DEWETRON

OXYGEN SCPI

Technical Reference Document
Version 1.26 – OXYGEN 7.1

WELCOME TO THE WORLD OF DEWETRON!

Congratulations on your new device! It will supply you with accurate, complete and reproducible measurement results for your decision making.
Look forward to the easy handling and the flexible and modular use of your DEWETRON product and draw upon more than 25 years of DEWETRON expertise in measurement engineering.

ISO9001

EN ISO 14001

THE MEASURABLE DIFFERENCE.

CONTENTS

CHAPTER

ONE

INTRODUCTION

This Technical Reference Document describes the Standard Commands for Programmable Instruments (SCPI) remote control interface to communicate with the DEWETRON Oxygen Software (OXYGEN). The intended audience of this document are instrument programmers who are responsible for writing SCPI-based programs to control the OXYGEN software product.

1.1 Scope

This document describes a set of SCPI commands and queries usable to programmers of SCPI-based devices and controllers via Ethernet based TCP networking, and interfacing to the OXYGEN software. This document also defines the basic TCP operation parameters necessary for a successful connection attempt to the OXYGEN software.

1.2 Conventions and Structure

Notes provide useful information about the context and awareness of special emphasis. Examples provide overview of real world data transmission.

The organization of this document is as follows to provide you a programmer-friendly guide for communicating with the OXYGEN software:

- Chapter “Syntax Conventions” describes the syntax conventions used.
- Chapter “SCPI with OXYGEN” provides information about the TCP networking and an overview of the mapping from the OXYGEN software model to corresponding SCPI systems.
- Chapter “Common Commands” describes the common SCPI commands and queries that are available for the OXYGEN software.
- Chapter “Application Control” describes the SCPI commands and queries that configure basic operation of the OXYGEN software.
- Chapter “Acquisition Control” describes the SCPI commands and queries that act on the acquisition module of the OXYGEN software.
- Chapter “Recording Control” describes the SCPI commands and queries that act on the recording module of the OXYGEN software.
- Chapter “Channel list Access” describes the access to channels and their properties of the OXYGEN softwareChapter “Measurement Values” describes the SCPI commands and queries that act on the retrieval of measurement values of the OXYGEN software.

1.3 Related Documents

Refer to the following documents for more information:

- *OXYGEN Feature Manual*. This document describes the operation of OXYGEN software and its software and related hardware components.
- *OXYGEN Power Technical Reference Manual*. This document describes the operation of OXYGEN software as a highly configurable and most accurate Power Analyzer.
- *Standards Commands for Programmable Instruments (SCPI)*, Volume 1-4, Version 1999.0 May 1999, SCPI Consortium.
- *Standard digital interface for programmable instrumentation – Part 2: Codes, formats, protocols, and common commands*, IEC 60488-2 First Edition 2004-5, IEEE.

CHAPTER

TWO

DOCUMENT HISTORY

Date	Changes	Revision
08/30/2017	First release version of the document.	1.5
10/02/2017	• Added section on document information and history. • Added :NUM:DIM command and query. • Added :NUM:DIMS query. • Updated description of :NUM:VAL query for array channels. • Updated parameter description of :NUM:NORM:ITEM command and query. • Updated explanation of :RATE command for array channels.	1.6
09/29/2018	• Removed loose Reference	1.6.1
10/09/2018	• Added :ELOG commands and queries • Added:DSTream command and queries • Updated ELOG:ITEMs & DSTREAM:ITEMs entries with error handling • Added ELOG limitation info • Added ELOG:PERiod limitation info • Update ELOG supported channels	1.7
02/11/2019	• Added: ELOG new timestamp format	1.8
03/18/2019	• Trigger support for STORE subsystem	1.9
04/28/2020	• Updated documentation	1.11
06/09/2020	• Corrected quotation mark character for simple copy&paste	1.12
09/01/2020	• Added :SETUP:ASYNC:LOAD command for async load configuration from file • Added :SETUP:ASYNC:STATE? query for the async loading state • Added :SETUP:NAME? query for the current setup name	1.13

CHAPTER

THREE

SYNTAX CONVENTIONS

The SCPI (Standard Commands for Programmable Instruments) is a universal ASCII-based textual remote programming language for electronic test and measurement (T&M) instruments. Based on the IEC 60488-2 specification, the remote transportation interface to the OXYGEN software is Ethernet based, as defined by the IEEE 802.3 working group, and the networking protocol is TCP.

Note: Check the reference manual of your DEWETRON measurement device for the availability and configuration options of the Ethernet interface.

The SCPI defines messages in the form of commands and queries to control the operation and functions for T&M instruments. The related topics below describe the syntax of these commands and queries, and the conventions that the OXYGEN software uses to process them. Commands modify settings and parameters of the OXYGEN software. Further, commands tell the OXYGEN software to perform a specific action. Queries cause the OXYGEN software to return data and status information.

Refer to the following table for the symbols used to describe the syntax of commands and queries.

Symbol	Meaning
< >	A defined element
::=	Is defined as
|	Exclusive OR
{ }	Option group; one element is required
[ ]	Optional; can be omitted
…	Previous elements can be repeated

3.1 SCPI Message Structure

The SCPI messages may consist of five element types, defined in the following table.

Element	Meaning
<Header>	This element represents the basic command name. In case if the header ends with a question mark, the command is a query. The header may begin with a colon or asterisk character. A header consists of one or more <Mnemonic> elements, representing the system or subsystem of the command or query category. The only exception
<Mnemonic>	A part of the <Header>. Some commands have only one <Mnemonic> defined by their <Header>. Mnemonics can have a short and a long form. The syntax of the mnemonics in this document is to capitalize the short form while the long form finalizes the <Mnemonic> in lower case characters.
<Parameter>	This is a parameter used as input to the addressed command or query identified by the <Header>.Some commands have no parameters while others have multiple parameters. A <Space> separates parameters from the <Header>, while a <Comma> separates parameters from each other.
<Comma>	A single comma separates parameters of multiple-parameter commands.
<Space>	A white space character separates the <Header> from its <Parameter> list.

Commands have the structure:

```
<Header>[<Space><Parameter>[<Comma><Parameter>]...]
```

Queries have the structure:

```
<Header>?[<Space><Parameter>[<Comma><Parameter>]...]
```

SCPI defines systems and subsystems as a grouping for mnemonics. For example, the mnemonics operating on the acquisition control are summarized within a group :ACQUISITION, containing the mnemonics START, STOP, PAUSE and STATE. Subsystems are additional groupings within a system or a subsystem. Some SCPI systems have no subsystem while others have many subsystems. Not all functions are grouped in systems or subsystems. Therefore, this document uses a logical grouping of function blocks.

A SCPI message can be composed of multiple commands and queries by separating them with a semicolon. The following rules apply when concatenating commands and queries:

1. Separate completely different headers by a semicolon and by the beginning colon on all headers. For example, the command :ACQUISITION:START and the query :STORE:STATE?, can be concatenated into the following single message:
   :ACQUISITION:START; :STORE:STATE?
2. If concatenated headers differ by only the last mnemonics, you can abbreviate the subsequent headers and eliminate the beginning colon. For example, to concatenate the command :ACQUISITION:START and the query :ACQUISITION:STATE?, the following single message can be formed:
   :ACQUISITION:START; STATE?
3. When concatenating multiple queries, a single response message is been generated by concatenating the responses to all queries.

4. The processing order of concatenated commands and queries is the order received.

An <EOM> message terminator must terminate each SCPI message. The OXYGEN software allows the usage of LF and CR LF characters as a valid message termination. The OXYGEN software evaluates only terminated messages.

3.2 Parameter And Response Types

Angle brackets, such as <num>, indicate parameters of commands. Angle brackets containing the parameter type indicate a response type of queries, such as <String>. There are several different types of parameters and response types defined by the SCPI and used by the RC_SCPI of the OXYGEN software, as listed in the following table:

Parameter Type	Description	Example
Arbitrary Block	A block of ASCII raw data described by a simple header of the format: #[1-9][0-9]* The hashtag indicates the beginning of an arbitrary block header. The following digit indicates the number of subsequent digits specifying the length of the data in bytes.	#14AHOI
Arbitrary ASCII String	An unquoted block of ASCII character terminated by a <CR>	Lorem Ipsum
String	Quoted alphanumeric characters, by either a single or a double quote.	“17E24 * 37E87 = BIG NUMBER” ‘Lorem Ipsum‘
Character	Unquoted alphanumeric characters in the format [a-zA-Z][a-zA-Z0-9 _]* The first character must not be a digit.	Test123
Boolean	Boolean numbers or values. The response type returns only boolean numbers 0 and 1.	ON or ≠ 0 OFF or 0
Hexadecimal	Hexadecimal coded integers of the form #Hxxxx	#HFF
Octal	Octal coded integers of the form #Qxxxx	#Q77
Binary	Binary coded integers of the form #Bxxxx	#B1101
NR1	Integers	0, 1, 42, -37
NR2	Decimal numbers 3.432,	-74.43
NR3	Decimal numbers in scientific notation	1.55433E+6
NRf	Flexible decimal numbers that may be of type NR1, NR2 or NR3	See NR1, NR2 and NR3 examples

SCPI defines special numeric values as possible parameters for commands and queries. If mentioned in the specific command or query documentation, you can use the following special character values instead of the numeric values:

Special Character Value	Description	Numeric Value
NAN	Not a Number, as defined in IEEE 754 for 32-bit floating point numbers	9.91 E 37
INFinity NINFinity	Infinity and negative infinity, as defined in IEEE 754 for 32-bit floating point numbers	9.9 E 37 -9.9 E 37
MINimum MAXimum	Denotes the minimum and maximum value of a range of numeric values	For the range 20..100 MIN = 20 MAX = 100
DEFault	The preset value which is set by the *RST command	See NR1, NR2 and NR3 examples

CHAPTER

FOUR

SCPI WITH OXYGEN

The OXYGEN SCPI remote control (RC_SCPI) interface is available via TCP networking using the TCP endpoint port number 10001. You can easily use HyperTerminal or PuTTY in raw TCP configuration and ASCII encoding to test the connection and issue SCPI commands and queries.

Note: The RC_SCPI interface permits only one active client connection to transmit TCP data. Multiple connections are not possible.

The OXYGEN software executes the received SCPI messages in sequential order. Each SCPI command and query is blocking and does not allow any further commands or queries executed until it is finished.

The internal application model of the OXYGEN software consists of major functions, potentially mapped to SCPI systems. The following table gives an overview to those mappings:

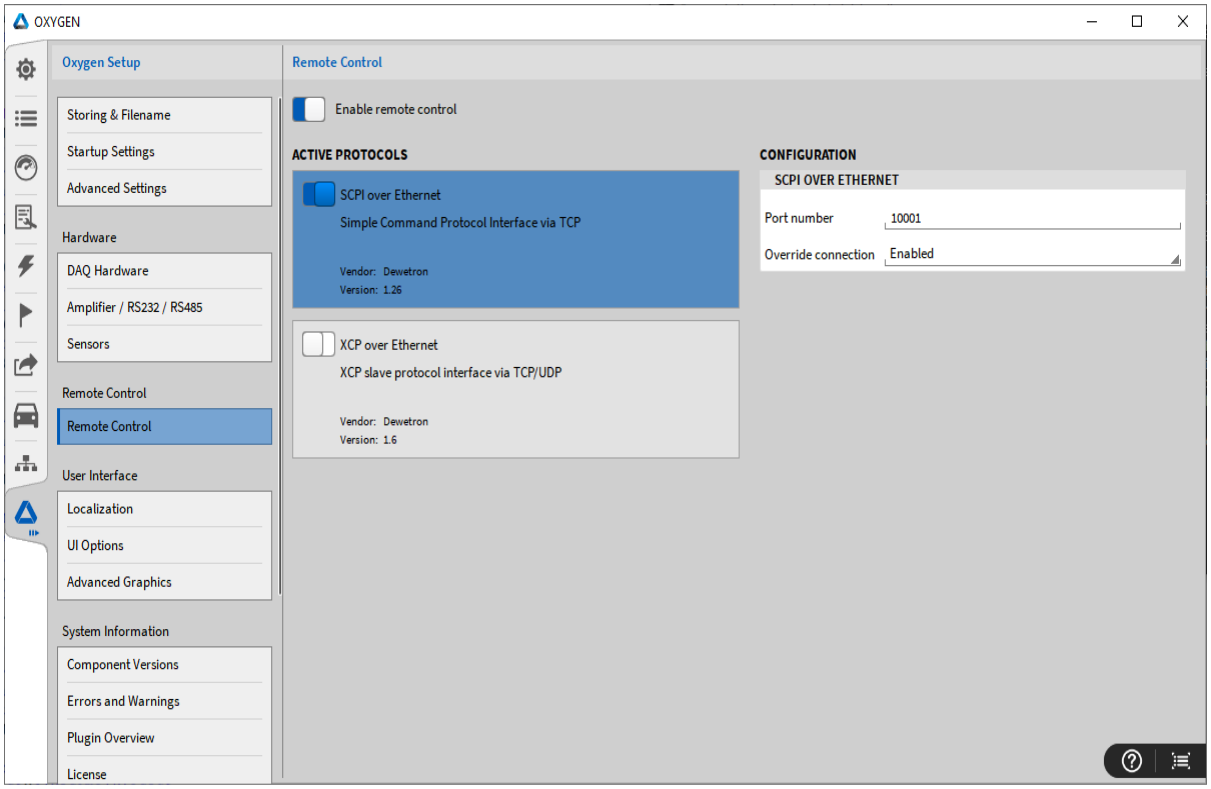
Part of OXYGEN	Description	SCPI Headers and Systems

Application Control	Control the application state and configuration via setups	`:SETup:*`
		`:SYSTEM:*`
		`:COMMUNICATE:*`

Acquisition Control	Control the acquisition state of the application.	`:ACQUISITION:*`
Recording Control	Control the recording state of the application	`:STORE:*`
Measurement Values	Read out actual measurement values.	`:RATE:*` `:NUMERIC:*`
Channel Access	Get channel information and get/set properties.	`:CHANNELlist:*`

4.1 Setup OXYGEN Measurement Software for SCPI use

1. Navigate to System Settings on the Dewetron Measurement device with Oxygen Software
2. Go to “Remote Control” Tab on the left
3. Enable the Remote Control feature and select the protocol type
4. Change the TCP/IP Port number if needed



CHAPTER

FIVE

COMMON COMMANDS

5.1 *IDN?

Syntax	***IDN?**
Description	Query the ID string of the instrument
Parameter	None
Explanation	The query returns a colon-separated four-field ASCII string. The first field contains the manufacturer name, the second field is the product name, the third field is the device serial number, and the fourth field is the product revision number
Return Format	<Arbitrary ASCII String>
Example	`-> *IDN?` `<- DEWETRON,OXYGEN,0,2.5.0`

5.2 *VER?

Syntax	***VER?**
Description	Query the software and SCPI interface version string
Parameter	None
Explanation	The query returns the version information for the relevant parts. The SCPI version, the RC_SCPI plugin version and the OXYGEN version is mandatory and always reported
Return Format	<Character>,<String>[,<Character>,<String>]...
Example	`-> *VER?` `<- SCPI,"1999.0",RC_SCPI,"1.10",OXYGEN,"5.1.1"`

5.3 *CLS

Syntax	*CLS
Description	Clears the standard event register, extended event register, and error queue
Parameter	None
Explanation	If the *CLS command is located immediately after the program message terminator, the output queue is also cleared
Example	`-> *CLS`

5.4 *ESE

Syntax	*ESE <num>
Description	Sets the standard event status enable register
Parameter	<num>: Integer, Range: 0 - 255, Default: 0
Explanation	• Specify the value as a sum of decimal values of each bit • For example, specifying "*ESE 251" will cause the standard enable register to be set to "11111011". In this case, bit 2 of the standard event register is disabled which means that bit 5 (ESB) of the status byte register is not set to 1, even if a "query error" occurs • A query using *ESE? will not clear the contents of the standard event enable register

Example	`-> *ESE 251`
	`-> *ESE?`
	`<- 251`

5.5 *ESE?

Syntax	*ESE?
Description	Queries the current setting of the standard event status enable register
Parameter	None
Explanation	The query returns the content of the standard event status enable register
Return Format	<NR1>
Example	-> *ESE 251 -> *ESE? <- 251

5.6 *ESR?

Syntax	*ESR?
Description	Queries the standard event status register and clears the register
Parameter	None
Explanation	• A sum of decimal values of each bit is returned • You can check what type of events occurred when an SRQ is generated • For example, if a value of “32” is returned, this indicates that the standard event register is set to “00100000.” In this case, you can see that the SRQ occurred due to a “command syntax error.” • A query using *ESR? will clear the contents of the standard event register
Return Format	<NR1>
Example	-> *ESR? <- 32

5.7 *OPC

Syntax	*OPC
Description	Sets bit 0 (OPC bit) of the standard event register to 1 upon the completion of the specified overlap command
Parameter	None
Explanation	Currently all operations are non-overlapped.*OPC sets the OPC bit and *OPC? returns the state of the OPC bit
Example	-> *OPC -> *OPC? <- 1

5.8 *OPC?

Syntax	*OPC?
Description	Queries the state of specified overlapped command
Parameter	None
Explanation	Currently all operations are non-overlapped.*OPC sets the OPC bit
Return Format	<Boolean>
Example	-> *OPC -> *OPC? <- 1

5.9 *RST

Syntax	*RST
Description	Initializes the device to default settings
Parameter	None
Explanation	• Ejects all open data files and switches to live mode • Stops the recording • Restarts the acquisition • Clears the event queue • Resets all settings except communication settings to factory default values
Example	-> *RST

5.10 *SRE

<table>
<tr><td>Syntax</td><td>*SRE <num></td></tr>
<tr><td>Description</td><td>Sets the service request enable register</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Range</td><td>Default</td></tr>
<tr><td><num></td><td>Integer</td><td>0 - 255</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>

Specify the value as a sum of decimal values of each bit
For example, specifying “SRE 239” will cause the *service request enable register to be set to “11101111.” In this case, bit 4 of the service request enable register is disabled which means that bit 4 (MAV) of the status byte register is not set to 1, even if “the output queue is not empty”.
Bit 6 (MSS) of the status byte register is the MSS bit itself, and therefore, is ignored
A query using *SRE? will not clear the contents of the service request enable register

</td></tr>
<tr><td>Example</td><td>
<pre>-> *SRE 239
-> *SRE?
<- 175</pre>
</td></tr>
</table>

5.11 *SRE?

Syntax	***SRE?**
Description	Queries the current setting of the service request enable register
Parameter	None
Explanation	The query returns the content of the service request enable register
Return Format	<NR1>

Example	`-> *SRE 239`
	`-> *SRE?`
	`<- 175`

5.12 *STB?

Syntax	*STB?
Description	Queries the status byte register
Parameter	None
Explanation	**The query returns the content of the status byte register.** • Bits 0, 1, 3, 4 and 7 Not used (always 0) • Bit 2 EAV (Error Available) Set to 1 when the error queue is not empty. In other words, this bit is set to 1 when an error occurs. See the page 7-9. • Bit 5 ESB (Event Summary Bit) Set to 0 when the logical product of the standard event register and the corresponding enable register is 1. In other words, this bit is set to 1, when an event takes place inside the instrument. • Bit 6 RQS (Request Service)/MSS (Master Status Summary) Set to 1 when the logical AND of the status byte excluding Bit 6 and the service request enable register is not 0. In other words, this bit is set to 1 when the instrument is requesting service from the controller. RQS is set to 1 when the MSS bit changes from 0 to 1, and cleared when serial polling is carried out or when the MSS bit changes to 0.
Return Format	<NR1>
Example	`-> *STB?` `<- 4`

5.13 *TST?

