	

2.3.12 SYSTem:LOCal:CONTrol

Syntax	SYSTem:LOCal:CONTrol <enable>
Description	This command enables/disables local control.
Parameter	<enable> = <BOOLEAN PROGRAM DATA>
Response	None.
Example	`SYST:LOC:CONT 1`
Note	This setting is applied all connected sessions and forgets when turn off SCPI.

Syntax	SYSTem:LOCal:CONTrol?
Description	This query returns enables/disables local control.
Parameter	None.
Response	<enable> = <BOOLEAN PROGRAM DATA>
Example	`SYST:LOC:CONT?` → `1`
Note	

2.3.13 SYSTem:WAIT[:IDLE]

Syntax	SYSTem:WAIT[:IDLE]
Description	This command waits for the instrument to go into IDLE state, i.e. no measurement or test is pending, running, loading or being stored. It also waits for load and save of settings to finish.
Parameter	None.
Response	None.
Example	`SYST:WAIT`
Note	There must be a connected application server for this command to be recognized as a legal command.
	Be careful when using this command as it may lead to undesired blocking of the remote interface. In some situations the instrument requires a remote command or other user intervention in order to return to IDLE state; e.g. when a measurement is running and measurement stop mode is set to MANual (`MEAS:SET:STOP` → `MAN`). In this situation, for the instrument to return to IDLE state press the START/STOP button on the GUI or apply the `MEASurement:STOP` command. The latter is NOT possible if `SYST:WAIT:IDLE` is currently being executed. If an undesired blocking occurs close and re-open the remote connection. And then send *RST command to reset undesired blocking remote connection.

2.3.14 SYSTem:WAIT:DURation

Syntax	SYSTem:WAIT:DURation <seconds>
Description	This command waits for the specified number of seconds.
Parameter	<seconds> = <NUMERIC PROGRAM DATA> *MINimum = 1, MAXimum = 3600*
Response	None.
Example	`SYST:WAIT:DUR 5`
Note	There must be a connected application server for this command to be recognized as a legal command.

2.4 SCPI Instrument Subsystem Commands

To use the application-specific SCPI commands, you need to connect the client session to the application server. By sending SCPI commands to the application server connected, you can control the application.

2.4.1 Connection to Application Server

To connect the client session to the application server, do one of the following:

- Use the `INST:CONN:ALL` command.
  This command connects a client session to the application server which is not occupied.
  If a different client session is already connected to the server (application server is occupied), the connection attempt fails.
- Use the `INST:CONN` command.
  This command connects a client session to all application servers which are not occupied.
- Use the `INST:STAR` command.
  This command starts a new application server and connects a client session to the application server.

2.4.2 Connection to multiple applications

A client session can connect to multiple application servers simultaneously. In this case, the SCPI command destination will be the selected application server.
The destination can be set by using the `INST:SEL` command. To confirm the selected application server, use the `INST?` command.
When a new application server has started by the `INST:STAR` command, the SCPI command destination will be changed to the newly started application server.

2.4.3 Connection from multiple users

When using a Network Master by multi-users, multiple client sessions connect to a Network Master. Under this condition, make sure that the application server is not occupied by any other user.

If a client session has already connected to an application server, Network Master cannot connect any other client session to the application server.
To connect to the application server occupied by a client session, you need to release the application server by disconnecting the client session which is connecting currently. To release the application server, do one of the following:

- Use the `INST:DISC` command.
  You can disconnect the server by sending the `INST:DISC` command from the connecting client session.
- Terminate the Client Session.
  After the client session terminated, all application servers to which the client session was connected are released.
  You can terminate the client session by disconnecting connection to Network Master from the User PC.

You can connect a client session to the released application server again by using the `INST:CONN` command or the `INST:CONN:ALL` command.

2.4.4 Force Termination of Application Server

An application server to which a session is connecting doesn't receive SCPI commands from any other session. An exception is the `INST:TERM:FORC` command. This command is always accepted even if sent from the other session and terminates the application server to which a session is connecting.

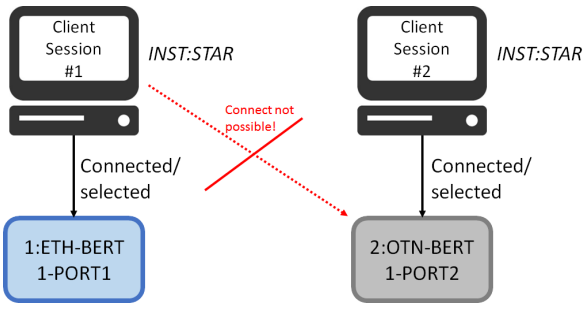

Figure 2.1: `INST:STAR` automatically *connects* and selects started application server.

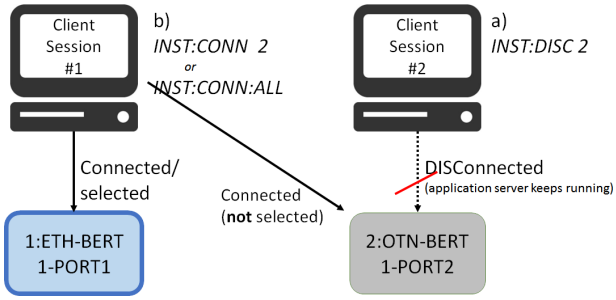

Figure 2.2: Original client session has to *disconnect*, before another client session can *connect*.

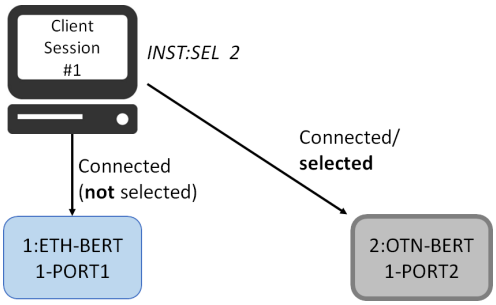

Figure 2.3: When connected to multiple application servers, client session *selects* to which application server the application specific commands are dispatched.