Syntax	*TST?
Description	Performs a self-test and queries the result
Parameter	None
Explanation	Currently a self-test is not defined and this query is no-op.
Return Format	<NR1>
Example	`-> *TST?` `<- 0`

5.14 *WAI

Syntax	*WAI
Description	Holds the subsequent command until the completion of the asynchronous operation
Parameter	None
Explanation	Currently all operations are synchronous operations
Example	`-> *WAI`

CHAPTER

SIX

APPLICATION CONTROL

6.1 Setup

6.1.1 :SETup:LOAD

<table>
<tr><td>Syntax</td><td>:SETup:LOAD <file_name>|<path></td></tr>
<tr><td>Description</td><td>Load specified setup file of the current configuration directory or an absolute path</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><file_name></td><td>String</td><td>The filename of the setup file in the default directory (data folder)</td><td>None</td></tr>
<tr><td><path></td><td>String</td><td>The absolute path of the setup file to load</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>This command loads a setup (measurement configuration) file from the current configuration location or an absolute path and applies it directly. The <file_name> does not need to include the file extension “.dms”.</td></tr>
<tr><td>Example</td><td>
<pre>-> :SETup:LOAD "setup1.dms"
<- :SETup:LOAD "C:DATAsetup1.dms"</pre>
</td></tr>
</table>

6.1.2 :SETup:APPLY "XML-String"

<table>
<tr><td>Syntax</td><td>:SETup:APPLY #<num_int><num_char><xml_setup></td></tr>
<tr><td>Description</td><td>Upload a measurement setup to device and apply it</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><num_int></td><td>NR1</td><td>Number of Integers following after this Integer 1 to 9</td><td>None</td></tr>
<tr><td><num_char></td><td>NR1</td><td>Number of Characters to be transferred 1 to 999999999</td><td>None</td></tr>
<tr><td><xml_setup></td><td>Arbitrary Data</td><td>Setup data (XML format)</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>This command uploads a measurement setup to the device and applies it after transfer complete. The measurement setup is a XML-String which can be downloaded via :SETUP:READ?.</td></tr>
<tr><td>Example</td><td><code>-> :SETup:APPLY #41234...</code></td></tr>
<tr><td>Related Commands</td><td><code>:SETup:READ?</code></td></tr>
</table>

6.1.3 :SETup:SAVE "Path"

<table>
<tr><td>Syntax</td><td>:SETup:SAVE <file_name>|<path></td></tr>
<tr><td>Description</td><td>Saves to current setup to the specified file in the current configuration directory or to an absolute path</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><file_name></td><td>String</td><td>The filename of the setup file in the default directory (data folder)</td><td>None</td></tr>
<tr><td><path></td><td>String</td><td>The absolute path of the setup file</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>This command loads a setup (measurement configuration) file from the current configuration location or an absolute path and applies it directly. The <file_name> does not need to include the file extension ".dms".</td></tr>
<tr><td>Example</td><td><pre>-> :SETup:SAVE "setup1.dms"
<- :SETup:SAVE "C:DATAsetup1.dms"</pre></td></tr>
</table>

6.1.4 :SETup:READ?

Syntax	:SETup:READ? [<path>]
Description	Download the current measurement setup or, if a path is specified, the measurement setup referenced to path from the device
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><path></td><td>String</td><td>The absolute path of the setup file to read (XML)</td><td>None</td></tr></table>
Explanation	Download the current measurement setup or, if a path is specified, the measurement setup referenced to path from the device.
Return Format	<Arbitrary Data>

Example	`-> :SETup:READ?`
	`<- #43432........`
	`...`
	`-> :SETup:READ? "C:Datatest.dms"`
	`<- #41242........`

Related Commands	`:SETup:LOAD` `:SETup:APPLY`

6.1.5 : SETup:NAMe?

Syntax	:SETup:NAMe?
Description	Get name of the current loaded oxygen setup
Parameter	None
Explanation	This query returns the filename of the current loaded setup file
Return Format	<String> | NONE
Example	`-> :SETup:NAMe?` `<- :SET:NAM "D:/DATA/last.dms"`

6.1.6 :SETup:ASync:LOAD “Path”

<table>
<tr><td>Syntax</td><td>:SETup:ASync:LOAD <file_name>|<path></td></tr>
<tr><td>Description</td><td>Starts loading a specified setup file of the current configuration directory or an absolute path</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><file_name></td><td>String</td><td>The filename of the setup file in the default directory (data folder)</td><td>None</td></tr>
<tr><td><path></td><td>String</td><td>The absolute path of the setup file to load</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>This command asynchronously loads a setup (measurement configuration) file from the current configuration location or an absolute path and applies it. The <file_name> does not need to include the file extension “.dms”.
Since this command only starts loading, other device commands or queries could be invalid or causes problems, when the loading is not finished yet.
Use the query “:SETup:ASync:STATE?” (see next command) to acquire the state of the loading operation.</td></tr>
<tr><td>Example</td><td><pre>-> :SETup:ASync::LOAD "setup1.dms"
<- :SETup:ASync::LOAD "C:DATAsetup1.dms"</pre></td></tr>
</table>

6.1.7 :SETup:ASync:STATe?

Syntax	:SETup:ASync:STATe?
Description	Queries the state of the last async load configuration operation
Parameter	None
Explanation	This command queries the state of the last async config load operation. The state could be • IDLE • LOADing Some execution errors are added, if the load operation is not finished correctly.
Return Format	<String>

Example	`-> :SETup:ASync::STATe?`
	`<- IDLE`
	`-> :SETup:ASync::LOAD "C:DATAsetup1.dms"`
	`-> :SETup:ASync::STATe?`
	`<- LOAD`
	`-> :SETup:ASync::STATe?`
	`<- LOAD`
	`...`
	`-> :SETup:ASync::STATe?`
	`<- IDLE`
	`...`
	`-> :SETup:ASync::LOAD "C:DATAUnknown.dms"`
	`-> :SETup:ASync::STATe?`
	`<- IDLE`
	`-> :SYST:ERR:ALL?`
	`<- -256, "File name not found"`

6.2 UI Control

6.2.1 :SYSTem:DATE?

Syntax	**:SYSTem:DATE?**
Description	Queries the system date
Parameter	None
Explanation	Returns the system date in the form <year>,<month>,<day> • <year>: INTEGER, four digit number • <month>: INTEGER with the range 1 to 12 • <day>: INTEGER with the range 1 to 31
Return Format	<NR1>,<NR1>,<NR1>
Example	`-> :SYSTem:DATE?` `<- 2017,07,25`
Related Commands	`:SYSTem:TIME?` `:SYSTem:TZONE?`

6.2.2 :SYSTem:KLOCK {ON|OFF}

Syntax	**:SYSTem:KLOCK {ON|1|OFF|0}**
Description	Locks/Unlocks the Screen
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>{ON|1|OFF|0}</td><td>Boolean</td><td>ON|1|OFF|0</td><td>None</td></tr></table>
Explanation	Locks/Unlocks the Screen of Oxygen Software
Example	`-> :SYSTem:KLOCK ON` `-> :SYSTem:KLOCK OFF`
Related Commands	`-> :SYSTem:KLOCK?`

6.2.3 :SYSTem:KLOCK?

Syntax	:SYSTem:KLOCK?
Description	Queries the Screen Lock status
Parameter	None
Explanation	Queries the Screen Lock status
Return Format	<Boolean>
Example	-> :SYSTem:KLOCK ON -> :SYSTem:KLOCK? <- 1
Related Commands	-> :SYSTem:KLOCK

6.2.4 :SYSTem:TIME?

Syntax	:SYSTem:TIME?
Description	Queries the system time
Parameter	None
Explanation	Returns the local system time in the form <hour>,<minute>,<second> • <hour>: INTEGER with the range 0 to +23 • <minute>: INTEGER with the range 0 to 59 • <second>: INTEGER with the range 0 to 59.99999… If Acquisition is not running, :SYSTEM:TIME returns the last known Time
Return Format	<Boolean>
Example	-> :SYSTem:TIME? <- 17,31,1.0237
Related Commands	:SYSTem:DATE? :SYSTem:TZONE?

6.2.5 :SYSTem:TZONE?

Syntax	:SYSTem:TZONE?
Description	Queries the system time zone
Parameter	None
Explanation	Returns the system time zone offset in the form <hour>,<minute> • <hour>: INTEGER with the range 0 to +23 • <minute>: INTEGER with the range 0 to 59 When each field is subtracted from the value of the TIME command, the result is the correct universal coordinated time (also known as UCT, Zulu, Greenwich Mean Time). |
Return Format	<NR1>,<NR1>
Example	`-> :SYSTem:TZONE?` `<- 2,0`
Related Commands	`:SYSTem:DATE?` `:SYSTem:TIME?`

6.2.6 :COMMunicate:HEADer {ON|OFF}

Syntax	:COMMunicate:HEADer {ON|1|OFF|0}
Description	Sets whether query responses should generate an output for the header.
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>{ON|1|OFF|0}</td><td>Boolean</td><td>ON|1|OFF|0</td><td>None</td></tr></table>
Explanation	Sets whether query responses include the header.

Example	`-> :COMMunicate:HEADer ON`
	`-> :SYSTem:TIME?`
	`<- :SYSTEM:TIME 15,14,15.7644`

Related Commands	`:COMMunicate:HEADer?` `:COMMUNICATE:VERBOSE`

6.2.7 :COMMunicate:HEADer?

Syntax	:COMMunicate:HEADer?
Description	Queries the communication response header setting
Parameter	None
Explanation	Queries the setting whether the response includes the header or not
Return Format	<Boolean>
Example	-> :COMMunicate:HEADer? <- 1
Related Commands	:COMMunicate:HEADer

6.2.8 :COMMunicate:VERBose {ON|OFF}

Syntax	:COMMunicate:VERBose {ON|1|OFF|0}
Description	Sets whether query responses should generate the long or short form of the header
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>{ON|1|OFF|0}</td><td>Boolean</td><td>ON|1|OFF|0</td><td>None</td></tr></table>
Explanation	Sets whether query responses should generate the long or short form of the header
Example	-> :COMMunicate:HEADer ON -> :COMMunicate:VERBose OFF -> :SYSTem:TIME? <- :SYST:TIME 15,14,15.7644 -> :COMMunicate:VERBose ON -> :SYSTem:TIME? <- :SYSTEM:TIME 15,14,32.7265
Related Commands	:COMMunicate:HEADer

6.2.9 :COMMunicate:VERBose?

Syntax	:COMMunicate:VERBose?
Description	Queries the setting, whether the response header is short or long form
Parameter	None
Explanation	Queries the setting, whether the response header is short or long form
Return Format	<Boolean>
Example	`-> :COMMunicate:VERBose?` `<- 1`
Related Commands	`:COMMunicate:VERBose`

CHAPTER

SEVEN

ACQUISITION CONTROL

7.1 :ACQUisition:STARt

Syntax	**:ACQUisition:STARt**
Description	Starts the oxygen acquisition
Parameter	None
Explanation	Start the acquisition if not already running
Example	`-> :ACQUisition:STARt`

7.2 :ACQUisition:STOP

Syntax	**:ACQUisition:STOP**
Description	Stops the acquisition
Parameter	None
Explanation	Stops the acquisition if not already stopped
Example	`-> :ACQUisition:STOP`

7.3 :ACQUisition:RESTARt

Syntax	**:ACQUisition:RESTARt**
Description	Restarts the acquisition
Parameter	None
Explanation	Stops, then starts the acquisition. If the acquisition was stopped or paused before, it will only be started.
Example	`-> :ACQUisition:RESTARt`

7.4 :ACQUisition:STATe?

Syntax	:ACQUisition:STATe?
Description	Queries the current acquisition state
Parameter	None
Explanation	**Returns the acquisition state as string:** • Stopped • Waiting_for_sync • Started
Return format	<String>

Example	`-> :ACQUisition:STARt`
	`-> ACQUisition:STATe?`
	`<- Started`

CHAPTER

EIGHT

RECORDING CONTROL

8.1 :STORe:FILE:NAME “PATH”

<table>
<tr><td>Syntax</td><td>(1) :STORe:FILE:NAME <filename>
(2) :STORe:FILE:NAME <path></td></tr>
<tr><td>Description</td><td>Set file name or absolute path of the recording file</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><file_name></td><td>String</td><td>The filename of the setup file in the default directory (data folder)</td><td>None</td></tr>
<tr><td><path></td><td>String</td><td>The absolute path of the setup file</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Set File Name or Path of the Recording File. The File will be overwritten on storing start, if the file already exists. If the file extension is not provided (.dmd), it will be appended automatically.</td></tr>
<tr><td>Example</td><td>

```
-> :STORe:FILE:NAME "Testrun_1"
-> :STORe:FILE:NAME "D:/DATA/Testrun_1"
```

</td></tr>
</table>

8.2 :STORe:FILE:NAME?

Syntax	:STORe:FILE:NAME?
Description	Get File Name and Path of the Recording File during recording
Parameter	None
Explanation	Get the File Name and Path of the Recording File while recording is started. If recording is currently stopped, NONE is returned.
Return Format	<String>|NONE
Example	-> :STORe:FILE:NAME? <- :STOR:FILE "D:/DATA/Testrun_1.dmd" -> :STORe:FILE:NAME? <- :STOR:FILE NONE

8.3 :STORe:STARt

Syntax	:STORe:STARt
Description	Start storing operation
Parameter	None
Explanation	Starts the storing operation or arms configured triggers. If the triggers are already armed forces storing start as if a store trigger were encountered. The file will be overwritten if it exists. If the storing operation was paused, the storing operation will be resumed in the same file.
Example	

8.4 :STORe:PAUSE

Syntax	:STORe:PAUSE
Description	Pause storing operation
Parameter	None
Explanation	Pauses the storing operation. Pausing in triggered recording is not supported and the PAUSE command will return an error if triggered recording is configured.
Example	

8.5 :STORe:STOP

Syntax	:STORe:STOP
Description	Stop storing operation
Parameter	None
Explanation	Stops the storing operation or disarms the triggers.
Example	

8.6 :STORe:STATe?

Syntax	:STORe:STATe?
Description	Queries the state of the recording operation {Started|Stopped|Stopping|Paused|Armed}
Parameter	None
Explanation	Returns the recording state: • Stopped (Storing can be started now) • Stopping (Waiting for a previous store to stop) • Paused (A current storing operation is pause) • Started (Storing is started) • Armed (Triggers for storing are armed) |
Return Format	Literal

Example	`-> :STORe:STARt`
	`-> :STORe:STATe?`
	`<- Started`

8.7 WAVEform access

8.8 :STORe:WAVEform:MODE?

Syntax	:STORe:WAVEform:MODE?
Description	Queries the current recording waveform mode {Continuous|Eventbased|Disabled}
Parameter	None
Explanation	Returns the recording waveform mode: • Continuous (Continuous recording can be started now) • EventBased (Eventbased triggering system is active) • Disabled (System is disabled) This is the default query in the waveform group
Return Format	Literal

Example	`-> :STORe:WAVEform:MODE?`
	`<- Eventbased`
	`...`
	`-> :STORe:WAVEform? //default`
	`<- Eventbased`

8.9 :STORe:WAVEform:MODE {Continuous | Eventbased | Disabled}

Syntax	STORe:WAVEform:MODE {Continuous|Eventbased|Disabled}
Description	Set the current waveform mode
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><Waveform Mode></td><td>Literal</td><td>{Continuous|Eventbased|Disabled}</td><td>None</td></tr></table>
Explanation	Sets the current waveform mode. This is the default command in the waveform group

Example	`-> :STORe:WAVEform:MODE Eventbased`
	`...`
	`-> :STORe:WAVEform Eventbased //default`

8.10 :STORe:WAVEform:CONTinuous

Syntax	:STORe:WAVEform:CONTinuous
Description	Sets the waveform mode to continuous
Parameter	None
Explanation	
Example	`-> :STORe:WAVEform:CONTinuous`

8.11 :STORe:WAVEform:EVENtbased

Syntax	:STORe:WAVEform:EVENtbased
Description	Sets the waveform mode to eventbased
Parameter	None
Explanation	Enables the oxy trigger event system
Example	`-> :STORe:WAVEform:EVENtbased`

8.12 STORe:WAVEform:DISabled

Syntax	:STORe:WAVEform:DISabled
Description	Sets the waveform mode to disabled
Parameter	None
Explanation	Disables the oxy trigger event system
Example	`-> :STORe:WAVEform:DISabled`

8.13 :STORe:WAVEform:PREtime?

Syntax	:STORe:WAVEform:PREtime?
Description	Queries the current waveform pretime values
Parameter	None
Explanation	Shows the current state (ON|OFF) and the current value of the waveform pretime
Return Format	<Boolean>,<Nrf>
Example	-> :STORe:WAVEform:PREtime? <- OFF,1.0

8.14 :STORe:WAVEform:PREtime {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}

Syntax	:STORe:WAVEform:PREtime {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}
Description	Sets the waveform pretime
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Pretime state or value</td><td>Literal | Nrf</td><td>ON | OFF | 1–max seconds</td><td>None</td></tr><tr><td>Pretime value</td><td>Nrf | None</td><td>1–max seconds</td><td>None</td></tr></table>
Explanation	Sets the current state and/or value of the waveform pretime. The state should be ON or OFF. The value is numeric in seconds. This command accepts one or two parameters (see examples)
Example	-> :STORe:WAVEform:PREtime OFF -> :STORe:WAVEform:PREtime? <- OFF,0.0 -> :STORe:WAVEform:PREtime 1.0 //sets the␣ ↪pretime enabled and to 1sec -> :STORe:WAVEform:PREtime? <- ON,1.0 -> :STORe:WAVEform:PREtime OFF,0.5 -> :STORe:WAVEform:PREtime? <- OFF,5.0E-1

8.15 :STORe:WAVEform:PAFTer?

Syntax	**:STORe:WAVEform:PAFTer?**
Description	Queries the current pause after values
Parameter	None
Explanation	Shows the current state (ON|OFF) and the current value of the waveform pause after
Return Format	<Boolean>,<Nrf>
Example	`-> :STORe:WAVEform:PAFTer?` `<- OFF,1.0`

8.16 :STORe:WAVE:PAFTer {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}

Syntax	**:STORe:WAVEform:PAFTer {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}**
Description	Sets the waveform pause after
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Pause after state or value</td><td>Literal | Nrf</td><td>ON | OFF | 1–max seconds</td><td>None</td></tr><tr><td>Pause after value</td><td>Nrf | None</td><td>1–max seconds</td><td>None</td></tr></table>
Explanation	Sets the current state and/or value of the waveform pause after. The state should be ON or OFF. The value is numeric in seconds. This command accepts one or two parameters (see examples)

Example	`-> :STORe:WAVEform:PAFTer OFF`
	`-> :STORe:WAVEform:PAFTer?`
	`<- OFF,0.0`
	`-> :STORe:WAVEform:PAFTer 1.0 //sets the pause enabled and to 1sec`
	`-> :STORe:WAVEform:PAFTer?`
	`<- ON,1.0`
	`-> :STORe:WAVEform:PAFTer OFF,0.5`
	`-> :STORe:WAVEform:PAFTer?`
	`<- OFF,5.0E-1`

8.17 :STORe:WAVEform:POSTtime?

Syntax	:STORe:WAVEform:POSTtime?
Description	Queries the current waveform posttime values
Parameter	None
Explanation	Shows the current state (ON|OFF) and the current value of the waveform posttime
Return Format	<Boolean>,<Nrf>
Example	-> :STORe:WAVEform:POSTtime? <- OFF,1.0

8.18 :STORe:WAVEform:POSTtime {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}

Syntax	:STORe:WAVEform:POSTtime {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}
Description	Sets the waveform posttime
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Posttime state or value</td><td>Literal | Nrf</td><td>ON | OFF | 1–max seconds</td><td>None</td></tr><tr><td>Posttime value</td><td>Nrf | None</td><td>1–max seconds</td><td>None</td></tr></table>
Explanation	Sets the current state and/or value of the waveform posttime. The state should be ON or OFF. The value is numeric in seconds. This command accepts one or two parameters (see examples)
Example	-> :STORe:WAVEform:POSTtime OFF -> :STORe:WAVEform:POSTtime? <- OFF,0.0 -> :STORe:WAVEform:POSTtime 1.0 //sets the␣ ↪pretime enabled and to 1sec -> :STORe:WAVEform:POSTtime? <- ON,1.0 -> :STORe:WAVEform:POSTtime OFF,0.6 -> :STORe:WAVEform:POSTtime? <- OFF,6.0E-1

8.19 :STORe:STATIstics?

Syntax	:STORe:STATIstics?
Description	Queries the current recording setup statistics values
Parameter	None
Explanation	Shows the current state (ON|OFF) and the current value of the recording statistics
Return Format	<Boolean>,<Nrf>
Example	-> :STORe:STATIstics? <- OFF,1.0

8.20 :STORe:STATIstics {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}

Syntax	:STORe:STATIstics {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}
Description	Sets the waveform posttime
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Statistics state or value</td><td>Literal | Nrf</td><td>ON | OFF | 1–max seconds</td><td>None</td></tr><tr><td>Statistics value</td><td>Nrf | None</td><td>1–max seconds</td><td>None</td></tr></table>
Explanation	Sets the current state and/or value of the recording setup statistics. The state should be ON or OFF. The value is numeric in seconds. This command accepts one or two parameters (see examples)
Example	-> :STORe:STATIstics OFF -> :STORe:STATIstics? <- OFF,1.0 -> :STORe:STATIstics 1.1 //enables statistcs␣ ↪state -> :STORe:STATIstics? <- ON,1.1 -> :STORe:STATIstics OFF,0.0 -> :STORe:STATIstics? <- OFF,0.0

8.21 :STORe:AUTOStart?

Syntax	:STORe:AUTOStart?
Description	Queries the current start measurement automatically state
Parameter	None
Explanation	Shows the current state (ON|OFF) of start measurement automatically
Return Format	<Boolean>
Example	-> :STORe:AUTOStart? <- OFF

8.22 :STORe:AUTOStart <Boolean>

Syntax	:STORe:AUTOStart {ON|1|OFF|0}
Description	Sets the start measurement automatically state
Parameter	Name: {ON|1|OFF|0} — Type: Boolean — Range: ON|1|OFF|0 — Default: None
Explanation	Sets the start measurement automatically state to enabled or disabled
Example	-> :STORe:AUTOStart ON -> :STORe:AUTOStart? <- ON

8.23 :STORe:RACQuisition?

Syntax	:STORe:RACQuisition?
Description	Queries the current restart acquisition on measurement start state
Parameter	None
Explanation	Shows the current state (ON|OFF) of restart acquisition on measurement start
Return Format	<Boolean>
Example	-> :STORe:RACQuisition? <- OFF

8.24 :STORe:RACQuisition <Boolean>

Syntax	:STORe:RACQuisition {ON|1|OFF|0}
Description	Sets the restart acquisition on measurement start flag
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>{ON|1|OFF|0}</td><td>Boolean</td><td>ON|1|OFF|0</td><td>None</td></tr></table>
Explanation	Sets the restart acquisition on measurement start to enabled or disabled

Example	`-> :STORe:RACQuisition ON`
	`-> :STORe:RACQuisition?`
	`<- ON`

8.25 :STORe:SAFTer?

Syntax	:STORe:WAVEform:SAFTer?
Description	Queries the current stop measurement after values
Parameter	None
Explanation	Shows the current state (ON|OFF) and the stop measurement after value
Return Format	<Boolean>,<Nrf>
Example	`-> :STORe:SAFTer?` `<- OFF,1.0`

8.26 :STORe:SAFTer {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}

<table>
<tr><td>Syntax</td><td>:STORe:SAFTer {ON|OFF|<Nrf>} | {{ON|OFF},<Nrf>}</td></tr>
<tr><td>Description</td><td>Sets the current stop measurement after state and/or value</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>Stop measurement after state or value</td><td>Literal | Nrf</td><td>ON | OFF | 1–max seconds</td><td>None</td></tr>
<tr><td>Stop measurement after value</td><td>Nrf | None</td><td>1–max seconds</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Sets the current state and/or value of stop measurement after. The state should be ON or OFF. The value is numeric in seconds. This command accepts one or two parameters (see examples)</td></tr>
<tr><td>Example</td><td>

```
-> :STORe:SAFTer OFF
-> :STORe:SAFTer?
<- OFF,1.0
-> :STORe:SAFTer 2.0          //sets the pretime␣
↪enabled and to 1sec
-> :STORe:SAFTer?
<- ON,2.0
-> :STORe:SAFTer OFF,9.0
-> :STORe:SAFTer?
<- OFF,9.0
```

</td></tr>
</table>

8.27 :STORe:ADVanced?

<table>
<tr><td>Syntax</td><td>STORe:ADVanced?</td></tr>
<tr><td>Description</td><td>Queries the current state of the individual channel configuration</td></tr>
<tr><td>Parameter</td><td>None</td></tr>
<tr><td>Explanation</td><td>Shows the current state (ON|OFF) of the individual channel configuration</td></tr>
<tr><td>Return Format</td><td><Boolean></td></tr>
<tr><td>Example</td><td>

```
-> :STORe:ADVanced?
<-  OFF
```

</td></tr>
</table>

8.28 :STORe:ADVanced <Boolean>

<table>
<tr><td>Syntax</td><td>:STORe:ADVanced {ON|1|OFF|0}</td></tr>
<tr><td>Description</td><td>Sets the individual channel configuration state</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Range</td><td>Default</td></tr>
<tr><td>{ON|1|OFF|0}</td><td>Boolean</td><td>ON|1|OFF|0</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Sets the individual channel configuration to enabled or disabled</td></tr>
<tr><td>Example</td><td>

```
-> :STORe:ADVanced ON
-> :STORe:ADVanced?
<- ON
```

</td></tr>
</table>

CHAPTER

NINE

CHANNELLIST ACCESS

9.1 :CHANNELlist:NAMes?

Syntax	**:CHANNELlist:NAMes?**
Description	Queries the long *name* and *ids* of all oxygenchannels
Parameter	None
Explanation	Returns a list of elements alternating the numeric *channel-id* and the *channel-name*
Return Format	(<String >, <String>)[,(< String >, <String>),...) | NONE

Example	`-> :CHANNEL:NAMes?`
	`<- ("577079996393193473 2","AI 2/1 Sim"),`
	`↪("577079996393193473 3","AI 2/2 Sim")`

9.2 :CHANNELlist:IDs?

<table>
<tr><td>Syntax</td><td>:CHANNELlist:IDs? [<channelname>[,<channelname>[,...]]]</td></tr>
<tr><td>Description</td><td>Queries the ids of all or selected channels. For selection use the channels long name as parameter.</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><Channel-Name></td><td>ASCII string</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Returns a list of channel-ids.</td></tr>
<tr><td>Return Format</td><td>(<String>[,<String >, ...]) | NONE</td></tr>
<tr><td>Example</td><td>

```
-> :CHANNEL:IDs?
<- "5770799963931934732","5770799963931934733"
...
-> :CHANNEL:ID? "AI 2/6 Sim"
<- :CHANNEL:ID "93103477960684339 46"
...
-> :CHANNEL:IDs? "AI 2/1 Sim","AI 2/2 Sim"
<- :CHANNEL:ID "7565484415439077397",␣
↪"7565484415439077398"
```

</td></tr>
</table>

9.3 :CHANNELlist:ITEM<ChannelID>:ATTR

The following channel config items (properties) should be possible to access (if available for channel type):

Key	Explanation	Type
Neon/Active	Indicates that the selected channel is enabled	Boolean
Neon/LongName	Channel name and device	String
Neon/Name	Channel name	String
Neon/PhysicalScaleFactor	Scale factor for the selected channel	Floating Point
Neon/PhysicalScaleOffset	Scale offset for the selected channel	Floating Point
Neon/Stored	Indicates whether the channel data will be recorded to the datafile	ENUM
Range	Physical measurement range	RANGE
SampleRate	Sample rate (sample frequency) of an oxygen channel	SCALAR
Unit	Physical measurement unit	String
Used	Indicates that the selected channel is enabled	Boolean

You can obtain all possible config items with the query **":CHANNELlist:ITEM<ChannelID>:ATTR:NAMes?"** described below.

Since a single config items describes a different aspect of an oxygen channels, they can have different types. This can be a single value of a fundamental type (Boolean) or a compound of values with different types (SCALAR). See the next table to evaluate oxygen config item types:

Config Item Type	SCPI parameter type	Example
Boolean	<Boolean>	ON or TRUE OFF or FALSE
Signed Integer	Numeric <Nr1> or <Nr2> or <Nr3>	1, -5
Unsigned Integer	Numeric <Nr1> or <Nr2> or <Nr3>	1, 1.0E-5
Floating point	Numeric <Nrf>	3.14, 0.001, 1.0E-5, -5E3
String	<String>	"AI 2/1 Sim"
ENUM	<String> Literal,<String>,<String>	ENUM,"Channel-Stored","Auto"
SCALAR	Extended Numeric <Nrf> <Nrf,>String> Literal,<Numeric>,<String>	0.01Hz 0.01,"Hz" SCALAR,0.0,"ms"
RANGE	Extended Numeric <Nrf>,<Nrf> Literal,<Numeric>,<String>,<Numeric>,<String>	0V,10.0V RANGE,-10.0,"V",10.0,"V"
RATIO (RationalScalar)	<Nr1>,<Nr1>,<String> Literal,<Numeric>,<Numeric>,<String>,	RATIO,1,3,"V"

For receiving the value of a single item use queries **:CHANNELlist:ITEM<ChannelID>:ATTR:VAL?** or **:CHANNELlist:PROPerty?**.

Please note that the spectrum of config items can vary with different channel types. Some of the config items are read only and cannot be changed.

:: test: ":CHANNELlist:ITEM<ChannelID>:ATTR:NAMes?"

9.4 :CHANNELlist:ITEM<ChannelID>:ATTR:NAMes?

Syntax	**:CHANNELlist:ITEM<ChannelID>:ATTR:NAMes?**
Description	Queries the names (keys) of properties (config-items) of a given channel
Parameter	None
Explanation	Returns a list of available attribute names
Return Format	(<String > [,<String >, ...]) | NONE

Example	`-> :CHANNEL:ITEM8316741119689883648:ATTR:NAMes?`
	`<- "ChannelType","DefaultName","Neon/Name","Neon/`
	`↪Stored"`

9.5 :CHANNELlist:ITEM<ChannelID>:ATTR:VAL?

Syntax	**:CHANNELlist:ITEM<ChannelID>:ATTR:VAL? <PropertyName>**
Description	Queries a specific property (config-item) of an channel
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><ChannelID></td><td>Integer</td><td>0-18446744073709551615</td><td>None</td></tr><tr><td><Parameter-Name></td><td>ASCII string</td><td></td><td>None</td></tr></table>
Explanation	Returns the name of the property as string and the value of the property in the most appropriate format
Return Format	ON | OFF | (<String>,<String>) | (<String>,<NRf>,<String>) | (<String>,<NRf>) | (<String>,<NRf>,<NRf>) | (<String>,<NR1>) | (<String>,<NR1>,<NR1>) | (<String>,<NRf>,<String>,<NRf>,<String>) | NONE
Example	`-> :CHANNEL:ITEM5770799963931934732:ATTR:VAL? "Range"` `<- (RANGE,-10.0,"V",10.0,"V")`

9.6 :CHANNELlist:PROPerty?

<table>
<tr><td>Syntax</td><td>:CHANNELlist:PROPerty? <ChannelID>,<PropertyName></td></tr>
<tr><td>Description</td><td>Queries a specific property (config-item) of an oxygen channel</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><ChannelID></td><td>ASCII string</td><td>Oxygen Id</td><td>None</td></tr>
<tr><td><Property-Name></td><td>ASCII string</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Returns the value(s) of the property in the most appropriate format. For finding possible channel IDs you can use the scpi query :CHANNELlist:NAMes? or :CHANNELlist:IDs? To evaluate all possible property names see query ::CHANNELlist:ITEM<ChannelID>:ATTR:NAMes?</td></tr>
<tr><td>Return Format</td><td>ON | OFF | (<String>,<String>) | (<String>,<NRf>,<String>) | (<String>,<NRf>) | (<String>,<NRf>,<NRf>) | (<String>,<NR1>) | (<String>,<NR1>,<NR1>) | (<String>,<NRf>,<String>,<NRf>,<String>) | NONE</td></tr>
<tr><td>Example</td><td><pre>-> :CHANNEL:PROPerty? "5770799963931934732","Range"
<- (RANGE,-10.0,"V",10.0,"V")</pre></td></tr>
</table>

9.7 :CHANNELlist:PROPerty

<table>
<tr><td>Syntax</td><td>:CHANNELlist:PROPerty <ChannelID>,<Property-Name>,<PARAM>[,<PARAM>[,....]]</td></tr>
<tr><td>Description</td><td>Set the value[s] of a specific property (config-item) of a given channel</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><ChannelID></td><td>ASCII string</td><td>Oxygen Id</td><td>None</td></tr>
<tr><td><Property-Name></td><td>ASCII string</td><td></td><td>None</td></tr>
<tr><td><PARAM></td><td>Numeric, ASCII string, Mnemonic</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Set the value[s] of a channel specific config item. The number and the type of the value[s] depends from the type of the config item. For finding possible channel IDs you can use the scpi query :CHANNELlist:NAMes? or :CHANNELlist:IDs? To evaluate all possible property names see query ::CHANNELlist:ITEM<ChannelID>:ATTR:Names? Use the CHANNELlist:CONSTRaint query below to see restrictions for the item Please note that some of the attributes are readonly and therefore cannot be set via scpi.</td></tr>
<tr><td>Example</td><td>

```
-> :CHANNEL:PROP? "15451287354374881280","Unit"
<- :CHANNEL:PROP "V"
-> :CHANNEL:PROP "15451287354374881280","Unit","A"
-> :CHANNEL:PROP? "15451287354374881280","Unit"
<- :CHANNEL:PROP "A"
...
-> :CHANNEL:PROP? "15451287354374881280","Neon/
 ↪Stored"
<- :CHANNEL:PROP (ENUM,"ChannelStored","No")
-> :CHANNEL:PROP "15451287354374881280","Neon/
 ↪Stored","Auto"
-> :CHANNEL:PROP? "15451287354374881280","Neon/
 ↪Stored”
<- :CHANNEL:PROP (ENUM,"ChannelStored","Auto")
-> :CHANNEL:PROP "15451287354374881280","Neon/
 ↪Stored",ENUM,"ChannelStored","No"
-> :CHANNEL:PROP? "15451287354374881280","Neon/
 ↪Stored”
<- :CHANNEL:PROP (ENUM,"ChannelStored","No")
```

</td></tr>
</table>

9.8 :CHANNELlist:CONSTRaint?

Syntax	**:CHANNELlist:CONSTRaint? <ChannelID>,<PropertyName>**
Description	Queries the constraints of a specific property (config-item) of a given channel
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><ChannelID></td><td>ASCII string</td><td>0-18446744073709551615</td><td>None</td></tr><tr><td><Property-Name></td><td>ASCII string</td><td></td><td>None</td></tr></table>
Explanation	Returns the constraints(s) of a channel property in the most appropriate format. For finding possible channel IDs you can use the :CHANNELlist:NAMes? or :CHANNELlist:IDs? query To evaluate all possible property names see query :CHANNELlist:ITEM<ChannelID>:ATTR:Names?
Return Format	(<String>,<String>) | (<String>,<NRf>,<String>) | (<String>,<NRf>) | (<String>,<NRf>,<NRf>) | (<String>,<NR1>) | (<String>,<NR1>,<NR1>) | (<String>,<NRf>,<String>,<NRf>,<String>) | NONE
Example	`-> :CHANNEL:CONSTR? "334870778933673986 1","InputType"` `<- :CHANNEL:CONSTR (STRING,"SE"),(STRING,"DIFF")`

9.9 :CHANNELlist:ITEM<ChannelID>:ACTion:ZERO

Syntax	**:CHANNELlist:ITEM<ChannelID>:ACTion:ZERO**
Description	Calls the zero action of the given oxygen channel
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><ChannelID></td><td>Integer</td><td>0-18446744073709551615</td><td>None</td></tr></table>
Explanation	The zero action resets the offset of the linear scaling function
Example	`-> :CHANNEL:ITEM5770799963931934732:ACTION:ZERO`

9.10 :CHANNELlist:ACTion:ZERO

<table>
<tr><td>Syntax</td><td>:CHANNELlist:ACTion:ZERO <channelname_or_id>[,<channelname_or_id>[,...]]</td></tr>
<tr><td>Description</td><td>Calls the zero action for a set of channels</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><channelname_or_id></td><td>ASCII string</td><td>Oxygen channel name or Id</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>The zero action resets the offset of the linear scaling function for a set of channels.</td></tr>
<tr><td>Example</td><td>

```
-> :CHANNELlist:ACTion:ZERO "16823196399452553216",
↪"AI 1/2 Sim"
```

</td></tr>
</table>

9.11 :CHANNELlist:TIMing:HIGHest?

Syntax	**:CHANNELlist:TIMing:HIGHest?**
Description	Queries the highest timing interval from all used channels
Parameter	None
Explanation	Returns the timing interval of the channel with the lowest sample rate in seconds
Return Format	<NRf>|NONE
Example	`-> :CHANNELlist:TIMing:HIGHest?` `<- :CHANNEL:TIM:HIGH 1.0E-3`

9.12 :CHANNELlist:TIMing:LOWest?

Syntax	:CHANNELlist:TIMing:LOWest?
Description	Queries the lowest timing interval from all used channels
Parameter	None
Explanation	Returns the timing interval of the channel with the highest sample rate in seconds
Return Format	<NRf>|NONE
Example	-> :CHANNELlist:TIMing:LOWest? <- CHANNEL:TIM:LOW 1.0E-4

9.13 :CHANNELlist:SATuration:VALue?