2.4.5 INSTrument:STARt[:DEFault]

Syntax	INSTrument:STARt[:DEFault] <app name>[, <port name>[, <port name>,...]]
Description	This command starts an application server with default settings.
Parameters	<app name> = <CHARACTER PROGRAM DATA> TP-APS-OTN: OTN Automatic Protection Switching application. TP-APS-SDHPDH: SDH/PDH Automatic Protection Switching application. TP-APS-SDHPDH-OTN: SDH/PDH over OTN Automatic Protection Switching application. TP-BERT-CPRI: CPRI Bit Error Rate Test application. TP-BERT-CPRI-OTN: CPRI over OTN Bit Error Rate Test application. TP-BERT-ETH: Ethernet Bit Error Rate Test application. TP-BERT-ETH-OTN: Ethernet over OTN Bit Error Rate Test application. TP-BERT-FC: Fibre Channel Bit Error Rate Test application. TP-BERT-FC-OTN: Fibre Channel over OTN Bit Error Rate Test application. TP-BERT-OTN: OTN Bit Error Rate Test application. TP-BERT-SDHPDH: PDH/SDH Bit Error Rate Test application. TP-BERT-SDHPDH-OTN: PDH/SDH over OTN Bit Error Rate Test application. TP-CABLE-ETH: Ethernet cable test application. TP-CHSTAT-ETH: Ethernet channel statistics application. TP-MONGEN-ETH: Ethernet monitor/generate application. TP-MONGEN-ETH-OTN: Ethernet over OTN monitor/generate application. TP-NOFRAME-DEVICE: No frame device test (Unframed Bit Error Rate Test) application. TP-PASS-CPRI: CPRI pass-through application. TP-PASS-ETH: Ethernet pass-through application. TP-PING-ETH: Ethernet ICMP ping application. TP-REFL-ETH: Ethernet reflector application. TP-REFL-ETH-OTN: Ethernet over OTN reflector application. TP-REFL-FC: Fibre Channel reflector application. TP-REFL-FC-OTN: Fibre Channel over OTN reflector application. TP-RFC-ETH: Ethernet RFC-2544 test application. TP-RFC-ETH-OTN: Ethernet over OTN RFC-2544 test application. TP-RFC6349-ETH: Ethernet RFC-6349 test application. TP-RTD-OTN: OTN Round Trip Delay test application. TP-RTD-SDHPDH: SDH/PDH Round Trip Delay test application. TP-RTD-SDHPDH-OTN: SDH/PDH Round Trip Delay test application. TP-SAT-ETH: Ethernet Service Activation Test application. TP-SAT-ETH-OTN: Ethernet over OTN Service Activation Test application. TP-TRACE-ETH: Ethernet trace-route application. TP-SYNCTEST-ETH: Ethernet sync test application. OTDR-OTDR: OTDR application.
	<port name> = <CHARACTER PROGRAM DATA> 1-PORT1: Port 1 on module 1 1-PORT2: Port 2 on module 1 2-PORT1: Port 1 on module 2 2-PORT2: Port 2 on module 2 1-PORT-SING1: Single port 1 on module 1 (for MU110011A/MU110012A) The physical port(s) given as parameters are assigned to logical port numbers in the application server. The logical port number range is from 1 to the actual number of port assigned to the application server. The list of ports returned by the `INSTrument:STATe? <id>` command reveals the virtual port number sequence. The ports will be shown sorted in (1^{st}) module order and (2^{nd}) port order.
Response	None.
Example	`INST:STAR TP-BERT-OTN,1-PORT1` `INST?` → `2`
Note	Operators can get started application server ID by using the `INSTrument:SELect?` command. When you start an application server, the application server will be connected and selected automatically. When using this command, the application server will be started with DEFault setup.

2.4.6 INSTrument:STARt:LAST

Syntax	INSTrument:STARt:LAST <app name>[, <port name>[, <port name>,...]]
Description	This command starts an application server and loads the applicable auto saved settings.
Parameters	The parameters of this command are similar to the parameters of the `INSTrument:STARt[:DEFault]` command above.
Response	None.
Example	`INST:STAR:LAST TP-BERT-OTN,1-PORT1` `INST? → 2`
Note	Operators can get started application server ID by using the `INSTrument:SELect?` command. When you start an application server, the application server will be connected and selected automatically. When using this command, the application server will be started with LAST setup.

2.4.7 INSTrument:STARt:GUI

Syntax	INSTrument:STARt:GUI [<test index>]
Description	This command starts GUI for the application server.
Parameter	<test index> = <NUMERIC PROGRAM DATA> Defaults to the current application server if a value is omitted.
Response	None.
Example	`INST:STAR TP-BERT-OTN,1-PORT1` `INST:STAR:GUI`
Note	Must connect to the application server first.

2.4.8 INSTrument:TERMinate

Syntax	INSTrument:TERMinate [<test index>]
Description	This command terminates an application server.
Parameter	<test index> = <NUMERIC PROGRAM DATA> Defaults to the current application server if a value is omitted.
Response	None.

Example	`INST:STAR TP-BERT-OTN,1-PORT1`
	`INST? → 2`
	`INST:TERM 2`

Note	Must connect to the application server first.

2.4.9 INSTrument:TERMinate:FORCe

Syntax	INSTrument:TERMinate:FORCe [<test index>]
Description	This command force terminates an application server.
Parameter	<test index> = <NUMERIC PROGRAM DATA> Defaults to the current application server if a value is omitted.
Response	None.

Example	`INST:STAR TP-BERT-OTN,1-PORT1`
	`INST? → 2`
	`INST:TERM:FORC 2`

Note	WARNING. This command can terminate the application to which the other session communicating.

2.4.10 INSTrument:COUNt?

Syntax	INSTrument:COUNt?
Description	This query returns the number of active application servers.
Parameter	None.
Response	<count> = <NR1 NUMERIC RESPONSE DATA>
Example	`INST:COUN? → 2`
Note	

2.4.11 INSTrument:CATalog?

Syntax	INSTrument:CATalog?
Description	This query returns test indices, application name and port name of all active application servers.
Parameter	None.
Response	<test index> = <EXPRESSION RESPONSE DATA> Expression format: (<test index>,<app name>,<port name>)
Example	`INST:CAT?` → `(1,TP-SAT-ETH,1-PORT1),(2,TP-BERT-SDHPDH,1-PORT2)`
Note	Return -1 if no application server is running.

2.4.12 INSTrument:STATe?

Syntax	INSTrument:STATe? <test index>
Description	This query returns status information about a given application server.
Parameter	<test index> = <NUMERIC PROGRAM DATA>
Response	<app name> = <CHARACTER RESPONSE DATA>
	<client connection> = <CHARACTER RESPONSE DATA> NON IP Address
	<select status> = <CHARACTER RESPONSE DATA> NON SELECTED
	<port name> = <CHARACTER RESPONSE DATA> Expression format: Character list
Example	`INST:STAT? 1` → `TP-BERT-OTN,192.168.128.21,SELECTED,1-PORT1,2-PORT2...`
Note	

2.4.13 INSTrument:CONNect

Syntax	INSTrument:CONNect <test index>
Description	This command allows client session to connect to an existing application server.
Parameter	<test index> = <NUMERIC PROGRAM DATA>
Response	None.
Example	`INST:CONN 1`
Note	This command fails if the application server is already connect to by another client session. Use `INSTrument:CATalog?` query to acquire the list of all existing application servers. If command succeeds, the application server will be selected automatically.

2.4.14 INSTrument:CONNect:ALL

Syntax	INSTrument:CONNect:ALL
Description	This command allows client session to connect to all existing application servers.
Parameter	None.
Response	None.
Example	`INST:CONN:ALL`
Note	This command fails if no application server is selected when the command exits, e.g. because all application servers was already connected to by other client sessions, or because there are no application servers at all. If connected to multiple application servers, the application server with the lowest index will be selected, but selected index will not change if an application server was already selected prior to issuing this command.

2.4.15 INSTrument:CONNect[:CATalog]?

Syntax	INSTrument:CONNect[:CATalog]?
Description	This query returns indices of all application servers for the current client session.
Parameter	None.
Response	<test index> = <EXPRESSION RESPONSE DATA> Expression format: Numeric list
Example	`INST:CONN?` → `0,1,...`
Note	Return -1 if current client session has no application servers.

2.4.16 INSTrument:DISConnect

Syntax	INSTrument:DISConnect <test index>
Description	This command disconnect the application server from the client session.
Parameter	<test index> = <NUMERIC PROGRAM DATA>
Response	None.
Example	`INST:DISC 1`
Note	When current application is disconnected, the application server lowest ID will be selected automatically. When a client session is disconnected all the currently connected application servers will be disconnected automatically.

2.4.17 INSTrument[:SELect]

Syntax	INSTrument[:SELect] <test index>
Description	This command select the current application server.
Parameter	<test index> = <NUMERIC PROGRAM DATA>
Response	None.

Example	`INST:STAR TP-BERT-OTN,1-PORT1`
	`INST?` → `1`
	`INST:STAR TP-BERT-OTN,1-PORT2`
	`INST?` → `2`
	`INST 1`

Note	All future commands are forwarded to the current application server until the current application server is changed.

Syntax	INSTrument[:SELect]?
Description	This query returns index of the currently selected application server.
Parameter	None.
Response	<count> = <NR1 NUMERIC RESPONSE DATA>

Example	`INST:STAR TP-BERT-OTN,1-PORT1`
	`INST?` → `1`
	`INST:STAR TP-BERT-OTN,1-PORT2`
	`INST?` → `2`
	`INST 1`

Note	Return -1 if current client session does not have a currently selected application server.

2.4.18 INSTrument:ERRor[:NEXT]?

Syntax	INSTrument:ERRor[:NEXT]?
Description	This query returns the oldest entry of the event queue for the currently selected application server and removes the returned entry from the queue.
Parameter	None.
Response	<description> = <STRING RESPONSE DATA>
Example	`INST:ERR?` → `"Signal abnormal"`
Note	Application server has its own event queue. This event queue is not destroyed when the session is closed. Before checking the server status by using this command, controller may check DB2 (Event Queue Summary) of the Status Byte Register to see if it is set. DB2 of the Status Byte Register will aggregate input from potentially many application servers, and clear if event queues of all servers are empty.

2.4.19 INSTrument:PORT?

Syntax	INSTrument:PORT?
Description	This query returns ports assigned for the currently selected application server.
Parameter	None.
Response	<port name> = <CHARACTER RESPONSE DATA> Expression format: Character list
Example	`INST:STAR TP-BERT-OTN,1-PORT1,1-PORT2` `INST:PORT?` → `1-PORT1,1-PORT2`
Note	Operates on the application server currently selected by the `INSTrument:SELect command`. Returns `NON` if the client session does not have a currently selected application server. Returns `NON` if no ports are assigned for the currently selected application server.

2.4.20 INSTrument:PORT:FREE?

Syntax	INSTrument:PORT:FREE? <app name>
Description	This query returns all unused ports for the target application name.
Parameter	<app name> = <CHARACTER PROGRAM DATA>
Response	<port name> = <CHARACTER RESPONSE DATA> Expression format: Character list
Example	`INST:PORT:FREE? TP-BERT-OTN` → `1-PORT1,1-PORT2`
Note	

2.4.21 INSTrument:PORT:CATalog?

Syntax	INSTrument:PORT:CATalog?
Description	This query returns all ports of device.
Parameter	None.
Response	<port name> = <CHARACTER RESPONSE DATA> Expression format: Character list
Example	`INST:PORT:CAT?` → `1-PORT1,1-PORT2`
Note	When SCPI client uses any of the STATus:PORT: commands, <bit> index is the same as returned by the INSTrument:PORT:CATalog?

2.4.22 INSTrument:MODule:CATalog?

Syntax	INSTrument:MODule:CATalog?
Description	This query returns module names of device.
Parameter	None.
Response	{<module n>,}* = <CHARACTER RESPONSE DATA>
Example	`INST:MOD:CAT?` → `MU100010A,MU100011A`
Note	

2.4.23 INSTrument:CTRL:NAME?

Syntax	INSTrument:CTRL:NAME?
Description	This query returns model name.
Parameter	None.
Response	<model name> = <CHARACTER RESPONSE DATA>
Example	`INST:CTRL:NAME?` → `MT1000A`
Note	

2.4.24 INSTrument:CTRL:SN?

Syntax	INSTrument:CTRL:SN?
Description	This query returns controller serial number.
Parameter	None.
Response	<serial number> = <CHARACTER RESPONSE DATA>
Example	`INST:CTRL:SN?` → `1234567890`
Note	

2.4.25 INSTrument:CTRL:TRT?

Syntax	INSTrument:CTRL:TRT?
Description	This query returns controller total run time(sec).
Parameter	None.
Response	<time> = <NR1 NUMERIC RESPONSE DATA>
Example	`INST:CTRL:TRT?` → `5000000`
Note	

2.4.26 INSTrument:CTRL:OPTion:CATalog?

Syntax	INSTrument:CTRL:OPTion:CATalog?
Description	This query returns controller enabled options.
Parameter	None.
Response	{<option n>, }* = <CHARACTER RESPONSE DATA>
Example	`INST:CTRL:OPT:CAT?` → `MT1000A-303,MT1000A-005`
Note	

2.4.27 INSTrument:MODule<Md>:NAME?

Syntax	INSTrument:MODule<Md>:NAME?
Description	This query returns module model number.
Parameter	None.
Response	<model name> = <CHARACTER RESPONSE DATA>
Example	`INST:MOD1:NAME?` → `MT1000A`
Note	

2.4.28 INSTrument:MODule<Md>:SN?

Syntax	INSTrument:MODule<Md>:SN?
Description	This query returns module serial number.
Parameter	None.
Response	<serial number> = <CHARACTER RESPONSE DATA>
Example	`INST:MOD1:SN?` → `1234567890`
Note	

2.4.29 INSTrument:MODule<Md>:TRT?

Syntax	INSTrument:MODule<Md>:TRT?
Description	This query returns module total run time(sec).
Parameter	None.
Response	<time> = <NR1 NUMERIC RESPONSE DATA>
Example	`INST:MOD1:TRT?` → `5000000`
Note	

2.4.30 INSTrument:MODule<Md>:OPTion:CATalog?

Syntax	INSTrument:MODule<Md>:OPTion:CATalog?
Description	This query returns module enabled options.
Parameter	None.
Response	{<option n>,}* = <CHARACTER RESPONSE DATA>
Example	`INST:MOD1:OPT:CAT?` → `MU100010A-001,MU100010A-002`
Note	

2.5 SCPI Status Subsystem Commands

2.5.1 STATus:OPERation[:EVENt]?

Syntax	STATus:OPERation[:EVENt]?
Description	This query returns and clears the operation event register.
Parameter	None.
Response	<value> = <NR1 NUMERIC RESPONSE DATA> The bits and their values for the register: DB1-DB4 = NOT USED DB5 (16) = Measuring DB6-DB16 = NOT USED
Example	`STAT:OPER?` → `16`
Note	All active sessions has their own register and it is cleared when the session starts.

2.5.2 STATus:OPERation:CONDition?

Syntax	STATus:OPERation:CONDition?
Description	This query returns the operation condition register.
Parameter	None.
Response	<value> = <NR1 NUMERIC RESPONSE DATA> The bits and their values for the register: DB1-DB4 = NOT USED DB5 (16) = Measuring DB6-DB16 = NOT USED
Example	`STAT:OPER:COND?` → `16`
Note	

2.5.3 STATus:OPERation:ENABle

Syntax	STATus:OPERation:ENABle <mask>
Description	This command sets the enable mask for the operation event register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1-DB4 = NOT USED DB5 (16) = Measuring DB6-DB16 = NOT USED *MINimum = 0, MAXimum = 65535*
Response	None.
Example	`STAT:OPER:ENAB 65535`
Note	

Syntax	STATus:OPERation:ENABle?
Description	This query returns the enable mask for the operation event register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA>
Example	`STAT:OPER:ENAB?` → `16`
Note	

2.5.4 STATus:OPERation:PTRansition

Syntax	STATus:OPERation:PTRansition <mask>
Description	This command sets the positive transition filter for the operation event register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1-DB4 = NOT USED DB5 (16) = Measuring DB6-DB16 = NOT USED *MINimum = 0, DEFault = 65535, MAXimum = 65535*
Response	None.
Example	`STAT:OPER:PTR 16384`
Note	

Syntax	STATus:OPERation:PTRansition?
Description	This query returns the positive transition filter for the operation event register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA>
Example	`STAT:OPER:PTR?` → `16384`
Note	

2.5.5 STATus:OPERation:NTRansition

Syntax	STATus:OPERation:NTRansition <mask>
Description	This command sets the negative transition filter for the operation event register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1-DB4 = NOT USED DB5 (16) = Measuring DB6-DB16 = NOT USED *MINimum = 0, MAXimum = 65535*
Response	None.
Example	`STAT:OPER:NTR 16384`
Note	

Syntax	STATus:OPERation:NTRansition?
Description	This query returns the negative transition filter for the operation event register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA>
Example	`STAT:OPER:NTR?` → `16384`
Note	

2.5.6 STATus:QUEStionable[:EVENt]?

Syntax	STATus:QUEStionable[:EVENt]?
Description	This query returns and clears the questionable event register.
Parameter	None.
Response	<value> = <NR1 NUMERIC RESPONSE DATA> The bits and their values for the register: DB1-DB14 = NOT USED DB15 (16384) = Command Warning DB16 = NOT USED
Example	`STAT:QUES?` → `16384`
Note	All active sessions has their own register and it is cleared when the session starts.