Syntax	:CHANNELlist:SATuration:VALue? {ALL | <ChannelID>}
Description	Queries the saturation of all or a selected oxygen scalar channel
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr><tr><td><ChannelID></td><td>ASCII string</td><td>Oxygen Id</td><td>None</td></tr></table>
Explanation	Returns the saturation in % since acquisition start.
Return Format	<NRf>|NONE
Example	-> :CHANNELlist:IDs? "AI 2/1 Sim" <- :CHANNEL:ID "17770359696083910677" ... -> :CHANNELlist:SATuration:VALue?␣ ↪"17770359696083910677" <- :CHANNEL:SAT:VAL 79.999988079070334 ... -> :CHANNELlist:SATuration:VALue? ALL <- :CHANNEL:SAT:VAL ("40450768817189028 15",1. ↪420911867502358E-2),("40450768817189028 16",97. ↪500000002468752), ("40450768817189028 19",1.420911867502358E-2), ↪("17770359696083910677",79.999988079070334)

9.14 :CHANNELlist:SATuration:RESet

Syntax	:CHANNELlist:SATuration:RESet
Description	Resets the saturation of all oxygen scalar channels
Parameter	None
Explanation	Rejects the saturation value of all oxygen scalar channels and restarts saturation calculation again.
Example	-> :CHANNELlist:IDs? "AI 2/1 Sim" <- :CHANNEL:ID "17770359696083910677" ... -> :CHANNELlist:SATuration:VALue?␣ ↪"17770359696083910677" <- :CHANNEL:SAT:VAL 79.999988079070334 ... -> :CHANNELlist:SATuration:RESet -> :CHANNELlist:SATuration:VALue?␣ ↪"17770359696083910677" <- :CHANNEL:SAT:VAL 10.942228000000001

9.15 :CHANNELlist:FIResponse

<table>
<tr><td>Syntax</td><td>:CHANNELlist:FIResponse? {<ChannelID>[,<size> [,<logarithmic frequency>,<logarithmic amplitude>]]}</td></tr>
<tr><td>Description</td><td>Queries the Finite Impuls Response of a FIR group channel</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><ChannelID></td><td>ASCII string</td><td>Oxygen Id</td><td>None</td></tr>
<tr><td><Size></td><td>Integer</td><td>10 - 2000</td><td>1000</td></tr>
<tr><td><use logarithmic frequency></td><td>Boolean</td><td>ON or TRUE
OFF or FALSE</td><td>TRUE</td></tr>
<tr><td><use logarithmic amplitude (db)></td><td>Boolean</td><td>ON or TRUE
OFF or FALSE</td><td>TRUE</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Returns a size number of frequency/amplitude pairs.</td></tr>
<tr><td>Return Format</td><td><(NR3,NR3),(NR3,NR3),...>|NONE</td></tr>
<tr><td>Example</td><td>

```
-> :CHANNELlist:IDs? "FIR_1"
<- :CHANNEL:ID "66934804265576693 76"
...
-> :CHANNELlist:FIR? "66934804265576693 76"
<- :CHANNEL:FIR (2.5E+2,-9.757726E-3),(3.1473135E+2,
 ↪1.7361388E-2),(3.962233E+2,9.5682509E-2),...
...
-> :CHANNELlist:FIR? "66934804265576693 76",10
<- :CHANNEL:FIR (2.5E+2,-9.757726E-3),(3.1473135E+2,
 ↪1.7361388E-2),(3.962233E+2,9.5682509E-2),...
...
-> :CHANNELlist:FIR? "66934804265576693 76",100,OFF,ON
<- :CHANNEL:FIR (0.0E+0,10.0E-1),(2.5E+1,9.
 ↪9526759E-1),(5.0E+1,9.9768966E-1),(7.5E+1,9.
 ↪9899028E-1),
(1.0E+2,9.9419968E-1),(1.25E+2,1.0003403E+0),(1.5E+2,
 ↪9.9625236E-1),(1.75E+2,9.9527457E-1),
(2.0E+2,1.0019049E+0),(2.25E+2,9.9261725E-1),(2.5E+2,
 ↪9.9887723E-1),(2.75E+2,1.0009722E+0),
(3.0E+2,9.8937606E-1),(3.25E+2,1.005167E+0),(3.5E+2,
 ↪9.9566524E-1),...
...
```

</td></tr>
</table>

CHAPTER

TEN

MEASUREMENT VALUES

10.1 :RATE {<num>[<unit>]|NONE}

<table>
<tr><td>Syntax</td><td>:RATE {<num>[<unit>]|NONE}</td></tr>
<tr><td>Description</td><td>Set numeric data aggregation time for output</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>{<num>[<unit>] | NONE}</td><td>Nrf,Literal</td><td>1–5000ms| 0.001-5s | NONE</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Set numeric data aggregation for output. If “NONE” specified, the aggregation time is off and the last hold value will be returned on :NUM:NORM:VAL?
The resolution is milliseconds.
Since revision 1.6: Data aggregation is implemented for scalar values only. More precisely, the last hold value will be returned for array channels.</td></tr>
<tr><td>Example</td><td><pre>-> :RATE 500ms</pre></td></tr>
</table>

10.2 :RATE?

Syntax	**:RATE?**
Description	Queries the numeric data aggregation time for output
Parameter	
Explanation	The returned numeric is without unit according to the SCPI reference. But, the implicit unit is [s].
Return Format	<NRf>|NONE

Example	`-> :RATE 500ms`
	`-> :RATE?`
	`<- 5.0E-1`

10.3 :NUMeric:NORMal:ITEMS <channel>[,<channel>[,...]]

Syntax	**:NUMeric:NORMal:ITEMS <channel> [,<channel> [,...]]**
Description	Set numeric data output items
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><channel></td><td>ASCII String</td><td>Oxygen Channel Name</td><td>None</td></tr><tr><td>“ABS-TIME”</td><td>ASCII String</td><td>Include Absolute Time Channel “YYYY-MM-DDTHH:mm:ss.sss”</td><td></td></tr><tr><td>“REL-TIME”</td><td>ASCII String</td><td>Include Relative Time Channel <NRf> Seconds since Acquisition Start</td><td></td></tr></table>
Explanation	Set numeric data output items starting from index 1 on directly
Example	`-> :NUMeric:NORMal:ITEMS "Channel-Name1","U1_tRMS@PowerGroup"`

10.4 :NUMeric:NORMal:ITEMS?

Syntax	**:NUMeric:NORMal:ITEMS?**
Description	Query numeric data output items
Parameter	None
Explanation	Get the full list of channels in the numeric data output system
Return Format	String[,[String[,...]]]

Example	`-> :NUMeric:NORMal:ITEMS "ABS-TIME","U1_tRMS@PowerGroup"`
	`-> :NUMeric:NORMal:ITEMS?`
	`-> "ABS-TIME","U1_tRMS@PowerGroup"`

10.5 :NUMeric:NORMAL:ITEM<x> <channel>

<table>
<tr><td>Syntax</td><td>:NUMeric:NORMAL:ITEM<x> <channel></td></tr>
<tr><td>Description</td><td>Set output item to specified channel name or system channel</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><channel></td><td>String</td><td>Oxygen Channel Name or a system channel (“ABS-TIME”, “REL-TIME”)</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td></td></tr>
<tr><td>Example</td><td><pre>-> :NUMeric:NORMAL:ITEM1 "U1_tRMS@PowerGroup"</pre></td></tr>
</table>

10.6 :NUMeric:NORMal:ITEM<x>?

Syntax	**:NUMeric:NORMal:ITEM<x>?**
Description	Query channel name of output item
Parameter	None
Explanation	
Return Format	String

Example	`-> :NUMeric:NORMAL:ITEM1 "U1_tRMS@PowerGroup"`
	`-> :NUMeric:NORMAL:ITEM1?`
	`<- "U1_tRMS@PowerGroup"`

10.7 :NUMeric:NORMal:CLEar {ALL|<NUM>[,<NUM>]}

Syntax	:NUMeric:NORMal:CLEar {ALL|<num>[,<num>]}
Description	Clear (Set to NONE) given items
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><num></td><td>Integer</td><td>1 - 32768</td><td>None</td></tr><tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr></table>
Explanation	Clear (Set to NONE) given items. If “ALL” is specified the hole list is cleared

Example	`-> :NUMeric:NORMal:ITEMS "ABS-TIME","U1_`
	`↪tRMS@PowerGroup" -`
	`-> :NUMeric:NORMal:CLEar ALL`
	`-> :NUMeric:NORMal:ITEMS?`
	`<- 0`

10.8 :NUMeric:NORMal:DELete <NUM>[,<NUM>]

Syntax	:NUMeric:NORMal:DELete {<num>[,<num>]}
Description	Delete Items, shift remaining to left
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><num></td><td>Integer</td><td>1 - 32768</td><td>None</td></tr></table>
Explanation	

Example	`-> :NUMeric:NORMal:ITEMS "ABS-TIME","U1_`
	`↪tRMS@PowerGroup"`
	`-> :NUMeric:NORMal:DELete 1`
	`-> :NUMeric:NORMAL:ITEM1?`
	`<- "U1_tRMS@PowerGroup"`

10.9 :NUMeric:NORMal:NUMber {<num>|ALL}

Syntax	:NUMeric:NORMal:NUMber {<num>|ALL}}
Description	Sets the number of items to be transferred beginning from index 0
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><num></td><td>Integer</td><td>1 - 32768</td><td>15</td></tr><tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr></table>
Explanation	

Example	`-> :NUMeric:NORMal:ITEMS "ABS-TIME","U1_`
	`↪tRMS@PowerGroup"`
	`-> :NUMeric:NORMal:NUMber 2`

10.10 :NUMeric:NORMal:NUMber?

Syntax	:NUMeric:NORMal:NUMber?
Description	Queries the number of items to be transferred
Parameter	None
Explanation	Queries the number of items to be transferred
Return Format	Nr1

Example	`-> :NUMeric:NORMal:ITEMS "ABS-TIME","U1_`
	`↪tRMS@PowerGroup"`
	`-> :NUMeric:NORMal:NUMber 1`
	`-> :NUMeric:NORMal:NUMber?`
	`<- 1`

10.11 :NUMeric:NORMal:DIM<x> { <i_max> | <i_list> | MAX }

<table>
<tr><td>Syntax</td><td>:NUMeric:NORMal:DIM<x> { <i_max> | <i_list> | MAX }</td></tr>
<tr><td>Description</td><td>Selects indices of array channels to be included in output of :NUMeric:NORMal:VALue?</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><i_max></td><td>Integer</td><td>1 - 32768</td><td>None</td></tr>
<tr><td><i_list></td><td>Numeric list</td><td></td><td>None</td></tr>
<tr><td>MAX</td><td>Literal</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>By specifying the single value parameter <I_MAX>, you limit the number of elements of array channels to the first <I_MAX> values.
If you want to include a specific set of indices, you can provide an <I_LIST>.
In order to reset the selected indices to all available indices, use the parameter mnemonic MAX.
If <I_MAX> or entries from <I_LIST> lie outside the range supported by the array channel, only indices in the allowed range will be saved, and an error will be generated.
Any call to this command will clear the effect of a previously issued :NUM:NORM:DIM command for the selected item.
Note that the indexing of array channels starts at 1.</td></tr>
<tr><td>Example</td><td>
<pre>
-> :NUMeric:NORMal:ITEM1 "AI 1/1"
-> :NUMeric:NORMal:ITEM3 "U1_hRMS@PowerGroup"
-> :NUMeric:NORMal:DIM1?
<- 1
-> :NUMeric:NORMal:DIM2?
<- 1
-> :NUMeric:NORMal:DIM3?
<- 128
-> :NUMeric:NORMal:DIM3 10
-> :NUMeric:NORMal:DIM3?
<- 10
-> :NUMeric:NORMal:DIM3 MAX
-> :NUMeric:NORMal:DIM3?
<- 128
-> :NUMeric:NORMal:DIM3 (1:10,50,60:70)
-> :NUMeric:NORMal:DIM3?
<- (1:10,50,60:70)
</pre>
</td></tr>
</table>

10.12 :NUMeric:NORMal:DIM<x>?

Syntax	:NUMeric:NORMal:DIM<x>?
Description	Queries the dimension of the data to be transferred for one item
Parameter	None
Explanation	• for each scalar channel, 1 will be returned; • for each NONE item, 1 will be returned; • for array channels, the output format will vary depending on how the item was set up with :NUM:DIM<X>: – if no upper bound was set, the number of items of the array channel will be returned – if an upper bound was set, that value will be returned – if a range of indices was selected, that range will be returned
Return Format	<NR1>|<NumericList>
Example	Cf. :NUMeric:NORMal:DIM<x> command.

10.13 :NUMeric:NORMal:DIMS?

Syntax	:NUMeric:NORMal:DIMS?
Description	Queries the dimension of the data to be transferred for each item
Parameter	None
Explanation	The list returned will contain one entry for each item in the format used for :NUM:NORM:DIM<x>?
Return Format	[<NR1>|<NumericList>[,…]
Example	Cf. :NUMeric:NORMal:DIM<x> command.

10.14 :NUMeric:NORMal:FORMat {ASCII|BIN_INTEL|BIN_MOTOROLA}

Syntax	:NUMeric:NORMal:FORMat {ASCII|BIN_INTEL|BIN_MOTOROLA}
Description	Sets the requested output format
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Format</td><td>ASCII literal(s)</td><td>• ASCII • BIN_INTEL • BIN_MOTOROLA</td><td>ASCII</td></tr></table>
Explanation	When format is set to ASCII, all numbers are returned in an ASCII format (NR3 for values, timestamps in NR2 format or as Strings) When format is set to BIN_INTEL or BIN_MOTOROLA, all values are returned in a single array of IEEE 754 float32 values (in the corresponding byte order). See *Arbitrary Block* for more information. Values that are not convertible to float32 values are reported as NaN.
Example	`-> :NUMeric:NORMal:FORMat BIN_INTEL`

10.15 :NUMeric:NORMal:FORMat?

Syntax	:NUMeric:NORMal:FORMat?
Description	Queries the currently configured output format
Parameter	None
Explanation	Returns the currently configured output format
Return Format	{ASCII|BIN_INTEL|BIN_MOTOROLA}

Example	`-> :NUMeric:NORMal:FORM ASCII`
	`-> :NUMeric:NORMal:FORM?`
	`<- ASCII`

10.16 :NUMeric:NORMal:VALue? [<NUM>]

<table>
<tr><td>Syntax</td><td>:NUMeric:NORMal:VALue? [<num>]</td></tr>
<tr><td>Description</td><td>Query numeric values of the preset item list</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><num></td><td>Integer</td><td>1…32768: Only return the value of the entry at index <num> in ITEMS.</td><td>Returns a flat list containing the values of all entries in ITEMS.</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>If no index is specified, all values in the list will be returned in a comma separated list (for ASCII format) up to the number of items to be transferred.
Since 1.6: The list returned by this query will be flat, i.e. values for array channels will be included as a sequence of values. Use the :NUM:NORM:DIMS? query to retrieve the number of elements that can be expected for each item.
If an index is specified the value for the item at index is returned, regardless if the index is higher than the number of items to be transferred.
Since 1.20: When the format is BIN_INTEL or BIN_MOTOROLA, instead of comma separated ASCII characters, on block of binary <Arbitrary Block> is returned. The data layout is the same as for ASCII.</td></tr>
<tr><td>Return Format</td><td>{ASCII|BIN_INTEL|BIN_MOTOROLA}</td></tr>
<tr><td>Example</td><td>
<pre>
-> :NUMeric:NORMal:ITEMS "REL-TIME","U1_
↪tRMS@PowerGroup"
-> :NUMeric:NORMal:VALue?
<- 15.2,2.4553
-> :NUMeric:NORMal:VALue? 2
<- 2.4553
-> :NUMeric:NORMal:ITEM3 "U1_fRMS@PowerGroup"
-> :NUMeric:NORMal:VALue? 3
<- 2.3451,-1.2425,...,4.0124
</pre>
</td></tr>
</table>

CHAPTER

ELEVEN

EXTERNAL DATA LOGGING (ELOG)

The ELOG subsystem allows the SCPI user to retrieve synchronized access to multiple channels via statistics calculations.

First, the subsystem needs to be configured. Possible parameters are a channel list, aggregation calculations, aggregation duration as well as result formats and timestamp formats.

After configuration, the user can start the computations and fetch all values. By continuously requesting data records, gap-less readout is possible. Data is kept available inside Oxygen for at least 20 seconds before fetching of old samples is no longer possible.

ELOG cannot be configured during recording or armed triggers state.

11.1 :ELOG:ITEMs <channel>[,<channel>[,...]]

<table>
<tr><td>Syntax</td><td>:ELOG:ITEMs <channel> [,<channel> [,...]]</td></tr>
<tr><td>Description</td><td>Defines the ordered list of requested channels</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><channel></td><td>ASCII String</td><td>Oxygen Channel Name</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>The user can specify a list of multiple scalar channels. The channels need not share the same sample-rate or raw numeric format. If a channel is not compatible, an error is generated, and the channel is not added to the list. If one of the registered channels becomes unused or is changed, the whole dataset will be discarded and the ELOG system will be in an error state. Use :ELOG:STATe? to get more details.
The following channel types are supported: Analog, Counter, Math, Statistics, Filter, Power, AI Async Pad, CAN Signals, GPS-Position/Velocity/Heading/SecondsOfDay/Distance and Acceleration. For proper operation, a minimum sample rate of 5Hz is suggested.</td></tr>
<tr><td>Example</td><td><code>-> :ELOG:ITEMs "U1_tRMS@PowerGroup","AI 1/2"</code></td></tr>
</table>

11.2 :ELOG:ITEMs?

Syntax	:ELOG:ITEMs?
Description	Returns the current list of requested channels
Parameter	None
Explanation	
Return Format	<ASCCI String>[, <ASCII String>[, ...]] | NONE
Example	-> :ELOG:ITEMs? <- "U1_tRMS@PowerGroup","AI 1/2"

11.3 :ELOG:PERiod <Duration>

Syntax	:ELOG:PERiod <Duration>
Description	Sets the duration (in seconds) of the data aggregation time of statistics from source channels
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><duration></td><td>NRf</td><td>> 0</td><td>0.1 seconds</td></tr></table>
Explanation	The duration period is the number of seconds where statistics values such as MIN/AVG/... are collected from the source channels (ITEMS). The minimum duration is limited by the frequency of the source channel. Values below the inverse of the source channel frequency will lead to an error state
Example	Set 2.5Hz (0.4) or 1kHz (0.001) minimum source channel frequency -> :ELOG:PERiod 0.4 ... -> :ELOG:PERiod 0.001

11.4 :ELOG:PERiod?

Syntax	:ELOG:PERiod?
Description	Queries the numeric data aggregation time for output (in seconds)
Parameter	None
Explanation	
Return Format	<NR2>
Example	-> :ELOG:PER 0.5 -> :ELOG:PER? <- 0.5

11.5 :ELOG:CALCulations {AVG|MIN|MAX|RMS}

Syntax	:ELOG:CALCulations {AVG|MIN|MAX|RMS}[,{AVG|MIN|MAX|RMS}[,...]]
Description	Sets a list of requested statistical calculations for all channels
Parameter	

Name	Type	Range	Default
Modes	ASCII literal(s)	• AVG Average values • MIN Minimum values • MAX Maximum values • RMS root-mean-square values	AVG

Explanation	One or multiple calculations are performed for each ITEMS channel. When fetching data, calculations are returned in the same order as requested in CALC.
Example	-> :ELOG:CALCulations AVG,MAX,RMS

11.6 :ELOG:CALCulations?

11.7 :ELOG:PERiod?