2.5.7 STATus:QUEStionable:CONDition?

Syntax	STATus:QUEStionable:CONDition?
Description	This query returns the questionable condition register.
Parameter	None.
Response	<value> = <NR1 NUMERIC RESPONSE DATA> The bits and their values for the register: DB1-DB14 = NOT USED DB15 (16384) = Command Warning DB16 = NOT USED
Example	`STAT:QUES:COND?` → `16384`
Note	

2.5.8 STATus:QUEStionable:ENABle

Syntax	STATus:QUEStionable:ENABle <mask>
Description	This command sets the enable mask for the questionable event register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1-DB14 = NOT USED DB15 (16384) = Command Warning DB16 = NOT USED *MINimum = 0, MAXimum = 65535*
Response	None.
Example	`STAT:QUES:ENAB 16384`
Note	

Syntax	STATus:QUEStionable:ENABle?
Description	This query returns the enable mask for the questionable event register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA>
Example	`STAT:QUES:ENAB?` → `16384`
Note	

2.5.9 STATus:QUEStionable:PTRansition

Syntax	STATus:QUEStionable:PTRansition <mask>
Description	This command sets the positive transition filter for the questionable event register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1-DB14 = NOT USED DB15 (16384) = Command Warning DB16 = NOT USED *MINimum = 0, DEFault = 65535, MAXimum = 65535*
Response	None.
Example	`STAT:QUES:PTR 16384`
Note	

Syntax	STATus:QUEStionable:PTRansition?
Description	This query returns the positive transition filter for the questionable event register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA>
Example	`STAT:QUES:PTR?` → `16384`
Note	

2.5.10 STATus:QUEStionable:NTRansition

Syntax	STATus:QUEStionable:NTRansition <mask>
Description	This command sets the negative transition filter for the questionable event register.
Parameter	<mask> = <NUMERIC PROGRAM DATA> The bits and their values for the register: DB1-DB14 = NOT USED DB15 (16384) = Command Warning DB16 = NOT USED *MINimum = 0, MAXimum = 65535*
Response	None.
Example	`STAT:QUES:NTR 16384`
Note	

Syntax	STATus:QUEStionable:NTRansition?
Description	This query returns the negative transition filter for the questionable event register.
Parameter	None.
Response	<mask> = <NR1 NUMERIC RESPONSE DATA>
Example	`STAT:QUES:NTR?` → `16384`
Note	

2.5.11 STATus:PORT[:EVENt]?

Syntax	STATus:PORT[:EVENt]? <port name>
Description	This query returns and clears the port event register.
Parameter	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
Response	<enable> = <BOOLEAN RESPONSE DATA>
Example	`STAT:PORT? 1-PORT1` → `1`
Note	All active sessions has their own register and it is cleared when the session starts. <port name> is the same as the one returned by `INSTrument:PORT:CATalog?`.

2.5.12 STATus:PORT:CONDition?

Syntax	STATus:PORT:CONDition? <port name>
Description	This query returns the port condition register.
Parameter	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
Response	<enable> = <BOOLEAN RESPONSE DATA>
Example	`STAT:PORT:COND? 1-PORT1` → `1`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

2.5.13 STATus:PORT:ENABle

Syntax	STATus:PORT:ENABle <port name>,<enable>
Description	This command sets the enable mask for the port event register.
Parameters	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
	<enable> = <BOOLEAN PROGRAM DATA>
Response	None.
Example	`STAT:PORT:ENAB 1-PORT1,ON`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

Syntax	STATus:PORT:ENABle? <port name>
Description	This query returns the enable mask for the port event register.
Parameter	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
Response	<enable> = <BOOLEAN RESPONSE DATA>
Example	`STAT:PORT:ENAB? 1-PORT1` → `1`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

2.5.14 STATus:PORT:PTRansition

Syntax	STATus:PORT:PTRansition <port name>,<enable>
Description	This command sets the positive transition filter for the port event register.
Parameters	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
	<enable> = <BOOLEAN PROGRAM DATA>
Response	None.
Example	`STAT:PORT:PTR 1-PORT1,ON`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

Syntax	STATus:PORT:PTRansition? <port name>
Description	This query returns the positive transition filter for the port event register.
Parameter	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
Response	<enable> = <BOOLEAN RESPONSE DATA>
Example	`STAT:PORT:PTR? 1-PORT1` → `1`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

2.5.15 STATus:PORT:NTRansition

Syntax	STATus:PORT:NTRansition <port name>,<enable>
Description	This command sets the negative transition filter for the port event register.
Parameters	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
	<enable> = <BOOLEAN PROGRAM DATA>
Response	None.
Example	`STAT:PORT:NTR 1-PORT1,OFF`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

Syntax	STATus:PORT:NTRansition? <port name>
Description	This query returns the negative transition filter for the port event register.
Parameter	<port name> = <CHARACTER PROGRAM DATA> Expression format: Character list
Response	<enable> = <BOOLEAN RESPONSE DATA>
Example	`STAT:PORT:NTR? 1-PORT1` → `0`
Note	<port name> is the same as one of the returned by the INSTrument:PORT:CATalog?

2.5.16 STATus:PRESet

Syntax	STATus:PRESet
Description	For the instrument-dependent status data structures, the PRESet commands sets the enable register to all 1's and the transition filter register to recognize only positive transitions. For the SCPI-mandated status structures (operation, questionable and port status) the PRESet command sets the transition filter registers to recognize only positive transitions and set the enable registers to 0's. This command does not affect either the Status Byte or the Standard Event Status register. PRESet does not clear any of the event registers or any item from the error or event queues.
Parameter	None.
Response	None.
Example	`STAT:PRES`
Note	All active sessions has their own set of registers. This command affects registers in all connected application servers.

2.6 Mass Memory Subsystem Commands

The commands in this section operates on files and directories placed in the following storage areas:

Location	Description
`Internal/`	The internal storage of the Network Master.
`Usb/`	An USB connected storage device. This location is only accessible when a USB storage device is mounted.
`Internal/remote/`	A remote network drive. Refer to the User Manual for information on how to connect to an external storage location. This location is only accessible when external storage is configured and the Network Master is able to connect to it.

Files must be located in one of the locations described in the table above - or a in a sub-directory of one of these.

2.6.1 MMEMory:LOAD

Syntax	MMEMory:LOAD <file>
Description	This command loads a file into the currently selected application server. The file may contain settings only or both settings and results data.
Parameter	<file> = <STRING PROGRAM DATA> The path and name of the file to be loaded.
Response	None
Example	`MMEM:LOAD "Internal/otdr-settings.cfg"`
Note	There must be a connected application server for this command to be recognized as a legal command.
	The application server must be in the idle state and the content of the loaded file must match the application server type.

2.6.2 MMEMory:STORe:STATe

Syntax	MMEMory:STORe:STATe <file>
Description	This command stores the current settings to a file on the instrument.
Parameter	<file> = <STRING PROGRAM DATA> The path and name of the file to store the data.
Response	None
Example	`MMEM:STOR:STAT "Internal/my-otdr-settings.cfg"`
Note	There must be a connected application server for this command to be recognized as a legal command.
	The application server must be in the idle state.