Syntax	**:ELOG:PERiod?**
Description	Queries the list of configured calculation modes
Parameter	None
Explanation	
Return Format	{AVG|MIN|MAX|RMS}[,{AVG|MIN|MAX|RMS}[,...]]

Example	`-> :ELOG:CALC MIN,MAX`
	`-> :ELOG:CALC?`
	`<- MIN,MAX`

11.8 :ELOG:FORMat {ASCII|BIN_INTEL|BIN_MOTOROLA}

Syntax	**:ELOG:FORMat {ASCII|BIN_INTEL|BIN_MOTOROLA}**
Description	Sets the requested output format
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Format</td><td>ASCII literal(s)</td><td>• ASCII • BIN_INTEL • BIN_MOTOROLA</td><td>ASCII</td></tr></table>
Explanation	When format is set to ASCII, all numbers are returned in the NR3 format (timestamps in NR2 format) When format is set to BIN_INTEL or BIN_MOTOROLA, all values are returned in IEEE 754 float32 formats (in the corresponding byte order). See FETCH for more details about record layouts
Example	`-> :ELOG:FORM BIN_INTEL`

11.9 :ELOG:FORMat?

Syntax	:ELOG:FORMat?
Description	Queries the list of configured calculation modes
Parameter	None
Explanation	Returns the currently configured output format
Return Format	{ASCII|BIN_INTEL|BIN_MOTOROLA}

Example	`->  :ELOG:FORM?`
	`->  :ELOG:FORM?`
	`<-  ASCII`

11.10 :ELOG:TIMestamp {OFF|REL|ABS|ELOG}

Syntax	:ELOG:TIMestamp {OFF|REL|ABS|ELOG}
Description	Sets the requested timestamp format
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>Mode</td><td>ASCII literal(s)</td><td>• OFF No timestamp • REL Seconds since acquisition start • ABS Absolute timestamp as ISO date/time • ELOG Seconds since the current ELOG session was started</td><td>ASCII</td></tr></table>
Explanation	Enables or disables timestamp reporting when requesting data through FETCH. Relative times (REL) are reported as seconds since acquisition start, ELOG times are seconds since the first sample from the current ELOG session (after calling :ELOG:STARt). Absolute times are only reported in ASCII output format and have the following format: YYYY-MM-DDThh:mm:ss.xxxxxx
Example	`->  :ELOG:TIM REL`

11.11 :ELOG:TIMestamp?

Syntax	:ELOG:TIMestamp?
Description	Queries the currently configured timestamp format
Parameter	None
Explanation	
Return Format	{OFF|REL|ABS|ELOG}
Example	-> :ELOG:TIM REL -> :ELOG:TIM? <- REL

11.12 :ELOG:STARt

Syntax	:ELOG:STARt
Description	Initializes and starts ELOG data gathering. Samples values can be fetched as early as one aggregation period after STARt. After calling STARt, no ELOG configuration is possible.
Parameter	None
Explanation	
Example	-> :ELOG:STARt

11.13 :ELOG:FETCh? [<NUM>]

Syntax	:ELOG:FETCh [<max_records>]
Description	Fetches MAX_RECORDS records or less from the internal ELOG buffer. If no parameter is given, all available records are returned. FETCh is only possible after STARt.
Parameter	

Name	Type	Range	Default
Max_records	NR1	1 .. MAX	MAX

Explanation

Fetch returns values at record level granularity. There is one record for each timestamp. Each record consists of values from all channels (ITEMs) and all calculation modes (CALC). If no data is available, NONE is returned. ERROR indicates an error.
In ASCII mode, the result is a list of NRf values, all values of one record are returned consecutively, possibly prepended by the timestamp value (see example).
In binary mode, for every channel-calculation, a separate Arbitrary Block with up to MAX_RECORDS float32 values is returned. When using timestamps, an Arbitrary Block with float32 timestamp values is returned as the first result.
In the following examples, we assume the following settings, which configure a record of four values:

```
-> :ELOG:ITEMs "CH0", "CH1"
-> :ELOG:CALC AVG, MIN
```

Example

Timestamp	CH0.AVG	CH0.MIN	CH1.AVG	CH1.MIN
0.1	1.5	1	10.5	10
0.2	2.5	2	20.5	20

```
-> :ELOG:FORM ASCII; :ELOG:TIM OFF; :ELOG:STARt
-> :ELOG:FETCh? 2
<- 1.5, 1, 10.5, 10, 2.5, 2, 20.5, 20
-> :ELOG:STOP
...
-> :ELOG:FORM ASCII; :ELOG:TIM REL; :ELOG:STARt
-> :ELOG:FETCh? 2
<- 0.1, 1.5, 1, 10.5, 10, 0.2, 2.5, 2, 20.5, 20
-> :ELOG:STOP
...
-> :ELOG:FORM BIN_INTEL; :ELOG:TIM REL; :ELOG:STARt
-> :ELOG:FETCh? 2
<- {0.1, 0.2}, {1.5, 2.5}, {1, 2}, {10.5, 20.5}, {10,
↪ 20}
-> :ELOG:STOP
```

11.14 :ELOG:STOP

Syntax	:ELOG:STOP
Description	Stops buffering data after a previous STARt call, clears the data buffer and disables fetching.
Parameter	None
Explanation	The configuration is not changed; therefore, data acquisition can be restarted by calling STARt again.
Example	`-> :ELOG:STOP`

11.15 :ELOG:RESet

Syntax	:ELOG:RESet
Description	Resets the ELOG configuration and stops fetching if started with STARt. All parameters are set to their default values and the channel list (ITEMs) is cleared.
Parameter	None
Explanation	Same behavior as when calling *RST, but limited to the ELOG subsystem.
Example	`-> :ELOG:RESet`

11.16 :ELOG:STATe?

Syntax	:ELOG:STATe?
Description	Returns the internal state of the ELOG subsystem. This command is not needed for regular operations but is useful for information purposes. Possible states are: CONFIG, RUNNING and INVALID
Parameter	None
Explanation	The ELOG subsystem uses states internally. Initially, ELOG is in configuration state (CONFIG) which accepts commands like ITEMs, PER, CALC, FORM and TIM. After calling STARt, the subsystem is switched into a data gathering state (RUNNING) which no longer allows to modify the configuration. The configuration state can be entered again by either calling STOP or RESet. While fetching data from Oxygen through ELOG, the measurement configuration in Oxygen must not be changed. If a change of the configuration is detected, ELOG will stop fetching data and an error is reported when calling FETCh. The state query will return the INVALID state in this case
Return Format	ASCII literal
Example	`-> :ELOG:STATe?` `<-  RUNNING`

CHAPTER

TWELVE

DATA STREAMING (DSTREAM)

The DSTREAM subsystem allows the SCPI user to configure and control fast data streaming via a TCP network protocol. This documentation describes the SCPI commands to control the subsystem. Details about the TCP transfer and examples can be found in a separate documentation.

The following SCPI commands can be used to configure and control one or more streaming groups (distinguished via a GRP number). Each group has its own list of channels and a TCP port where data is provided. Each group can be configured and started individually. Some commands provide an ALL option to start/stop all configured groups.

Each streaming group has its own internal state. It can be queried using the STATe? query. The following list describes the allowed operations in each state:

INVALID: An invalid state means the streaming group does not exist, it will be automatically created when calling configuration commands such as ITEMs and PORT

CONFIG: In the configuration state, ITEMs and PORT commands can be used to configure the group. The INIT command will initialize the subsystem.

INITIALIZED: The streaming group is initialized (e.g. has an open TCP port) and is waiting for connections and the STARt command. If a configuration command such as ITEMs or PORT is executed, the state is set back to CONFIG.

RUNNING: The streaming group is actively sending data. Call STOP to enter the INITIALIZED state again.

ERROR: If one of the channels to register is unused, the whole dataset will be discarded and the DSTREAM system will be in an error state. Use *:DSTream:STATe?* to get more details. It can be reset using the RESet command.

12.1 :DSTream:ITEMs[<GRP>] <channel>[,<channel>[,...]]

Syntax	:DSTream:ITEMs[<GRP>] <channel>[,<channel> [,...]]
Description	Defines the ordered list of requested channels for one stream group GRP
Parameter	

Name	Type	Range	Default
<GRP>	NR1	Group number >= 1	1
<channel>	ASCII String	Oxygen Channel Name	None

Explanation	The user can specify a list of multiple channels. If one channel does not exist, is invalid or disabled, no channel is set for the streaming group (see system error log for more details). In addition, the following channel types are not supported: Digital channels, video channels and Rosette group channels. |
Example	Set two channels for stream group 2: AI 1/1 and AI 1/2 `-> :DST:ITEMs2 "AI 1/1", "AI 1/2"`

12.2 :DSTream:ITEMs[<GRP>]?

Syntax	:DSTream:ITEMs[<GRP>]?
Description	Returns the current list of requested channels for one channel group GRP
Parameter	

Name	Type	Range	Default
<GRP>	NR1	Group number >= 1	1

Explanation	Returns NONE if no items are set
Return Format	<ASCCI String>[, <ASCII String>[, ...]] | NONE

Example	Query the items of stream group 2:
	`-> :DST:ITEMs2?`
	`<- "AI 1/1", "AI 1/2"`

12.3 :DSTream:PORT[<GRP>] <PORT>

Syntax	**:DSTream:PORT[<GRP>] <PORT>**
Description	Configures the port number of the TCP server for a streaming group GRP
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr><tr><td><PORT></td><td>NR1</td><td>1 .. 65536</td><td>10003</td></tr></table>
Explanation	The port must be a valid TCP port. It must not be used by any other TCP server on the local system but can be shared by multiple streaming groups of the Oxygen instance.
Example	Set the TCP port for stream group 2 to 10005: `-> :DST:PORT2 10005`

12.4 :DSTream:PORT[<GRP>]?

Syntax	**:DSTream:PORT[<GRP>]?**
Description	Queries the currently configured TCP port
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr></table>
Explanation	Returns the NR1 numeric value of the configured TCP port
Return Format	<Nr1>
Example	`-> :DST:PORT2?` `<- 10005`

12.5 :DSTream:INIT [<GRP> | ALL]

<table>
<tr><td>Syntax</td><td>:DSTream:INIT [<GRP> | ALL]</td></tr>
<tr><td>Description</td><td>Initializes one or all data streaming groups and opens the TCP port</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr>
<tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>The server will listen for new connection on the configured port</td></tr>
<tr><td>Example</td><td>Initializes the streaming group 1:
<code>-> :DST:INIT 1</code></td></tr>
</table>

12.6 :DSTream:STARt [<GRP> | ALL]

<table>
<tr><td>Syntax</td><td>:DSTream:STARt [<GRP> | ALL]</td></tr>
<tr><td>Description</td><td>Starts streaming of one or all data streaming groups</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr>
<tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Only possible after calling INIT for the specified group(s).</td></tr>
<tr><td>Example</td><td>Start streaming for all configured streaming groups:
<code>-> :DST:STARt ALL</code></td></tr>
</table>

12.7 :DSTream:STOP [<GRP> | ALL]

Syntax	**:DSTream:STOP [<GRP> | ALL]**
Description	Stops streaming of one or all data streaming groups
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr><tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr></table>
Explanation	Only possible after calling STARt for the specified group(s). After stopping, the streaming group is in INITIALIZED state.
Example	Stop streaming for all configured streaming groups: `-> :DST:STOP ALL`

12.8 :DSTream:DELete [<GRP> | ALL]

Syntax	**:DSTream:DELete [<GRP> | ALL]**
Description	Deletes a configured streaming group
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr><tr><td>ALL</td><td>Literal</td><td></td><td>None</td></tr></table>
Explanation	The streaming group must be in config state. All ITEMs and port settings are deleted.
Example	Delete streaming group 1: `-> :DST:DEL 1`

12.9 : DSTream:RESet

Syntax	:DSTream:RESet
Description	Resets the data streaming subsystem
Parameter	None
Explanation	Stops the server if started, and resets the configuration to its default settings
Example	`-> :DST:RES`

12.10 : DSTream:STATe[<GRP>]?

Syntax	:DSTream:STATe[<GRP>]?
Description	Queries the internal state of the data streaming subsystem
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr></table>
Explanation	**Possible states are:** • INVALID: Streaming group does not exist • CONFIG: Configuration state (allows settings of ITEMs and PORT) • INITIALIZED: The streaming group has already been initialized • RUNNING: The server is started and cannot be configured anymore • ERROR: An error has occurred; the user needs to call RESet • UNLICENSED: A valid license for the data streaming subsystem was not found
Return Format	<ASCCI String>[, <ASCII String>[, …]] | NONE

Example	Query the state of the streaming group 1 (default value):
	`-> :DST:STATe?`
	`<- CONFIG`

12.11 : DSTream:TRIG[<GRP>] {ON|OFF}

Syntax	**:DSTream:TRIG[<GRP>] {ON|OFF}**
Description	Configure a streaming group to send data only when a trigger is active
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr><tr><td>{ON|OFF}</td><td>Boolean</td><td>{ON|OFF}</td><td>OFF</td></tr></table>
Explanation	When this property is enabled for a streaming group, data will only be streamed during a triggered measurement, while any waveform recording trigger is active.
Example	Enable the triggered mode of streaming group 1 (default value): `-> :DST:TRIG ON`

12.12 : DSTream:TRIG[<GRP>]?

Syntax	**:DSTream:TRIG[<GRP>]?**
Description	Query the triggered property for a given streaming group
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><GRP></td><td>NR1</td><td>Group number >= 1</td><td>1</td></tr></table>
Explanation	**Returns the current state of the triggered property for a stream group, possible values are** • ON (triggered mode is active) • OFF (data is streamed continuously)

Example	Query the triggered mode of streaming group 1 (default value):
	`-> :DST:TRIG?`
	`<- ON`

12.13 : DSTream:REPLAY?

Syntax	:DSTream:REPLAY?
Description	Query the dstream REPLAY state
Parameter	None
Explanation	**Returns the current REPLAY state, possible values are** • LIVE (data streaming as in live mode) • BULK (streaming continuously as fast as possible)
Example	`-> :DST:REPLAY?` `<- LIVE`

12.14 :DSTream:REPLAY {LIVE | BULK}

Syntax	:DSTream:REPLAY {LIVE | BULK}
Description	Sets the datasream REPLAY state
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>{LIVE | BULK}</td><td>Literal</td><td>{LIVE|BULK}</td><td>BULK</td></tr></table>
Explanation	Sets the data streaming REPLAY mode. Possible values are LIVE or BULK

Example	`-> : DSTream:REPLAY BULK`
	`-> : DSTream:REPLAY?`
	`<- BULK`

CHAPTER

THIRTEEN

EXPORT COMMANDS

13.1 :EXPort:DIRectory

Syntax	**:EXPort:DIRectory?**
Description	Queries the current export folder
Parameter	None
Explanation	Returns the current directory of the oxygen export files. If the current folder is empty, the oxygen export default directory is used.
Return Format	String
Example	`-> :EXP:DIR?` `<- :EXP:DIR "d:/temp"`

13.2 :EXPort:DIRectory “path”

Syntax	**:EXPort:DIRectory “path”**
Description	Sets the current export folder
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><path></td><td>ASCII String</td><td></td><td>None</td></tr></table>
Explanation	Sets the oxygen export folder. An error code is set if the input parameter is wrong or ill formed.

Example	`-> :EXP:DIR "c:/temp"`
	`-> :EXP:DIR?`
	`<- :EXP:DIR "c:/temp"`

13.3 :EXPort:AUTO?

Syntax	:EXPort:AUTO?
Description	Queries the auto export flag
Parameter	None
Explanation	Returns the current oxygen export flag.
Return Format	<Boolean>
Example	-> :EXP:AUTO? <- :EXP:AUTO ON

13.4 :EXPort:AUTO {ON|OFF}

Syntax	:EXPort:AUTO {ON|OFF}
Description	Sets the oxygen auto export flag.
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td>switch</td><td>Literal</td><td>• ON: enables auto export • OFF: disables auto export</td><td>None</td></tr></table>
Explanation	Enables/disables the oxygen auto export. When enabled oxygen exports selected data to the export folder, after the recording stop event (see :STORe:STOP command above). An error code is set if input parameter is wrong or ill formed-
Example	-> :EXP:AUTO ON -> :EXP:AUTO? <- :EXP:AUTO ON

CHAPTER

FOURTEEN

MARKER COMMANDS

14.1 :MARKer:ADD <label>[,<description>|<time>|,<description>,<time>]]

Syntax	**:MARKer:ADD label>[,<description>|,<time>|[,<description>,<time>]]**
Description	Add markers to current measurement
Parameter	

Name	Type	Range	Default
<label>	ASCII String		None
<description>	ASCII String		None
<time>	Nr2		Current time

Explanation	Add a single marker at position <TIME> (in seconds from acquisition start) with <LABEL> and description <DESCRIPTION>. If no <TIME>-parameter is specified in the parameter list, the marker is added at the current time.

Example	`-> :MARK:ADD "Label 1","Description of Marker 1"`
	`-> :MARK:ADD "Label 1",0.0`
	`-> :MARK:ADD "Label 1","Description of Marker 1",1.2345`
	`<- 0`

CHAPTER

FIFTEEN

SYNCHRONISATION STATE

15.1 :SYNC:STATe?

Syntax	**::SYNC:STATe?**
Description	Query the current synchronization hardware state
Parameter	None
Explanation	Returned synchronization hardware state • SYNCED: hardware is synced (correspond to the green sync symbol) • SYNCING: hardware is syncing (correspond to the yellow sync symbol) • OUT_OF_SYNC: current state is not synced (correspond to the red sync symbol) • SYNC_UNAVAILABLE: internal sync source is used (correspond to the grey sync symbol) • ERROR: sync validation error or no valid sync hardware found (correspond to the red sync symbol)
Return Format	String
Example	`-> :SYNC:STATe?` `<- :SYNC:STATe "SYNCED"`

CHAPTER

SIXTEEN

MEASUREMENT SCREEN COMMANDS

16.1 :SCReen:INSTRuments:OUTputchannel:STARt

Syntax	**:SCReen:INSTRuments:OUTputchannel:STARt**
Description	Start streaming to analog out channels
Parameter	None
Explanation	Starts streaming on the default stream output instrument if it is in the Paused or Stopped states. The default instrument is the oldest output channel instrument instance, on the measurement screen with the lowest index, that is configured for replay mode.
Example	`-> :SCReen:INSTRuments:OUTputchannel:STARt`

16.2 :SCReen:INSTRuments:OUTputchannel:PAUSE

Syntax	**:SCReen:INSTRuments:OUTputchannel:PAUSE**
Description	Start streaming to analog out channels
Parameter	None
Explanation	Stop sending data on the default stream output instrument that is in the Started state.
Example	`-> :SCReen:INSTRuments:OUTputchannel:PAUSE`

16.3 :SCReen:INSTRuments:OUTputchannel:STOP

Syntax	:SCReen:INSTRuments:OUTputchannel:STOP
Description	Start streaming to analog out channels
Parameter	None
Explanation	Stop sending data on the default stream output instrument that is in the Started state. Then reset the playback cursor to the start position
Example	`-> :SCReen:INSTRuments:OUTputchannel:STOP`

16.4 :SCReen:INSTRuments:OUTputchannel:STATe?

Syntax	:SCReen:INSTRuments:OUTputchannel:STATe?
Description	Query the current state of the output channel instrument.
Parameter	None
Explanation	**Returns the instrument state as a string:** • **Not_found** (no stream output instrument found on the measurement screens) • **Configuration_error** (instrument is not correctly set up and cannot be used) • **Blocked** (another instrument instance is currently playing) • **Started** (the instrument is streaming dmd data) • **Paused** (the instrument does not stream) • **Stopped** (the instrument does not stream, stream will start at the beginning) • **Active** (the instrument continuously sends data; not used for stream output) • **Error**
Return format	<String>

Example	`-> :SCReen:INSTRuments:OUTputchannel:STARt`
	`-> :SCReen:INSTRuments:OUTputchannel:STATe?`
	`<- Started`

16.5 :SCReen:SAVE “PATH”

<table>
<tr><td>Syntax</td><td>::SCReen:SAVE [<path>]</td></tr>
<tr><td>Description</td><td>Save the current screen to a png file</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><path></td><td>String</td><td>A valid filename of the png file. If no absolute path is specified, the file is stored in the oxygen export folder</td><td>A file starting with “screenshot” and with screen number and the current timestamp is generated in the oxygen export folder</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Saves the current oxygen instruments screen to a png file.</td></tr>
<tr><td>Example</td><td>

```
-> :SCReen:SAVE                 //stores the current␣
 ↪screen with number to file (e.g. "c:/export/
 ↪screenshot_1_20230713_120948.png")
...
-> :SCReen:SAVE "xyz"           //stores the current␣
 ↪screen to file (e.g. "c:/export/xyz.png")
...
-> :SCReen:SAVE "c:/temp/xyz" //stores the current␣
 ↪screen to "c:/temp/xyz.png"
```

</td></tr>
</table>

16.6 :SCReen:ITEM<ScreenNumber>:SAVE “PATH”

<table>
<tr><td>Syntax</td><td>::SCReen:ITEM<ScreenNumber>:SAVE [<path>]</td></tr>
<tr><td>Description</td><td>Save screen to a png file</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><path></td><td>String</td><td>A valid filename of the png file. If no absolute path is specified, the file is stored in the oxygen export folder</td><td>A file starting with “screenshot” and with the <ScreenNumber> and the current timestamp is generated in the oxygen export folder</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Saves the oxygen instruments screen with the given <ScreenNumber> to a png file. The <ScreenNumber> should be a valid oxygen instrument screen (1 - 32768)</td></tr>
<tr><td>Example</td><td>

```
-> :SCReen:ITEM1:SAVE                  //stores screen␣
 ↪number 1 to file (e.g. "c:/export/screenshot_1_
 ↪20230713_120948.png")
...
-> :SCReen:ITEM2:SAVE "xyz"            //stores screen␣
 ↪number 2 to file (e.g. "c:/export/xyz.png")
...
-> :SCReen:ITEM3:SAVE "c:/temp/xyz" //stores screen␣
 ↪number 3 to "c:/temp/xyz.png"
```

</td></tr>
</table>

CHAPTER

SEVENTEEN

MEASUREMENT REPORT COMMANDS

17.1 :REPort:SAVE[:ALL] “PATH”

<table>
<tr><td>Syntax</td><td>::REPort:SAVE:ALL [<path>]</td></tr>
<tr><td>Description</td><td>Save all report pages to a pdf file</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><path></td><td>String</td><td>A valid filename of the pdf file. If no absolute path is specified, the file is stored in the oxygen export folder.</td><td>A file starting with “report” and the current timestamp is generated in the oxygen export folder. All oxygen reports are stored.</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Save all possible oxygen reports to a pdf file. “ALL” is the preferred command of scpi group “REPort:SAVe” and does not have to be spelled out</td></tr>
<tr><td>Example</td><td>

```
-> :REPort:SAVE:ALL           //stores all oxygen␣
 ↪report pages to file (e.g. "c:/export/report_
 ↪20230713_120948.pdf")
...
-> :REPort:SAVE              //same command as␣
 ↪:REPort:SAVE:ALL
...
-> :REPort:SAVE:ALL "xyz"     //stores the current␣
 ↪report to file (e.g. "c:/export/xyz.pdf")
...
-> :REPort:SAVE "c:/temp/xyz" //stores the current␣
 ↪report to file "c:/temp/xyz.pdf"
```

</td></tr>
</table>

17.2 :REPort:ITEM<PageNumber>:SAVE “PATH”

<table>
<tr><td>Syntax</td><td>::REPort:ITEM<PageNumber>:SAVE [<path>]</td></tr>
<tr><td>Description</td><td>Save a single report page to a pdf file</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><path></td><td>String</td><td>A valid filename of the pdf file. If no absolute path is specified, the file is stored in the oxygen export folder</td><td>A file starting with “report” and the current timestamp is generated in the oxygen export folder</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Saves the oxygen report with the given <ReportNumber> to a pdf file. The <ReportNumber> should be a valid oxygen report (1 - 32768)</td></tr>
<tr><td>Example</td><td>

```
-> :REPort:ITEM1:SAVE                  //stores page␣
 ↪number 1 to file (e.g. "c:/export/report_20230713_
 ↪120948.pdf")
...
-> :REPort:ITEM2:SAVE "xyz"            //stores page␣
 ↪number 2 to file (e.g. "c:/export/xyz.pdf")
...
-> :REPort:ITEM3:SAVE "c:/temp/xyz" //stores page␣
 ↪number 3 to "c:/temp/xyz.pdf"
```

</td></tr>
</table>

CHAPTER

EIGHTEEN

MEASUREMENT HEADER DATA

With the Measurement Header Data the user can tag stored recordings with specific user defined data lines. The lines are managed via the Oxygen System Setup.

The following SCPI commands allows to retrieve and manipulate the measurement header data lines from the System Setup.

18.1 :HEADer:ADD <key>,<description>

<table>
<tr><td>Syntax</td><td>(1) :HEADer:ADD <key>, <description>
(2) :HEADer:ADD TEXT, <key>, <description>
(3) :HEADer:ADD NUMERIC_CONSTANT, <key>, <VALUE></td></tr>
<tr><td>Description</td><td>Add a new measure header data line</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><KEY></td><td>ASCII String</td><td></td><td>None</td></tr>
<tr><td><DESCRIP-TION></td><td>ASCII String</td><td></td><td>None</td></tr>
<tr><td><VALUE></td><td>Nrf</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Command (1) + (2): Adds a new measurement header data line entry with the specified key
Command (3) Adds a named constant with the specified value. Constants need to have unique names and can only edited while not in a measurement.</td></tr>
<tr><td>Example</td><td>
<pre>
-> :HEAD:GET? "Header 1"
-> :HEAD:ADD "Header 1","Description in Line 1"
-> :HEAD:GET? "Header 1"
<- :HEAD:GET "Description in Line 1"
</pre>
</td></tr>
</table>

18.2 :HEADer:GET? <key>

Syntax	:HEADer:GET? <key>
Description	Query a measurement data line
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><KEY></td><td>ASCII String</td><td></td><td>None</td></tr></table>
Explanation	Returns the content (description) of the header data line with the specified key.
Return Format	String
Example	`-> :HEADer:GET? "Key 1"` `<- :HEAD:GET "HEAD DATA Line 1"`

18.3 :HEADer:KEYs?

Syntax	:HEADer:KEYs?
Description	Query all keys of the measurement header data
Parameter	None
Explanation	Returns a list of all keys (names) of all measurement header data lines.
Return Format	[String[,String[]…]] | None

Example	`-> :HEAD:ADD "Header 1","Description in Line 1"`
	`-> :HEAD:ADD TEXT,"Header 2","Description in Line 2"`
	`-> :HEAD:KEYs?`
	`<- :HEAD:KEY "Header 1","Header 2"`

18.4 :HEADer:SET <key>,<description>

<table>
<tr><td>Syntax</td><td>(1) :HEADer:SET <key>,<description>
(2) :HEADer:SET TEXT, <key>, <description>
(3) :HEADer:SET NUMERIC_CONSTANT, <key>, <VALUE></td></tr>
<tr><td>Description</td><td>Sets a content to a measure header data line</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><KEY></td><td>ASCII String</td><td></td><td>None</td></tr>
<tr><td><DESCRIP-TION></td><td>ASCII String</td><td></td><td>None</td></tr>
<tr><td><VALUE></td><td>Nrf</td><td></td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Command (1) + (2): Set the description value of the measurement header data line with the key <key>.
Command (3) Set the value of the specified constant. Constants need to have unique names and can only edited while not in a measurement.</td></tr>
<tr><td>Example</td><td>
<pre>
-> :HEAD:GET? "Header 1"
<- :HEAD:GET "Description in Line 1”
-> :HEAD:SET "Header 1","A new description in line 1"
-> :HEAD:GET? "Header 1"
<- :HEAD:GET "A new description in line 1"
</pre>
</td></tr>
</table>

18.5 :HEADer:DELete <key>[,<key>[,...]]

Syntax	:HEADer:DELete <key>,<key>,...
Description	Delete measurement header data line(s)
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><KEY></td><td>ASCII String</td><td></td><td>None</td></tr></table>
Explanation	This command allows to delete one or more measurement header data lines. Constants cannot be deleted during measurement.

Example	`-> :HEAD:GET? "Header 1"`
	`<- :HEAD:GET "Description in Line 1”`
	`-> :HEAD:KEYs?`
	`<- :HEAD:KEY "Header 1","Header 2",”Header 3”`
	`-> :HEAD:DELete "Header 1","Header 3"`
	`-> :HEAD:KEYs?`
	`<- :HEAD:KEY "Header 2"`

18.6 :HEADer:VALues?

Syntax	:HEADer:VALues?
Description	Query the measurement header data
Parameter	
Explanation	Returns a list of all key value fields stored in the measurement header data. The third element is the data type, currently either TEXT or NUMERIC_CONSTANT. New elements may be added to the fields in the future depending on the type.
Return Format	(String, String),(String, String),...

Example	`-> :HEAD:VALues?`
	`<- :HEAD:VAL ("Header 1","Description in line 1",`
	`↪TEXT),("Header 2","Description in Line 2",TEXT)`

CHAPTER

NINETEEN

UTILITY COMMANDS

19.1 :SYSTem:VERSion?

Syntax	**:SYSTem:VERSion?**
Description	Queries the implemented SCPI Standard Version
Parameter	None
Explanation	Queries the implemented SCPI Standard Version
Return Format	ASCII String
Example	`-> :SYSTem:VERSion?` `<- "1999.0"`

19.2 :SYSTem:HELP:HEADers?

Syntax	**:SYSTem:HELP:HEADers?**
Description	Queries the implemented commands as list
Parameter	None
Explanation	Queries the implemented commands as list
Return Format	#<num_int><num_char><Arbitrary Data>

Example	`-> :SYSTem:HELP:HEADers?`
	`<- #41063`
	`<- :SYSTem:ERRor:ALL?/qonly/`
	`<- :SYSTem:ERRor:CODE:ALL?/qonly/`
	`<- :SYSTem:ERRor:CODE[:NEXT]?/qonly/`
	`<- :SYSTem:ERRor:COUNt?/qonly/`
	`<- ...`

CHAPTER

TWENTY

TRIGGER EVENTS

Note that commands to set trigger event parameters are not valid if the oxygen is within a measurement.

20.1 :TRIGger[:GET]?

Syntax	**:TRIGger:GET?**
Description	Queries the number, enabled state and the name of all stored trigger events
Parameter	None
Explanation	Returns a list of elements alternating the numeric event-number, the event-enabled state and the event-name. Note: this is the default query in the trigger subsystem (see examples)
Return Format	(<Integer>, <ON|OFF>, <String>)[,(<Integer>, <ON | OFF>, <String>)[,...]] | NONE
Example	-> :TRIGger:GET? <- (1,ON,"Event 1"),(2,ON,"My Event") ... -> :TRIG? //use as default query <- (1,ON,"Event 1"),(2,ON,"My Event")

20.2 :TRIGger:RESet

Syntax	**:TRIGger:RESet**
Description	Delete all trigger events
Parameter	None
Explanation	This command removes all trigger events
Example	-> :TRIGger:RESet -> :TRIGger? <- NONE

20.3 :TRIGger:ADDevent

Syntax	:TRIGger:ADDevent
Description	Add a new trigger event
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><event name></td><td>ASCII String</td><td>optional</td><td>“Event <next event-number>”</td></tr></table>
Explanation	Add a new single trigger event to the trigger subsystem. If no event-name is specified then default name “Event <next event-number>” is assigned

Example	`-> :TRIGger?`
	`<- :NONE`
	`↪ //no Events`
	`-> :TRIGger:ADDevent`
	`-> :TRIGger:ADDevent "My Event"`
	`-> :TRIG?`
	`<- (1,ON,"Event 1"),(2,ON,"My Event")`

20.4 :TRIGger:EVent<context> Commands and Queries

The Oxygen trigger subsystem facilitates one or more trigger events, to control the oxygen recording setup. Each event can be identified by a corresponding event number and an event name. The event number is used as context for several commands and queries and directs each command to the dedicated event (e.g. :TRIGger::EVent<event-number>:…). If no number is specified, event number 1 is assumed.

Each event is constructed with one or more condition(s) and one or more action(s). For each condition and each action a corresponding number is dedicated. These numbers also can be used as context numbers for specific condition and action commands (e.g. :TRIGger::EVent<Number>:CONDition<condition-number>:…).

The following commands and queries can be used to query and setup all available Oxygen trigger events.

20.5 :TRIGger:EVent<event-number>[:SETup]?

Syntax	TRIGger:EVent<event-number>:SETup?
Description	Queries the state, name, conditions and actions of a specific trigger event
Parameter	None
Explanation	Returns the state, name and a list of all conditions and all actions specified for the event with the corresponding event number. Note: this is the default query in the trigger event subsystem (see example).
Return Format	<ON | OFF>,<String>[,(Condition)[,...]] [,(Action)[,...]] | NONE
Example	-> :TRIGger:EVent2:SETup? <- ON,"Event 2" ↪ //Event 2 without any condition or action ... -> :TRIG:EV? ↪ //use as default query, event number 1 is ↪assumed <- ON,"Event 1",(CONDITION,HIGHLEVEL,0.0,OFF, ↪"18864507313434132 48") //no action configured

20.6 :TRIGger:EVent<event-number>[:SETup] {<String>} | {{ON|OFF},<String>}

Syntax	**:TRIGger:EVent<event-number>:SETup {<String>} | {{ON|OFF},<String>}**
Description	Sets the state and/or name and state of the corresponding event
Parameter	

Name	Type	Range	Default
Event state or name	Literal | <String>	ON | OFF | <String>>	None
Event name	<String>		None

Explanation	Sets the current state and/or name of the corresponding event. The state should be ON or OFF. Note: this is the default command in the trigger event subsystem (see example).
Example	

```
-> :TRIGger:EVent2:SETup?
<- OFF,"Event 2"                                    //Event␣
 ↪without any condition or action
-> TRIGger:EVent2:SETup "My Event"                  //set␣
 ↪name and the state enabled
-> :TRIG:EV2?
<- ON,"My Event"
-> :TRIG:EV2 OFF,"My Event 2"
-> :TRIG:EV2?
<- OFF,"My Event 2"
```

20.7 :TRIGger:EVent<event-number>:VALId?

Syntax	**:TRIGger:EVent<event-number>:VALId?**
Description	Queries validation state of the corresponding event
Parameter	None
Explanation	Returns TRUE if the event has a valid configuration. FALSE is returned if the event does not exist, a condition is missing, or any of the conditions and actions are mismatched.
Return Format	TRUE | FALSE
Example	

```
-> :TRIGger:EVent1:VALId?
<- TRUE
-> :TRIG:EV2:VALI?
<- FALSE
```

20.8 :TRIGger:EVent<event-number>:DELete

Syntax	:TRIGger:EVent<event-number>:DELete
Description	Deletes the trigger event with the corresponding event number
Parameter	None
Explanation	Deletes the trigger event with all conditions and actions. Note: All subsequent events are reordered with a new event number. If no event number is specified, event 1 is deleted.
Example	-> :TRIGger? <- (1,ON,"Event 1"),(2,ON,"Event 2"),(3,ON,"Event 3") -> :TRIGger:EVent2:DELete -> :TRIGger? <- (1,ON,"Event 1"),(2,ON,"Event 3") -> :TRIGger:EVent2:DELete -> :TRIGger? <- (1,ON,"Event 1")

20.9 :TRIGger:EVent<event-number>:ADDCondition

Syntax	:TRIGger:EVent<event-number>:ADDCondition
Description	Add a new trigger event condition to the event
Parameter	None
Explanation	Add a new single condition to the corresponding trigger event. If no event-number is specified as context event number 1 is assumed. The constructed condition is a high level condition with its own default parameters. For changing the condition type or parameters see the condition setup commands below.
Example	-> :TRIGger:EVent1? <- ON,"Event 1" -> :TRIGger:EVent1:ADDCondition -> :TRIGger:EVent1? <- ON,"Event 1",(CONDITION,HIGHLEVEL,0.0,OFF) / ↪/event 1 with one high level condition

20.10 :TRIGger:EVent<event-number>:ADDAction

Syntax	:TRIGger:EVent<event-number>:ADDAction
Description	Add a new trigger event action to the event
Parameter	None
Explanation	Add a new single action to the corresponding trigger event. If no event-number is specified as context event number 1 is assumed. The constructed action is a recording action with start recording as default parameter. For changing the action type or parameters see the action setup commands below.
Example	-> :TRIGger:EVent1? <- ON,"Event 1" -> :TRIGger:EVent1:ADDAction -> :TRIGger:EVent1? <- ON,"Event 1",(ACTION,RECORDING,START) //event↵ ↪1 with one recording action

20.11 :TRIGger:EVent<event-number>:CONDition<condition-number>:GET?

Syntax	:TRIGger:EVent<event-number>:CONDition<condition-number>:GET?
Description	Queries the condition type and parameter of the corresponding trigger event condition
Parameter	None
Explanation	Returns the condition type and all of the specific parameters for this type. Note: this is the default query in the trigger event condition subsystem (see sample example).
Return Format	([<condition-type>[,<parameter 1>[,…]]]) | NONE • High level condition: (CONDITION,HIGHLEVEL,<nrf>,<literal>|<nrf>,<String>,<String>, …) • Low level condition: (CONDITION,LOWLEVEL,<nrf>,<literal>|<nrf>,<String>,<String>, …) • In window condition: (CONDITION,INWINDOW,<nrf>,<nrf>,<String>,<String>,…) • Out window condition: (CONDITION,OUTWINDOW,<nrf>,<nrf>,<String>,<String>,…) • Keyboard condition: (CONDITION,KEYBOARD,<literal>,<String>) • Time condition: (CONDITION,TIME,<literal>|<String>,<literal>|<nrf>,<literal>|<nrf>) For a detailed parameter name and description of all trigger event conditions see the condition setup commands below.
Example	-> :TRIGger:EVent1:CONDition1:GET? <- NONE -> :TRIGger:EVent1:ADDCondition -> :TRIGger:EVent1:CONDition1? // ↪use as default query <- (CONDITION,HIGHLEVEL,0.0,OFF) -> :TRIG:EV:COND? // ↪short form <- (CONDITION,HIGHLEVEL,0.0,OFF)

20.12 :TRIGger:EVent<event-number>:CONDition<condition-number>:VALId

Syntax	:TRIGger:EVent<event-number>:CONDition<condition-number>:VALId?
Description	Queries validation state of the condition
Parameter	None
Explanation	Returns TRUE if the condition has a valid configuration. FALSE is returned if the condition does not exist, or one or more parameters are missing or mismatched (e.g. no channel assigned).
Return Format	TRUE | FALSE
Example	-> :TRIGger:EVent1:CONDition1:VALId? <- TRUE

20.13 :TRIGger:EVent<event-number>::CONDition<condition-number>:DELe

Syntax	:TRIGger:EVent<event-number>::CONDition<condition-number>:DELete
Description	Deletes the trigger condition with corresponding condition number
Parameter	None
Explanation	Deletes the trigger condition with the corresponding condition number from the corresponding event. Note: All subsequent conditions are reordered with a new condition number. If no condition number is specified, condition 1 is deleted.
Example	-> :TRIGger:EVent1? <- ON,"Event 1",(CONDITION,HIGHLEVEL,0.0,OFF), ↪(CONDITION,KEYBOARD,SINGLE,"Shift+C") -> :TRIGger:EVent1:CONDition1:DELete -> :TRIGger:EVent1? <- ON,"Event 1",(CONDITION,KEYBOARD,SINGLE,"Shift+C")

20.14 :TRIGger:EVent<event-number>:CONDition<condition-number>:HIGHl <nrf>,<literal>|<nrf>,<String>[,<String>[,...]]