2.6.3 MMEMory:STORe:DATA

Syntax	MMEMory:STORe:DATA <file>
Description	This command stores the current settings and result data to a file on the instrument.
Parameter	<file> = <STRING PROGRAM DATA> The path and name of the file to store the data.
Response	None
Example	`MMEM:STOR:DATA "Usb/my-otdr-trace.sor"`
Note	There must be a connected application server for this command to be recognized as a legal command.
	The application server must be in the idle state.
	The files stored in `Internal/` folder or in its subfolder can be loaded.
	When USB storage device is attached to Network Masterset Usb/ directory to the file path to access the files.

2.6.4 MMEMory:DELete

Syntax	MMEMory:DELete <file>
Description	This command deletes a file.
Parameter	<file> = <STRING PROGRAM DATA> The path to the file to be deleted.
Response	None.
Example	`MMEM:DEL "Internal/report.pdf"`
Note	

2.6.5 MMEMory:DATA?

Syntax	MMEMory:DATA? <file>
Description	This command retrieves a file.
Parameter	<file> = <STRING PROGRAM DATA> The path to the file to be retrieved.
Response	<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> = #<nonzero digit><digits><8 bit data bytes>, where: <nonzero digit> is a single ASCII character in the range of '1'-'9'. It represents the length of <digits> in number of bytes. <digits> is a number of ASCII characters in the range of '0'-'9', which together are a decimal representation of the number of succeeding data bytes.
Example	`MMEM:DATA? "Internal/report.pdf"` → #49137<9137 bytes of binary data>
Note	This command cannot be used together with other commands in a compound command.

2.6.6 MMEMory:COPY

Syntax	MMEMory:COPY <source-file>,<destination-file>
Description	This command copies a file.
Parameter	<source-file> = <STRING PROGRAM DATA> The path to the file to be copied.
	<destination-file> = <STRING PROGRAM DATA> The path to the new file.
Response	None.
Example	`MMEM:COPY "Internal/report.pdf","Usb/report.pdf"`
Note	

2.6.7 MMEMory:MOVE

Syntax	MMEMory:MOVE <old-file>,<new-file>
Description	This command moves or renames a file.
Parameter	<old-file> = <STRING PROGRAM DATA> The path to the file to be moved or renamed.
	<new-file> = <STRING PROGRAM DATA> The new path to the file.
Response	None.
Example	`MMEM:MOVE "Internal/report.pdf","Usb/report.pdf"`
Note	

2.6.8 MMEMory:INFO?

Syntax	MMEMory:INFO? <file>
Description	This command retrieves information about a file.
Parameter	<file> = <STRING PROGRAM DATA> The path to the file to retrieve file information about.
Response	<file-date-time> = <STRING RESPONSE DATA> Last file modification date.
	<file-size> = <NR1 NUMERIC RESPONSE DATA> The file size in bytes.
Example	`MMEM:INFO? "Internal/report.pdf"` → `"2015-05-29 16:02:20",9137`
Note	

2.6.9 MMEMory:CATalog?

Syntax	MMEMory:CATalog? <directory>[,<pattern>]
Description	This command lists the files present in a directory.
Parameters	<directory> = <STRING PROGRAM DATA> The path to the directory to be listed.
	<pattern> = <STRING PROGRAM DATA> An optional case sensitive file name pattern. Wildcard characters are * and ?.
Response	({<item>} + {,}∗) = <EXPRESSION RESPONSE DATA> A list of quoted file and directory names.
Example	`MMEM:CAT? "Internal/reports"` → `("report.pdf","setup.cfg")`
Note	

2.6.10 MMEMory:DCATalog?

Syntax	MMEMory:DCATalog? <directory>
Description	This command lists the sub-directories present in a directory.
Parameter	<directory> = <STRING PROGRAM DATA> The path to the directory to be listed.
Response	({<directory>} + {,}∗) = <EXPRESSION RESPONSE DATA> A list for quoted directory names.
Example	`MMEM:DCAT? "Internal/"` → `("diagnostics","favorites","logs","remote","screens","windowsinstaller")`
Note	

2.6.11 MMEMory:MDIRectory

Syntax	MMEMory:MDIRectory <directory>
Description	This command makes a new sub-directory.
Parameter	<directory> = <STRING PROGRAM DATA> The path to the directory to be created.
Response	None.
Example	`MMEM:MDIR "Internal/reports"`
Note	

2.6.12 MMEMory:RDIRectory

Syntax	MMEMory:RDIRectory <directory>[,<force>]
Description	This command removes an existing directory.
Parameter	<directory> = <STRING PROGRAM DATA> The path to the directory to be created.
	<force> = <BOOLEAN PROGRAM DATA> If set to `ON` then ALL CONTENTS i.e. files and sub-directories will be deleted.
Response	None.
Example	`MMEM:RDIR "Internal/reports"`
Note	None.

2.6.13 MMEMory:SAVE

Syntax	MMEMory:SAVE <enable>
Description	This command enables or disables auto saving configuration when the application is terminated
Parameter	<enable> = <BOOLEAN PROGRAM DATA> *DEFault = OFF*
Response	None.

Example	`INST:STAR:LAST OTDR-OTDR,1-PORT1`
	`MMEMory:SAVE ON`
	`INST?` → `2`
	`INST:TERM 2`

Note	

Syntax	MMEMory:SAVE?
Description	This query returns whether or not auto saving configuration enabled when the application is terminated .
Parameter	None.
Response	<enable> = <NR1 NUMERIC RESPONSE DATA>
Example	`MMEMory:SAVE?` → `1`
Note	

Chapter 3

Standard OTDR

This chapter describes the commands available for the Standard OTDR application. The remote control commands are unavailable for the FTTA, Construction, and OLTS applications.

3.1 Measurement Conditions

3.1.1 OTDR:SOURce:PORT

Syntax	OTDR:SOURce:PORT <port>
Description	Sets the port for OTDR measurement.
Parameters	<port>=<CHARACTER PROGRAM DATA> SM: Single Mode MM: Multi Mode
Response	None.
Example	`otdr:sour:port SM`
Note	"MM" is available only on MU100021A.

Syntax	OTDR:SOURce:PORT?
Description	Queries the current output port setting for OTDR.
Parameters	None.
Response	<port>=<CHARACTER RESPONSE DATA>
Example	`otdr:sour:port?` → `SM`
Note	

3.1.2 OTDR:SOURce:TESt

Syntax	OTDR:SOURce:TESt <mode>
Description	Sets the measurement mode.
Parameters	<mode>=<CHARACTER PROGRAM DATA> AUTO: Auto Mode MANUAL: Manual Mode
Response	None
Example	`otdr:sour:test AUTO`
Note	

Syntax	OTDR:SOURce:TESt?
Description	Queries the current measurement mode.
Parameters	None.
Response	<mode> = <CHARACTER RESPONSE DATA>
Example	`otdr:sour:test?` → `AUTO`
Note	

3.1.3 OTDR:SOURce:WAVelength:AVAilable?

Syntax	OTDR:SOURce:WAVelength:AVAilable?
Description	Queries the list of available wavelengths of the OTDR module.
Parameters	None.
Response	<wavelength> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:wav:ava?` → `1310, 1550`
Note	

3.1.4 OTDR:SOURce:WAVelength

Syntax	OTDR:SOURce:WAVelength <value>
Description	Sets the wavelength used for the measurement in nanometer (nm) unit. Settable wavelength depends on the module.
Parameters	<wavelength> = <NUMERIC PROGRAM DATA>
Response	None.
Example	`otdr:sour:wav 1310`
Note	Wavelength ALL cannot be selected.

Syntax	OTDR:SOURce:WAVelength?
Description	Queries the current wavelength setting.
Parameter	None.
Response	<wavelength> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:wav?` → `1310`
Note	The first wavelength will be returned when Wavelength ALL is set.

3.1.5 OTDR:SOURce:RANge:AVAilable?

Syntax	OTDR:SOURce:RANge:AVAilable?
Description	Queries the list of available distance ranges (km) for the current wavelength settings.
Parameters	None.
Response	<range> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:ran:ava?` → `5.0, 10.0, 20.0, 50.0, 100.0, 200.0, 300.0`
Note	

3.1.6 OTDR:SOURce:RANge

Syntax	OTDR:SOURce:RANge <range>
Description	Sets the distance range (km) of the measurement. One of the available distance ranges on the current wavelength settings can be set.
Parameters	<range> = <NUMERIC PROGRAM DATA>
Response	None.
Example	`otdr:sour:ran 100`
Note	

Syntax	OTDR:SOURce:RANge?
Description	Queries the current distance range (km) setting.
Parameters	None.
Response	<range> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:ran?` → `50.0`
Note	

3.1.7 OTDR:SOURce:RESo:AVAilable?

Syntax	OTDR:SOURce:RESo:AVAilable?
Description	Queries the list of available sampling resolution.
Parameters	None.
Response	<res> = <CHARACTER RESPONSE DATA>
Example	`otdr:sour:res:ava?` → `COARSE, MEDIUM, FINE`
Note	

3.1.8 OTDR:SOURce:RESo

Syntax	OTDR:SOURce:RESo <res>
Description	Sets the resolution of the measurement.
Parameters	<res> = <CHARACTER PROGRAM DATA>
Response	None.
Example	`otdr:sour:res MEDIUM`
Note	

Syntax	OTDR:SOURce:RESo?
Description	Queries the current resolution setting.
Parameters	None.
Response	<res> = <CHARACTER RESPONSE DATA>
Example	`otdr:sour:res?` → `MEDIUM`
Note	

3.1.9 OTDR:SOURce:PULSe:AVAilable?

Syntax	OTDR:SOURce:PULSe:AVAilable?
Description	Queries the list of available pulse width (ns).
Parameters	None.
Response	<width> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:puls:ava?` → `10, 20, 50, 100`
Note	

3.1.10 OTDR:SOURce:PULSe

Syntax	OTDR:SOURce:PULSe <width>
Description	Sets current pulse width.
Parameter	<width> = <NUMERIC PROGRAM DATA>
Response	None.
Example	`otdr:sour:puls 100`
Note	Settable pulse width depends on the distance range and resolution.

Syntax	OTDR:SOURce:PULSe?
Description	Queries the current pulse width (ns) setting.
Parameter	None.
Response	<width> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:puls?` → `100`
Note	

3.1.11 OTDR:SOURce:AVERages:TIMe

Syntax	OTDR:SOURce:AVERages:TIMe <time>
Description	Sets the averaging time (s) used in the manual mode.
Parameter	<time> = <NUMERIC PROGRAM DATA>
Response	None.
Example	`otdr:sour:aver:tim 120`
Note	During the measurement, this command will be ignored.

Syntax	OTDR:SOURce:AVERages:TIMe?
Description	Queries the current averaging time setting.
Parameter	None.
Response	<time> = <NUMERIC RESPONSE DATA>
Example	`otdr:sour:aver:tim?` → `120`
Note	

3.2 IOR/BSC

3.2.1 OTDR:SENSe:FIBer:IOR

Syntax	OTDR:SENSe:FIBer:IOR <ior>
Description	Sets index of refraction.
Parameters	<ior> = <NUMERIC PROGRAM DATA> Range: 1.300000 to 1.700000
Response	None.
Example	`otdr:sens:fib:ior 1.45`
Note	The set value will be applied to the measurement result next time.

Syntax	OTDR:SENSe:FIBer:IOR?
Description	Queries the current index of refraction setting.
Parameter	None.
Response	<ior> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:fib:ior?` → `1.450000`
Note	

3.2.2 OTDR:SENSe:FIBer:BSC

Syntax	OTDR:SENSe:FIBer:BSC <bsc>
Description	Sets backscatter coefficient of the fiber.
Parameter	<bsc> = <NUMERIC PROGRAM DATA> Range: -90.00 to -40.00
Response	None.
Example	`otdr:sens:fib:bsc -83.0`
Note	The set value will be applied to the measurement result next time.

Syntax	OTDR:SENSe:FIBer:BSC?
Description	Queries the current backscatter coefficient setting.
Parameter	None.
Response	<bsc> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:fib:bsc?` → `-83.0`
Note	

3.3 Splittter

3.3.1 OTDR:SOURce:SPLitter

Syntax	OTDR:SOURce:SPLitter <number>[,<splitter1>[,<splitter2>[,<splitter3>]]]
Description	Sets the number of splitters on the fiber and the number of split of each splitter.
Parameter	<number> = <CHARACTER PROGRAM DATA> NONE: No splitter. 1 to 3: Sets splitters of the specified number. DETECT: Detects the number of splitters automatically.
	<splitter1>, <splitter2>, <splitter3>= <CHARACTER PROGRAM DATA> X2: 1×2 X4: 1×4 X8: 1×8 X16: 1×16 X32: 1×32 X64: 1×64 X128: 1×128 AUTO: 1×?? (Detects the number of splits automatically)
Response	None.

Example	`otdr:sour:spl NONE`
	`otdr:sour:spl 1,X2`
	`otdr:sour:spl 2,X4,X8`
	`otdr:sour:spl 3,X4,X8,X16`
	`otdr:sour:spl DETECT`

Note	

Syntax	OTDR:SOURce:SPLitter?
Description	Queries the number of splitters and the number of splits of each splitter.
Parameter	None.
Response	<number>[,<splitter1>[,<splitter2>[,<splitter3>]]]
	<number> = <CHARACTER RESPONSE DATA>
	<splitter1>, <splitter2>, <splitter3>= <CHARACTER RESPONSE DATA>

Example	`otdr:sour:spl?` → `NONE`
	`otdr:sour:spl?` → `1,X2`
	`otdr:sour:spl?` → `2,X4,X8`
	`otdr:sour:spl?` → `3,X4,X8,X16`
	`otdr:sour:spl?` → `DETECT`

Note	

3.4 Status

3.4.1 OTDR:SENSe:AVERages:TIMe?

Syntax	OTDR:SENSe:AVERages:TIMe?
Description	Queries the elapsed time since the measurement start.
Parameter	None.
Response	<time> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:aver:tim?` → `28`
Note	

3.4.2 OTDR:SENSe:TRACe:READY?

Syntax	OTDR:SENSe:TRACe:READY?
Description	Queries if trace data is ready.
Parameter	None.
Response	<status> = <BOOLEAN RESPONSE DATA> 1 = trace data is ready and can be transferred. 0 = no trace data available in the memory.
Example	`otdr:sens:trac:ready?` → `1`
Note	

3.5 Measurement Functions

3.5.1 OTDR:SENSe:CONCheck

Syntax	OTDR:SENSe:CONCheck <value>
Description	Sets whether to perform Connection Check.
Parameter	<value> = <BOOLEAN PROGRAM DATA> 1 = Performs Connection check. 0 = Does not perform Connection check.
Response	None.
Example	`otdr:sens:conc 1`
Note	

Syntax	OTDR:SENSe:CONCheck?
Description	Queries whether to perform Connection Check.
Parameter	None.
Response	<value> = <BOOLEAN RESPONSE DATA> 1 = Performs Connection check. 0 = Does not perform Connection check.
Example	`otdr:sens:conc?` → `1`
Note	

3.5.2 OTDR:CONTinue

Syntax	OTDR:CONTinue
Description	The measurement resumes after performing the Connection Check. The measurement can resume irrespective of the Connection Check results.
Parameters	None.
Response	None.
Example	`otdr:cont`
Note	