Syntax	**:TRIGger:EVent<event-number>:CONDition<condition-number>:HIGHlevel:SETup <nrf>,<literal>|<nrf>,<String>[,<String>[,...]]**
Description	Converts corresponding condition to a high level condition
Parameter	

Name	Type	Range	Default
Threshold	<nrf>	channel range min max	
Rearm level	<Boolean> | <nrf>	ON | OFF | range min - threshold value	
Channel ID List	<String>[,<String>[,...]]	<String>[,<String>[,...]]	

Explanation	Converts the corresponding condition to a high level condition and sets its parameter. Note: the rearm level param should be ON or OFF or a numric value (see channel range). If this parameter is a numeric value the rearm level is enabled automatically.
Example	

```
-> :TRIGger:EVent1:CONDition1:GET?
<- NONE
-> :TRIGger:EVent1:ADDCondition
-> :TRIGger:EVent1:CONDition1?                    //
 ↪same GET as default query
<- (CONDITION,HIGHLEVEL,0.0,OFF)
-> :TRIGger:EVent1:CONDition1:HIGHlevel:SETup 1,0.5,
 ↪"17342517731483713537","17342517731483713538"
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,1.0,5.0E-1,
 ↪"17342517731483713537","17342517731483713538")
-> :TRIG:EV1:COND1:HIGH 1,0.5,"17342517731483713537",
 ↪"17342517731483713538" //short command
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,1.0,5.0E-1,
 ↪"17342517731483713537","17342517731483713538")
```

20.15 :TRIGger:EVent<event-number>:CONDition<condition-number>:LOWle <nrf>,<literal>|<nrf>,<String>[,<String>[,...]]

Syntax	**:TRIGger:EVent<event-number>:CONDition<condition-number>:LOWlevel:SETup <nrf>,<literal>|<nrf>,<String>[,<String>[,...]]**
Description	Converts corresponding condition to a low level condition
Parameter	

Name	Type	Range	Default
Threshold	<nrf>	channel range min max	
Rearm level	<Boolean> | <nrf>	ON | OFF | threshold value - range max	
Channel ID List	<String>[,<String>[,...]]	<String>[,<String>[,...]]	

Explanation	Converts the corresponding condition to a low level condition and sets its parameter. Note: the rearm level param should be ON or OFF or a numric value (see channel range). If this parameter is a numeric value the rearm level is enabled automatically.
Example	

```
-> :TRIGger:EVent1:CONDition1:GET?
<- NONE
-> :TRIGger:EVent1:ADDCondition
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,0.0,OFF)
-> :TRIGger:EVent1:CONDition1:LOWlevel:SETup 0.5,1.0,
 ↪"17342517731483713537","17342517731483713538"
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,LOWLEVEL,5.0E-1,1.0,
 ↪"17342517731483713537","17342517731483713538")
-> :TRIG:EV1:COND1:LOW 0.5,1.0,"17342517731483713537",
 ↪"17342517731483713538" //short command
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,LOWLEVEL,5.0E-1,1.0,
 ↪"17342517731483713537","17342517731483713538")
```

20.16 :TRIGger:EVent<event-number>:CONDition<condition-number>:INwin<nrf>,<nrf>,<String>[,<String>[,...]]

Syntax	**:TRIGger:EVent<event-number>:CONDition<condition-number>:INwindow:SETup <nrf>,<nrf>,<String>[,<String>[,...]]**
Description	Converts corresponding condition to a in window condition
Parameter	

Name	Type	Range	Default
Lower level	<nrf>	range min - upper level	
Upper level	<nrf>	lower level - range max	
Channel ID List	<String>[,<String>[,<String>[,...]]]		

Explanation	Converts the corresponding condition to a in window condition and sets its parameters lower level, upper level and channel ids. For lower and upper level see the channel range min-max for relevant values
Example	

```
-> :TRIGger:EVent1:CONDition1:GET?
<- NONE
-> :TRIGger:EVent1:ADDCondition
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,0.0,OFF)
-> :TRIGger:EVent1:CONDition1:INwindow:SETup -0.5,0.
 ↪5,"17342517731483713537","17342517731483713538"
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,INWINDOW,-5.0E-1,5.0E-1,
 ↪"17342517731483713537","17342517731483713538")
-> :TRIG:EV1:COND1:IN -0.5,0.5,"17342517731483713537",
 ↪"17342517731483713538" // short command
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,INWINDOW,-5.0E-1,5.0E-1,
 ↪"17342517731483713537","17342517731483713538")
```

20.17 :TRIGger:EVent<event-number>:CONDition<condition-number>:OUTw <nrf>,<nrf>,<String>[,<String>[,...]]

Syntax	**:TRIGger:EVent<event-number>:CONDition<condition-number>:OUTwindow:SETu <nrf>,<nrf>,<String>[,<String>[,...]]**
Description	Converts corresponding condition to a out window condition
Parameter	

Name	Type	Range	Default
Lower level	<nrf>	range min - upper level	
Upper level	<nrf>	lower level - range max	
Channel ID List	<String>[,<String>[,...]]	<String>[,<String>[,...]]	

Explanation	Converts the corresponding condition to a out window condition and sets its parameters lower level, upper level and channel ids. For lower and upper level see the channel range min-max for relevant values
Example	

```
-> :TRIGger:EVent1:CONDition1:GET?
<- NONE
-> :TRIGger:EVent1:ADDCondition
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,0.0,OFF)
-> :TRIGger:EVent1:CONDition1:OUTwindow:SETup -0.3,0.
↪3,"17342517731483713537"
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,OUTWINDOW,-3.0E-1,3.0E-1,
↪"17342517731483713537")
-> :TRIG:EV1:COND1:OUT -0.3,0.3,
↪"17342517731483713537" // short command
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,OUTWINDOW,-3.0E-1,3.0E-1,
↪"17342517731483713537")
```

20.18 :TRIGger:EVent<event-number>:CONDition<condition-number>:KEYB

<table>
<tr><td>Syntax</td><td>:TRIGger:EVent<event-number>:CONDition<condition-number>:KEYBoard:SETup <literal>,<String></td></tr>
<tr><td>Description</td><td>Converts corresponding condition to a keyboard condition</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>Pressed behaviour</td><td><literal></td><td>SINGLE | TOGGLE</td><td></td></tr>
<tr><td>Keyboard Key</td><td><String></td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Converts the corresponding condition to a keyboard condition and sets its parameters pressed behaviour and keyboard key. Note: the keyboard key should be a valid key string with “Shift”, “Ctrl”, “Alt” and a valid key char (e.g. “Ctrl+Alt+K”)</td></tr>
<tr><td>Example</td><td>

```
-> :TRIGger:EVent1:CONDition1:GET?
<- NONE
-> :TRIGger:EVent1:ADDCondition
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,0.0,OFF)
-> :TRIGger:EVent1:CONDition1:KEYBoard:SETup TOGGLE,
 ↪"Ctrl+C"
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,KEYBOARD,TOGGLE,"Ctrl+C")
-> :TRIG:EV1:COND1:KEYB TOGGLE,"Ctrl+C" // short␣
 ↪command
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,KEYBOARD,TOGGLE,"Ctrl+C")
```

</td></tr>
</table>

20.19 :TRIGger:EVent<event-number>:CONDition<condition-number>:TIME:S

<table>
<tr><td>Syntax</td><td>:TRIGger:EVent<event-number>:CONDition<condition-number>:TIME:SETup <literal>|<String>,<literal>|<nrf>,<literal>|<nrf></td></tr>
<tr><td>Description</td><td>Converts corresponding condition to a time condition</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>First trigger time</td><td><Boolean> | <String></td><td>ON | OFF | “yyyy-MM-dd’T’HH:mm:ss”</td><td></td></tr>
<tr><td>Trigger interval time</td><td><Boolean> | <nrf></td><td>ON | OFF | seconds</td><td></td></tr>
<tr><td>Active for time</td><td><Boolean> | <nrf></td><td>ON | OFF | seconds</td><td></td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Converts the corresponding condition to a time condition and sets its parameters first trigger time, trigger interval and active for time. Note: the first trigger time should be ON or OFF or a valid ISO 8601 time string format. if a valid Iso string is given ON is assumed automatically.</td></tr>
<tr><td>Example</td><td>

```
-> :TRIGger:EVent1:CONDition1:GET?
<- NONE
-> :TRIGger:EVent1:ADDCondition
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,HIGHLEVEL,0.0,OFF)
-> :TRIGger:EVent1:CONDition1:TIME:SETup␣
 ↪"2023-11-14T12:59:00",60,20
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,TIME,"2023-11-14T12:59:00",60.0,20.0)
-> :TRIG:EV1:COND1:TIME "2023-11-14T12:59:00",60,20 /
 ↪/ short command
-> :TRIGger:EVent1:CONDition1?
<- (CONDITION,TIME,"2023-11-14T12:59:00",60.0,20.0)
```

</td></tr>
</table>

20.20 :TRIGger:EVent<event-number>:ACTion<action-number>:GET?

Syntax	:TRIGger:EVent<event-number>:ACTion<action-number>:GET?
Description	Queries the action type and all parameters of the corresponding trigger event action
Parameter	None
Explanation	Returns the action type and all of the specific parameters for this type. Note: this is the default query in the trigger event action subsystem (see sample example).
Return Format	([<action-type>[,<parameter 1>[,…]]]) | NONE • Recording action: (ACTION,RECORDING,START|EVENT|STOP|PAUSE|TOGGLE) • Digital out action: (ACTION,DIGITALOUT,<literal>|<nrf>,<literal>|<nrf>,<literal>),<String>,<String>, …) • Alarm action: (ACTION,ALARM,<literal>,<literal>|<nrf>,<literal>|<nrf>,<literal>),<String>,<String>, …) • Marker action: (ACTION,MARKER,”Event 1”,<literal>) • Snapshot action:(ACTION,SNAPSHOT,<literal>,<nrf>,<String>,<String>, …) • Arm action: (ACTION,ARM,<literal>) For a detailed parameter name and description of all trigger event actions see the action setup commands below.

Example	`-> :TRIGger:EVent1:ACTION1:GET?`
	`<- NONE`
	`-> :TRIGger:EVent1:ADDAction`
	`-> :TRIGger:EVent1:ACTion?`
	`<- (ACTION,RECORDING,START)`

20.21 :TRIGger:EVent<event-number>:ACTion<action-number>:VALId?

Syntax	:TRIGger:EVent<event-number>:ACTion<action-number>:VALId?
Description	Queries validation state of the corresponding action
Parameter	None
Explanation	Returns TRUE if the action has a valid configuration. FALSE is returned if the action does not exist, or one or more parameters are missing or mismatched (e.g. no channel assigned).
Return Format	TRUE | FALSE
Example	`-> :TRIGger:EVent1:ACTion1:VALId?` `<- TRUE`

20.22 :TRIGger:EVent<event-number>::ACTion<action-number>:DELete

Syntax	:TRIGger:EVent<event-number>::ACTion<action-number>:DELete
Description	Deletes the trigger action with corresponding action number
Parameter	None
Explanation	Deletes the trigger action with the corresponding action number from the corresponding event. Note: All subsequent actions are reordered with a new action number. If no action number is specified, action 1 is deleted.
Example	-> :TRIGger:EVent2? <- ON,"Event 2",(ACTION,SNAPSHOT,AVG,1.0),(ACTION, ↪ARM,OFF) -> :TRIGger:EVent2:ACTion2:DELete -> :TRIGger:EVent2? <- ON,"Event 2",(ACTION,SNAPSHOT,AVG,1.0)

20.23 :TRIGger:EVent<event-number>:ACTion<action-number>:RECording:S
<literal>

Syntax	:TRIGger:EVent<event-number>:ACTion<action-number>:RECording:SETup <literal>
Description	Converts corresponding action to a recording action
Parameter	Name: Recording action; Type: <literal>; Range: START | EVENT | STOP | PAUSE | TOGGLE; Default:
Explanation	Converts the corresponding action to a recording action and sets its action parameter. Note: this is the default command in the trigger event action recording subsystem (see sample example).
Example	-> :TRIGger:EVent1:ACTion1:GET? <- NONE -> :TRIGger:EVent1:ADDAction -> :TRIGger:EVent1:ACTion1? <- (ACTION,RECORDING,START) -> :TRIGger:EVent1:ACTion1:RECording:SETup EVENT -> :TRIGger:EVent1:ACTion1? <- (ACTION,RECORDING,EVENT) -> :TRIG:EV1:ACT1:REC EVENT // short command -> :TRIGger:EVent1:ACTion1? <- (ACTION,RECORDING,EVENT)

20.24 :TRIGger:EVent<event-number>:ACTion<action-number>:DIGOut:SETup <literal>|<nrf>,<literal|<nrf>,<literal>,<String>[,<String>[,...]]

<table>
<tr><td>Syntax</td><td>TRIGger:EVent<event-number>:ACTion<action-number>:DIGOut:SETup <literal>|<nrf>,<literal|<nrf>,<literal>,<String>[,<String>[,...]]</td></tr>
<tr><td>Description</td><td>Converts corresponding action to a digital out action</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>Delay do channels</td><td><Boolean> | <nrf></td><td>ON | OFF | 0-3600s</td><td></td></tr>
<tr><td>Auto reset do channels</td><td><Boolean> | <nrf></td><td>ON | OFF | 0-3600s</td><td></td></tr>
<tr><td>Digital out level</td><td><literal></td><td>HIGH | LOW</td><td></td></tr>
<tr><td>Channel ID List</td><td><String>[,<String>[,<String>[,...]]]</td><td><String>[,<String>[,...]]</td><td></td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Converts the corresponding action to a digital out action and sets its action parameters. Note: this is the default command in the trigger event action digital out subsystem (see sample example).</td></tr>
<tr><td>Example</td><td>

```
-> :TRIGger:EVent1:ACTion1:GET?
<- NONE
-> :TRIGger:EVent1:ADDAction
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,RECORDING,START)
-> :TRIGger:EVent1:ACTion1:DIGout:SEtup 1,60,LOW,
 ↪"17342517731483713537"
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,DIGITALOUT,1.0,60.0,LOW,
 ↪"17342517731483713537")
-> :TRIG:EV1:ACT1:DIGO 1,60,LOW,
 ↪"17342517731483713537”  // short command
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,DIGITALOUT,1.0,60.0,LOW,
 ↪"17342517731483713537")
```

</td></tr>
</table>

20.25 :TRIGger:EVent<event-number>:ACTion<action-number>:ALARm:SETup <literal>,<literal>|<nrf>,<literal|<nrf>,<literal>,<String>[,<String>[,...]]

<table>
<tr><td>Syntax</td><td>TRIGger:EVent<event-number>:ACTion<action-number>:ALARm:SETup <literal>,<literal>|<nrf>,<literal|<nrf>,<literal>,<String>[,<String>[,...]]</td></tr>
<tr><td>Description</td><td>Converts corresponding action to a alarm action</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>Add marker on Alarm</td><td><Boolean></td><td>ON | OFF</td><td></td></tr>
<tr><td>Auto reset do channels</td><td><Boolean> | <nrf></td><td>ON | OFF | 0-3600s</td><td></td></tr>
<tr><td>Auto reset do channels</td><td><Boolean> | <nrf></td><td>ON | OFF | 0-3600s</td><td></td></tr>
<tr><td>Digital out level</td><td><literal></td><td>HIGH | LOW</td><td></td></tr>
<tr><td>Channel ID List</td><td><String>[,<String>[,<String>[,...]]]</td><td><String>[,<String>[,...]]</td><td></td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Converts the corresponding action to a alarm action and sets its action parameters. Note: this is the default command in the trigger event action alarm subsystem (see sample example).</td></tr>
<tr><td>Example</td><td>

```
-> :TRIGger:EVent1:ACTion1:GET?
<- NONE
-> :TRIGger:EVent1:ADDAction
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,RECORDING,START)
-> :TRIGger:EVent1:ACTion1:ALARm:SETup ON,1,60,LOW,
 ↪"17342517731483713537"
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,ALARM,ON,1.0,60.0,LOW,
 ↪"17342517731483713537")
-> :TRIG:EV1:ACT1:ALAR ON,1,60,LOW,
 ↪"17342517731483713537"  // short command
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,ALARM,ON,1.0,60.0,LOW,
 ↪"17342517731483713537")
```

</td></tr>
</table>

20.26 :TRIGger:EVent<event-number>:ACTion<action-number>:MARKer:SETup <String>,<literal>

<table>
<tr><td>Syntax</td><td>:TRIGger:EVent<event-number>:ACTion<action-number>:MARKer:SETup <String>,<literal></td></tr>
<tr><td>Description</td><td>Converts corresponding action to a marker action</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td>Marker Text</td><td><String></td><td></td><td></td></tr>
<tr><td>Append State</td><td><literal></td><td>ONACTIVE | ONINCATIVE | ONBOTH</td><td></td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Converts the corresponding action to a marker action and sets its action parameters. Note: this is the default command in the trigger event marker subsystem (see sample example).</td></tr>
<tr><td>Example</td><td>

```
-> :TRIGger:EVent1:ACTion1:GET?
<- NONE
-> :TRIGger:EVent1:ADDAction
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,RECORDING,START)
-> :TRIGger:EVent1:ACTion1:MARKer:SETup "My Marker",
 ↪ONBOTH
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,MARKER,"My Marker",ONBOTH)
-> :TRIG:EV1:ACT1:MARK "My Marker",ONBOTH  // short␣
 ↪command
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,MARKER,"My Marker",ONBOTH)
```

</td></tr>
</table>

20.27 :TRIGger:EVent<event-number>:ACTion<action-number>:SNAPshot:SETup <literal>,<nrf>,<String>[,<String>[,...]]

Syntax	**:TRIGger:EVent<event-number>:ACTion<action-number>:SNAPshot:SETup <literal>,<nrf>,<String>[,<String>[,...]]**
Description	Converts corresponding action to a snapshot action
Parameter	

Name	Type	Range	Default
Calculation Mode	<literal>	ACTUAL | MIN | RMS | PEAK | AVG | MAX | ACRMS	
Snapshot Window	<nrf>	0.001 - 10s	
Channel ID List	<String>[,<String>[,<String>[,...]]]	<String>[,<String>[,<String>[,...]]]	