3.5.3 OTDR:SENSe:LIVCheck

Syntax	OTDR:SENSe:LIVCheck <value>
Description	Sets whether to perform the live fiber check.
Parameter	<value> = <BOOLEAN PROGRAM DATA> 1 = Performs the live fiber check. 0 = Does not perform the live fiber check.
Response	None.
Example	`otdr:sens:livc 1`
Note	

Syntax	OTDR:SENSe:LIVCheck?
Description	Queries whether to perform the live fiber check.
Parameter	None.
Response	<value> = <BOOLEAN RESPONSE DATA> 1 = Performs the live fiber check. 0 = Does not perform the live fiber check.
Example	`otdr:sens:livc?` → `1`
Note	

3.6 Analysis

3.6.1 OTDR:SENSe:PATCh:LAUnch

Syntax	OTDR:SENSe:PATCh:LAUnch <value>
Description	Sets the patch code for the launch fiber.
Parameter	<value> = <CHARACTER PROGRAM DATA> NONE = Does not set the launch fiber point. EVENT1 = sets the ditance of event1 to starting points of the launch fiber. EVENT2 = sets the ditance of event2 to starting points of the launch fiber. EVENT3 = sets the ditance of event3 to starting points of the launch fiber. <value> = <NR2 NUMERIC PROGRAM DATA> sets the start point of the launch fiber by distance. Range: Within the current distance range.
Response	None.
Example	`OTDR:SENSe:PATCh:LAUnch 10.0`
Note	

Syntax	OTDR:SENSe:PATCh:LAUnch?
Description	Queries the patch code setup for the launch fiber.
Parameter	None.
Response	<value> = <CHARACTER RESPONSE DATA> <value> = <NR2 NUMERIC PROGRAM DATA>
Example	`OTDR:SENSe:PATCh:LAUnch?` → `EVENT2`
Note	

3.6.2 OTDR:SENSe:PATCh:RECeive

Syntax	OTDR:SENSe:PATCh:RECeive <value>
Description	Sets the patch code for the receive fiber.
Parameter	<value> = <CHARACTER PROGRAM DATA> NONE = Does not set the receive fber point. EVENT1 = sets the ditance of event1 to starting points of the receive fiber. EVENT2 = sets the ditance of event2 to starting points of the receive fiber. EVENT3 = sets the ditance of event3 to starting points of the receive fiber. <value> = <NR2 NUMERIC PROGRAM DATA> sets the start point of the launch fiber by distance. Range: Within the current distance range.
Response	None.
Example	`OTDR:SENSe:PATCh:RECeive 10.0`
Note	

Syntax	OTDR:SENSe:PATCh:RECeive?
Description	Queries the patch code setup for the receive fiber.
Parameter	None.
Response	<value> = <CHARACTER RESPONSE DATA> <value> = <NR2 NUMERIC PROGRAM DATA>
Example	`OTDR:SENSe:PATCh:RECeive?` → `EVENT2`
Note	

3.6.3 OTDR:SENSe:ACURsor

Syntax	OTDR:SENSe:ACURsor <value>
Description	Sets the cursor A position (km).
Parameter	<value> = <NUMERIC PROGRAM DATA> Range: Within the range of the setting distance range.
Response	None.
Example	`otdr:sens:acur 20.5`
Note	

Syntax	OTDR:SENSe:ACURsor?
Description	Queries the cursor A position (km).
Parameter	None.
Response	<value> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:acur?` → `20.5`
Note	

3.6.4 OTDR:SENSe:BCURsor

Syntax	OTDR:SENSe:BCURsor <value>
Description	Sets the cursor B position (km).
Parameter	<value> = <NUMERIC PROGRAM DATA> Range: Within the range of the setting distance range.
Response	None.
Example	`otdr:sens:bcur 20.5`
Note	

Syntax	OTDR:SENSe:BCURsor?
Description	Queries the cursor B position (km).
Parameter	None.
Response	<value> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:bcur?` → `20.5`
Note	

3.6.5 OTDR:SENSe:LSALeft

Syntax	OTDR:SENSe:LSALeft <start>,<stop>
Description	Sets start and stop positions (km) for left LSA marker.
Parameter	<start> = <NUMERIC PROGRAM DATA> <stop> = <NUMERIC PROGRAM DATA> Range: -100.0 to 400.0.
Response	None.
Example	`otdr:sens:lsal 0.0, 0.5`
Note	Start value must be less than stop value. If the set value is out of the distance range, the maximum in the distance range will be set to the marker position.

Syntax	OTDR:SENSe:LSALeft?
Description	Queries the start and stop positions (km) for left LSA marker.
Parameter	None.
Response	<start>,<stop> <start> = <NUMERIC RESPONSE DATA> <stop> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:lsal?` → `0.0, 0.5`
Note	

3.6.6 OTDR:SENSe:LSARight

Syntax	OTDR:SENSe:LSARight <start>,<stop>
Description	Sets start and stop positions (km) for right LSA marker.
Parameter	<start> = <NUMERIC PROGRAM DATA> Range: -100.0 to 400.0.
	<stop> = <NUMERIC PROGRAM DATA> Range: -100.0 to 400.0.
Response	None.
Example	`otdr:sens:lsar 0.0, 0.5`
Note	Start value must be less than stop value. If the set value is out of the distance range, the maximum in the distance range will be set to the marker position.

Syntax	OTDR:SENSe:LSARight?
Description	Queries the start and stop positions (km) for right LSA marker.
Parameter	None.
Response	<start>,<stop> <start> = <NUMERIC RESPONSE DATA> <stop> = <NUMERIC RESPONSE DATA>
Example	`otdr:sens:lsar?` → `0.0, 0.5`
Note	

3.6.7 OTDR:SENSe:LOSS:MODE

Syntax	OTDR:SENSe:LOSS:MODE <mode>
Description	Sets the Loss Mode.
Parameters	<mode> = <CHARACTER PROGRAM DATA> SPLICE: Splice Loss TP: 2-Pt Loss TPLSA: 2-Pt LSA DBKM: dB/km Loss DBKMLSA: dB/km LSA TPDBKM: 2-Pt Loss, dB/km ORL: ORL
Response	None.
Example	`otdr:sens:loss:mode SPLICE`
Note	

Syntax	OTDR:SENSe:LOSS:MODE?
Description	Queries the current Loss Mode.
Parameter	None.
Response	<mode> = <CHARACTER RESPONSE DATA>
Example	`otdr:sens:loss:mode?` → `SPLICE`
Note	

3.6.8 OTDR:SENSe:ORL:MODE

Syntax	OTDR:SENSe:ORL:MODE <mode>
Description	Sets ORL Mode.
Parameters	<mode> = <CHARACTER PROGRAM DATA> ACURSOR: Cursor A. ORIGIN: Origin. FULL: Full Trace.
Response	None.
Example	`otdr:sens:orl:mode ACURSOR`
Note	

Syntax	OTDR:SENSe:ORL:MODE?
Description	Queries the current ORL Mode.
Parameter	None.
Response	<mode> = <CHARACTER RESPONSE DATA>
Example	`otdr:sens:orl:mode?` → `ACURSOR`
Note	

3.6.9 OTDR:SENSe:ANALyze:PARameters

Syntax	OTDR:SENSe:ANALyze:PARameters <spliceloss>,<reflectance>,<endloss>,<macrobend>, <splitter1x2>,<splitter1x4>,<splitter1x8>,<splitter1x16>,<splitter1x32>,<splitter1x64>, <splitter1x128>
Description	Sets thresholds for automatic detection.
Parameter	<splice loss> = <NUMERIC PROGRAM DATA> Range: 0.01 to 9.99
	<reflectance> = <NUMERIC PROGRAM DATA> Range: -70.0 to -20.0
	<end loss> = <NUMERIC PROGRAM DATA> Range: 1 to 99
	<Macro Bend> = <NUMERIC PROGRAM DATA> Range: 0.3 to 2.0
	<Splitter Loss(1x2)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x4)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x8)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x16)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x32)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x64)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x128)> = <NUMERIC PROGRAM DATA> Range: 1.0 to 30.0
Response	None.
Example	`otdr:sens:anal:par 0.05,-60.0,3,0.3,4.1,7.0,10.0,13.0,16.0,19.0,22.0`
Note	The analysis will restart automatically if the thresholds have changed.

Syntax	OTDR:SENSe:ANALyze:PARameters?
Description	Queries the current thresholds setting for the automatic detection.
Parameter	None.
Response	<splice loss> = <NUMERIC RESPONSE DATA> Range: 0.01 to 9.99
	<reflectance> = <NUMERIC RESPONSE DATA> Range: -70.0 to -20.0
	<end loss> = <NUMERIC RESPONSE DATA> Range: 1 to 99
	<Macro Bend> = <NUMERIC RESPONSE DATA> Range: 0.3 to 2.0
	<Splitter Loss(1x2)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x4)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x8)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x16)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x32)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x64)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
	<Splitter Loss(1x128)> = <NUMERIC RESPONSE DATA> Range: 1.0 to 30.0
Example	`otdr:sens:anal:par?` → `0.05,-60.0,3,0.3,4.1,7.0,10.0,13.0,16.0,19.0,22.0`
Note	

3.7 TRACE

3.7.1 OTDR:TRACe:PARameters?

Syntax	OTDR:TRACe:PARameters?
Description	Queries the main OTDR parameters used to collect the trace data. <wave>, <range>, <pulse>, <avg>, <reso>, <ior>, <bsc>
Parameter	None.
Response	<wave> = <NUMERIC PROGRAM DATA>
	<range> = <NUMERIC PROGRAM DATA>
	<avg> = <NUMERIC PROGRAM DATA>
	<reso> = <NUMERIC PROGRAM DATA>
	<ior> = <NUMERIC PROGRAM DATA>
	<bsc> = <NUMERIC PROGRAM DATA>
Example	`otdr:trac:par?` → `1310, 16.415554, 50, 6144, 0.656621, 1.467700, -78.500000`
Note	

3.7.2 OTDR:TRACe:ANALyze

Syntax	OTDR:TRACe:ANALyze
Description	Performs analysis on the trace.
Parameter	None.
Response	None.
Example	`otdr:trac:anal`
Note	The analysis cannot be performed during the measurement.

3.7.3 OTDR:TRACe:ANALyze:ORL

Syntax	OTDR:TRACe:ANALyze:ORL
Description	Performs ORL calculations on the trace.
Parameter	None.
Response	None.
Example	`otdr:trac:anal:orl`
Note	The ORL calculations cannot be performed during the measurement.

3.7.4 OTDR:TRACe:MDLOss?

Syntax	OTDR:TRACe:MDLOss?
Description	Queries the trace loss values according to the current loss mode.
Parameter	None.
Response	<loss> = <NUMERIC RESPONSE DATA>
	<dBloss> = <NUMERIC RESPONSE DATA>
	Response data is -99.99 except when Loss mode is 2-pt Loss or dB/km Loss.
Example	`otdr:trac:mdlo?` → `-4.610,-99.99`
Note	This query command will be ignored during the measurement.

3.7.5 OTDR:TRACe:EELOss?

Syntax	OTDR:TRACe:EELOss?
Description	Queries trace end-to-end loss.
Parameter	None.
Response	<loss> = <NUMERIC RESPONSE DATA>
Example	`otdr:trac:eelo?` → `-4.610`
Note	This query command will be ignored during the measurement.

3.7.6 OTDR:TRACe:LOAD:TEXT?

Syntax	OTDR:TRACe:LOAD:TEXT? [<start>[,<end>]]
Description	Queries SOR trace information in the text format.
Parameter	<start> = <NUMERIC RESPONSE DATA> Range: Within the range of the setting distance range (km).
	<end> = <NUMERIC RESPONSE DATA> Range: Within the range of the setting distance range (km).
Response	(<parameters>+,∗) = <EXPRESSION RESPONSE DATA>
Example	`otdr:trac:load:text?` → `#6140479` *//SCPI data size message for binary data transfer* `WL = 1310 nm` *//Wavelength* `FBR = SM` *//Fiber Type* `DR = 5 km` *//Distance Range* `PW = 50 ns` *//Pulse Width* `AVG = 6144` *//Number of hardware averages* `IOR = 1.467700` *//IOR value* `BSC = -78.50` *//BSC value* `DATE = 08/13/09` *//Date of test* `TIME = 10:19 PM` *//Time of test* `MXDB = 64 dB` *//dB Range* `RESO = 0.200 m` *//Resolution value* `DX = 0.20440100621721 m` *//Point spacing* `PTS = 25001` *//Number of data points in the trace* <... *Trace Data* ...> `Events 1` *//Number of events found by analysis* `Dist 1.0505 km` *//Event distance* `Type E` *//Event type* `Loss >3.00 dB` *//Event loss value* `Reflectance N/A` *//Event reflectance value* `dB / km 59.420 dB` *//dB/km Loss value* `Cumulative Loss 1.81 dB` *//Cumulative loss value*
Note	This command will be ignored if the measurement is being executed or the measurement results are not ready.

3.7.7 OTDR:TRACe:HOFFset?

Syntax	OTDR:TRACe:HOFFset?
Description	Queries the horizontal offset (km) for the displayed trace(s).
Parameter	None.
Response	<value> = < NR2 NUMERIC RESPONSE DATA>
Example	`otdr:trac:hoff?` → `-0.234`
Note	

Chapter 4

Measurement

4.1 Application, Start and Stop

4.1.1 MEASurement:APPLication?

Syntax	MEASurement:APPLication?
Description	This query returns the application server type.
Parameter	None.
Response	<application> = <CHARACTER RESPONSE DATA> OTDR-OTDR: Standard OTDR application.
Example	`MEAS:APPL?` → `OTDR-OTDR`
Note	

4.1.2 MEASurement:STARt

Syntax	MEASurement:STARt
Description	This command starts a measurement. It operates the same as pressing the START button on the GUI.
Parameter	None
Response	None.
Example	`MEAS:STAR`
Note	

4.1.3 MEASurement:STOP

Syntax	MEASurement:STOP
Description	This command stops an ongoing measurement. It operates the same as pressing the STOP button on the GUI.
Parameter	None.
Response	None.
Example	`MEAS:STOP`
Note	

4.1.4 MEASurement:RESult:SUMMary?

Syntax	MEASurement:RESult:SUMMary?
Description	Queries the summary of the test result.
Parameter	None.
Response	Possible responses are: PASS = Passed. FAIL = Failed.
Example	`meas:res:summ?` → `PASS`
Errors	

Appendix A

Example Scripts

This chapter shows various example scripts for all interfaces which are remote controllable.

A.1 Hints

To ensure that the instrument always start from a well defined state, it is in general a good idea to begin all scripts with the following command. It will terminate all application servers (virtual instruments).

```
*RST
```

A.2 OTDR Test

This example runs an OTDR test. It requies to connect a single mode optical fibre to SM port. Thie examle performs an auto mode test with wavelength of 1310nm and stores the trace data.

```
*RST
INST:STAR OTDR-OTDR,1-PORT1
SYST:WAIT:IDLE

OTDR:SOUR:PORT SM
OTDR:SOUR:TES AUTO
OTDR:SOUR:WAV 1310

MEAS:STAR
SYST:WAIT:IDLE

OTDR:SENS:TRAC:READY?
MMEM:STOR:DATA ""Usb/my-otdr-trace.sor""

SYST:ERR?
INST:TERM
```