Explanation	Converts the corresponding action to a snapshot and sets its action parameters. Note: this is the default command in the trigger event action snapshot subsystem (see sample example).
Example	

```
-> :TRIGger:EVent1:ACTion1:GET?
<- NONE
-> :TRIGger:EVent1:ADDAction
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,RECORDING,START)
-> :TRIGger:EVent1:ACTion1:SNAPshot:SETup AVG,1.0,
 ↪"73031551831632773l2"
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,SNAPSHOT,AVG,1.0,"73031551831632773l2")
-> :TRIG:EV1:ACT1:SNAP AVG,1.0,"73031551831632773l2"␣
 ↪ // short command
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,SNAPSHOT,AVG,1.0,"73031551831632773l2")
```

20.28 :TRIGger:EVent<event-number>:ACTion<action-number>:ARM:SETup <literal>

<table>
<tr><td>Syntax</td><td>:TRIGger:EVent<event-number>:ACTion<action-number>:ARM:SETup <literal></td></tr>
<tr><td>Description</td><td>Converts corresponding action to an arm action</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Range</td><td>Default</td></tr>
<tr><td>Arm State</td><td><literal></td><td>ON | OFF</td><td></td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>Converts the corresponding action to an arm and sets its action parameter. Note: this is the default command in the trigger event action arm subsystem (see sample example).</td></tr>
<tr><td>Example</td><td>

```
-> :TRIGger:EVent1:ACTion1:GET?
<- NONE
-> :TRIGger:EVent1:ADDAction
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,RECORDING,START)
-> :TRIGger:EVent1:ACTion1:ARM:SETup ON
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,ARM,ON)
-> :TRIG:EV1:ACT1:ARM ON  // short command
-> :TRIGger:EVent1:ACTion1?
<- (ACTION,ARM,ON)
```

</td></tr>
</table>

CHAPTER

TWENTYONE

ANALYSIS CONTROL

With the following commands and queries you can control the oxygen analysis mode.

Please note, when the oxygen analysis mode is active, several commands of other subsystems do not make sense and would return an execution error. The start/restart of oxygens acquisition is not possible in analysis mode. Several commands from the recording control would also return execution errors. The loading of setup files is not possible, but saving the setup of the current loaded dmd files is possible.

21.1 :ANAlysis:ACTIve?

Syntax	**:ANAlysis:ACTIve**
Description	Queries the oxygen analysis state
Parameter	None
Explanation	Return true if the oxygen is in the file analysis state.
Return Format	<Boolean>
Example	`-> :ANAlysis:Active?` `<- 0`

21.2 :ANAlysis:OPEN <file_name>|<path>[,<file_name>|<path>[,...]]

<table>
<tr><td>Syntax</td><td>:ANAlysis:OPEN <file_name>|<path>[,<file_name>|<path>[,...]]</td></tr>
<tr><td>Description</td><td>Load specified data files.</td></tr>
<tr><td>Parameter</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr>
<tr><td><file_name></td><td>String</td><td>The filename of the data file in the default data directory.</td><td>None</td></tr>
<tr><td><path></td><td>String</td><td>The absolute path of the setup file to load</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>Explanation</td><td>This command loads one or more measurement data files. If the specified file path is relative, the file is loaded from the current recording directory. The <file_name> does not need to include the file extension “.dmd”. If one file cannot be loaded, Oxygen switches back to live mode and returns one of the following error codes in the error queue.
Data corrupt or stale (-230)
Invalid format (-232)
Invalid version (-233)
File name not found (-256)
Settings conflict (-221)
Media protected (-258)
In all other error situations Execution error (-200) is returned</td></tr>
<tr><td>Example</td><td>

```
-> :ANAlysis:OPEN "file1.dmd"
-> :ANAlysis:Active?
<- 1
...
-> :ANAlysis:OPEN "file1.dmd","file2.dmd"
...
-> :ANAlysis:OPEN "d:/DATA/file1.dmd","d:/DATA/file2.
↪dmd"
```

</td></tr>
</table>

21.3 :ANAlysis:CLOSe

Syntax	:ANAlysis:CLOSe
Parameter	None
Description	Closes all opened data files.
Explanation	This command closes all opened measurement data files. Oxygen returns to live mode.
Example	`-> :ANAlysis:CLOSE`

21.4 :ANAlysis:FILES?

Syntax	:ANAlysis:FILES?
Description	Queries all opened data FILES.
Parameter	None
Explanation	Returns a list of all opened FILES. The returned filename is an absolute path.
Return Format	(<String>[,<String >, ...]) | NONE
Example	`-> :ANAlysis:FILES?` `<- "d:/DATA/file1.dmd","d:/DATA/file2.dmd"`

CHAPTER

TWENTYTWO

ERROR HANDLING

22.1 :SYSTem:ERRor[:NEXT]?

Syntax	:SYSTem:ERRor[:NEXT]?
Description	Queries the next element in the Error Queue
Parameter	None
Explanation	Queries the next element in the Error Queue
Return Format	<NR1>, String
Example	-> :SYSTem:ERRor? <- -102, "Syntax error" -> :SYSTem:ERRor? <- -108, "Parameter not allowed" -> :SYSTem:ERRor? <- 0, "No error"

22.2 :SYSTem:ERRor:ALL?

Syntax	:SYSTem:ERRor:ALL?
Description	Queries all entries the Error Queue
Parameter	None
Explanation	Queries all entries the Error Queue
Return Format	<NR1>, String, <NR1>, String, ...
Example	-> :SYSTem:ERRor:ALL? <- -102, "Syntax error", -108, "Parameter not␣ ↪allowed"

22.3 :SYSTem:ERRor:CODE[:NEXT]?

Syntax	:SYSTem:ERRor:CODE:NEXT?
Description	Queries next error code in the Error Queue
Parameter	None
Explanation	Queries next error code in the Error Queue
Return Format	<NR1>
Example	-> :SYSTem:ERRor:CODE? <- -102 -> :SYSTem:ERRor:CODE? <- -108

22.4 :SYSTem:ERRor:CODE:ALL?

Syntax	:SYSTem:ERRor:CODE:ALL?
Description	Queries all error codes in the Error Queue
Parameter	None
Explanation	Queries all error codes in the Error Queue
Return Format	<NR1>,<NR1>,...
Example	-> :SYSTem:ERRor:CODE:ALL? <- -102,-108

22.5 :SYSTem:ERRor:COUNt?

Syntax	:SYSTem:ERRor:COUNt?
Description	Queries the Error Queue number of elements
Parameter	None
Explanation	Queries the Error Queue number of elements
Return Format	<NR1>
Example	-> :SYSTem:ERRor:COUNt? <- 2

22.6 :SYSTem:ERRor:ENABle:ADD (<num>:<num>)

Syntax	:SYSTem:ERRor:ENABle:ADD (<num>:<num>)
Description	Add a range of error codes to be queued in the Error Queue
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><num></td><td>Integer</td><td>-32768 to 32768</td><td>None</td></tr></table>
Explanation	Add a range of error codes to be queued in the Error Queue.
Example	-> :SYSTem:ERRor:ENABle:LIST? <- (-499:-100,1:32767) -> :SYSTem:ERRor:ENABle:ADD (-1000:-900) -> :SYSTem:ERRor:ENABle:LIST? <- (-1000:-900,-499:-100,1:32767)

22.7 :SYSTem:ERRor:ENABle:DELete (<num>:<num>)

Syntax	:SYSTem:ERRor:ENABle:DELete (<num>:<num>)
Description	Delete a range of error codes to be queued in the Error Queue
Parameter	<table><tr><th>Name</th><th>Type</th><th>Range</th><th>Default</th></tr><tr><td><num></td><td>Integer</td><td>-32768 to 32768</td><td>None</td></tr></table>
Explanation	Delete a range of error codes to be queued in the Error Queue
Example	-> :SYSTem:ERRor:ENABle:LIST? <- (-499:-100,1:32767) -> :SYSTem:ERRor:ENABle:DELete (-199:-100) -> :SYSTem:ERRor:ENABle:LIST? <- (-499:-200,1:32767)

22.8 :SYSTem:ERRor:ENABle[:LIST]?

Syntax	:SYSTem:ERRor:ENABle:LIST?
Description	Queries the range of error codes to be queued in the Error Queue
Parameter	None
Explanation	Queries the range of error codes to be queued in the Error Queue
Return Format	(<NR1>:<NR1>,<NR1>:<NR1>)
Example	-> :SYSTem:ERRor:ENABle:LIST <- (-499:-100,1:32767)

CHAPTER

TWENTYTHREE

ERROR CODES

Use the :SYSTem:ERRor:ALL? command to return all errors that have occurred. The errors returned have to following syntax <Error Code>,<Error Message>. The following table shows the error codes possible when interacting with OXYGEN software. SCPI defines the negative error codes, while the vendor specific error codes defined by DEWETRON GmbH are positive numbers (currently unused).

Error Code	Error Message	Description
0	No error	
100	Command error	
-102	Syntax error	Indicates that an unrecognized command or data type was encountered. For example, a string was received when the device does not accept strings.
-104	Data type error	The parser recognized a data element different than one allowed. For example, numeric or string data was expected but block data was encountered.
-108	Parameter not allowed	Indicates that less parameters were received than required for the header.
-109	Missing parameter	Indicates that more parameters were received than expected for the header
-113	Undefined header	Indicates the header is syntactically correct, but it is undefined for this specific device.
-114	Header suffix out of range	Indicates the value of a header suffix attached to a program mnemonic makes the header invalid.
-138	Suffix not allowed	Indicates that a suffix was encountered after a numeric element that does not allow suffixes.
-200	Execution error	General execution error.
-220	Parameter error	Indicates that a program data element related error occurred.
-221	Settings conflict	Indicates that a legal program data element was parsed but could not be executed due to the current device state.
-222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value was outside the legal range defined by the devices.
-224	Illegal parameter value	Indicates that a program data element is ill-formed
-250	Mass storage error	Indicates that a mass storage error occurred.
-256	File name not found	Indicates that a legal program command or query could not be executed because the file name was not found on the media.
-294	Incompatible type	
-300	Device-specific error	
-350	Queue overflow	
-400	Query error	

CHAPTER

TWENTYFOUR

EXAMPLES

24.1 Fetch Online Measurement Data

```
-> *RST                                  // Reset Device
-> :COMMunicate:HEADer 0                 // Switch Off Header response
-> *IDN?                                 // Query Identification
<- "DEWETRON,OXYGEN,0,2.5 RC1"
-> *VER?                                 // Query Version Information
<- "SCPI,"1999.0",RC_SCPI,"1.5",OXYGEN,"2.5 RC1""
-> :SETUP:LOAD "scpi_test_setup.dms"     // Load Measurement setup
-> :ACQU:STAT?                           // Query Acquisition state
<- Waiting_for_sync
-> :ACQU:STAT?                           // Query Acquisition state
<- Started
-> :RATE 500ms                           // Set Aggregation Rate to␣
↪500ms
-> :NUM:NORMal:ITEMs "ABS-TIME","U1_tRMS@PG1","I1_tRMS@PG1","P1_
↪t@PG1"
-> :NUM:NORMal:ITEMs?                    // Query Output Channels
<- "ABS-TIME","U1_tRMS@PG1","I1_tRMS@PG1","P1_t@PG1"
-> :NUM:NORMal:VAL?
<- "2017-08-28T13:17:26.9715+00:00",5.6568531E+1,5.6568531E+1,3.
↪1999988E+3
```

24.2 Store Measurement Data on Device

```
-> *RST                                  // Reset Device
-> :COMMunicate:HEADer 0                 // Switch Off Header response
-> *IDN?                                 // Query Identification
<- "DEWETRON,OXYGEN,0,5.1.1"
-> *VER?                                 // Query Version Information
<- "SCPI,"1999.0",RC_SCPI,"1.10",OXYGEN,"5.1.1""
-> :SETUP:LOAD "scpi_test_setup.dms"     // Load Measurement setup
-> :ACQU:STAT?                           // Query Acquisition state
<- Waiting_for_sync
-> :ACQU:STAT?                           // Query Acquisition state
<- Started
```

```
-> :STORe:FILE:NAME "TEST_1"          // Set File Name for storing␣
 ↪operation
-> :STORe:STARt                       // Start storing operation
-> :STORe:PAUSe                       // Pause storing operation
-> :STORe:STARt                       // Resume storing operation
<- :STORe:STOP                        // Stop storing operation
```

24.3 Set Channel Properties

24.3.1 Set a bool Item example

```
-> :CHANNEL:ID? "Sync Sim 0"          //get channel id
<- :CHANNEL:ID "10271021882991968256"
-> :CHANNEL:CONSTR? "10271021882991968256","Used"  //check␣
 ↪constraints
<- :CHANNEL:CONSTR (BOOL,OFF),(BOOL,ON)
-> CHANNEL:PROP? "10271021882991968256","Used"     //get value
<- :CHANNEL:PROP (BOOL,OFF)
-> :CHANNEL:PROP "10271021882991968256","Used",ON  //set value
-> :CHANNEL:PROP? "10271021882991968256","Used"
<- :CHANNEL:PROP (BOOL,ON)
-> :CHANNEL:PROP "10271021882991968256","Used",OFF
-> :CHANNEL:PROP? "10271021882991968256","Used"
<- :CHANNEL:PROP OFF
```

24.3.2 Set a string Item example

```
-> :CHANNEL:ID? "AI 2/1 Sim"  //get channel id
<- :CHANNEL:ID "8785115489526349845"
-> :CHANNEL:PROP? "8785115489526349845","Mode"  //get current value
<- :CHANNEL:PROP (STRING,"Voltage")
-> :CHANNEL:CONSTR? "8785115489526349845","Mode"  //get item␣
 ↪constraints
<- :CHANNEL:CONSTR (STRING,"Calibration"),(STRING,"Voltage"),
   (STRING,"Resistance"),(STRING,"IEPE"),(STRING,"Bridge"),
   (STRING,"ExcCurrentMonitor"),(STRING,"ExcVoltMonitor"),
   (STRING,"Current")
-> :CHANNEL:PROP "8785115489526349845","Mode","Resistance"  //set␣
 ↪new value
-> :CHANNEL:PROP? "8785115489526349845","Mode"  //get current value
<- :CHANNEL:PROP (STRING,"Resistance")
```

24.3.3 Set a floating point Item example

```
-> :CHANNEL:PROP? "3789779077842337813","Neon/PhysicalScaleFactor"
<- :CHANNEL:PROP (FLOAT,1.2)
-> :CHANNEL:PROP? "3789779077842337813","Neon/PhysicalScaleOffset"
<- :CHANNEL:PROP (FLOAT,0.0)
-> :CHANNEL:PROP "3789779077842337813","Neon/PhysicalScaleFactor",1.
↪1
-> :CHANNEL:PROP "3789779077842337813","Neon/PhysicalScaleOffset",0.
↪1
-> :CHANNEL:PROP? "3789779077842337813","Neon/PhysicalScaleFactor"
<- :CHANNEL:PROP (FLOAT,1.1)
-> :CHANNEL:PROP? "3789779077842337813","Neon/PhysicalScaleOffset"
<- :CHANNEL:PROP (FLOAT,1.0E-1)
```

24.3.4 Set an enum item example

```
-> :CHANNEL:ID? "Sync Sim 0" //get channel id
<- :CHANNEL:ID "10439625394041651200"
-> :CHANNEL:PROP? "10439625394041651200","Neon/Stored" //get␣
↪current value
<- :CHANNEL:PROP (ENUM,"ChannelStored","Auto")
-> :CHANNEL:CONSTR? "10439625394041651200","Neon/Stored" //get␣
↪constraints
<- :CHANNEL:CONSTR (ENUM,"ChannelStored","Auto"),(ENUM,
↪"ChannelStored","No")
-> :CHANNEL:PROP "10439625394041651200","Neon/Stored","No" //set␣
↪item
-> :CHANNEL:PROP? "10439625394041651200","Neon/Stored" //get␣
↪current value
-> :CHANNEL:PROP (ENUM,"ChannelStored","No")
-> :CHANNEL:PROP "10439625394041651200","Neon/Stored",
↪"ChannelStored","Auto"
-> :CHANNEL:PROP? "10439625394041651200","Neon/Stored"
-> :CHANNEL:PROP (ENUM,"ChannelStored","Auto")
```

24.3.5 Set scalar item example

```
-> :CHANNEL:ID? "AI 2/1 Sim"
<- :CHANNEL:ID "14649928104269578261"
-> :CHANNEL:PROP? "14649928104269578261","SensorDelay"   //get␣
↪current value
<- :CHANNEL:PROP (SCALAR,0.0,"ms")
-> :CHANNEL:CONSTR? "14649928104269578261","SensorDelay" //get item␣
↪constraints
-> :CHANNEL:CONSTR (FLOAT,0.0),(FLOAT,500.0)
-> :CHANNEL:PROP "14649928104269578261","SensorDelay",SCALAR,100,
↪"ms"  //set item
-> :CHANNEL:PROP? "14649928104269578261","SensorDelay"   //get␣
↪current value
```

```
<- :CHANNEL:PROP (SCALAR,100.0,"ms")
-> :CHANNEL:PROP "14649928104269578261","SensorDelay",500  //set␣
↪item
-> :CHANNEL:PROP? "14649928104269578261","SensorDelay"   //get␣
↪current value
-> :CHANNEL:PROP (SCALAR,500.0,"ms")
-> :CHANNEL:PROP "14649928104269578261","SensorDelay",0.4s   //set␣
↪item
-> :CHANNEL:PROP? "14649928104269578261","SensorDelay"   //get␣
↪current value
<- :CHANNEL:PROP (SCALAR,4.0E-1,"s")
-> :CHANNEL:PROP "14649928104269578261","SensorDelay",300,"ms"
-> :CHANNEL:PROP? "14649928104269578261","SensorDelay"
<- :CHANNEL:PROP (SCALAR,300.0,"ms")
```

24.3.6 Set range item example

```
-> :CHANNEL:PROP? "8785115489526349845","Range"
<- :CHANNEL:PROP (RANGE,-10.0,"V",10.0,"V")
-> :CHANNEL:CONSTR? "8785115489526349845","Range"
<- :CHANNEL:CONSTR (FLOAT,2.0E-4),(FLOAT,10.0),(RANGE,-10.0,"V",10.
↪0,"V"),
    (RANGE,-3.0,"V",3.0,"V"),(RANGE,-1.0,"V",1.0,"V"),
    (RANGE,-3.0E-1,"V",3.0E-1,"V"),(RANGE,-1.0E-1,"V",1.0E-1,"V"),
    (RANGE,-3.0E-2,"V",3.0E-2,"V"),(RANGE,-1.0E-2,"V",1.0E-2,"V")
-> :CHANNEL:PROP "8785115489526349845","Range",RANGE,-1.0E-2,"V",1.
↪0E-2,"V"
-> :CHANNEL:PROP? "8785115489526349845","Range"    //get current␣
↪value
<- :CHANNEL:PROP (RANGE,-1.0E-2,"V",1.0E-2,"V")
-> :CHANNEL:PROP "8785115489526349845","Range",-3.0V,3.0V   //set␣
↪value
-> :CHANNEL:PROP? "8785115489526349845","Range"    //get current␣
↪value
<- :CHANNEL:PROP (RANGE,-3.0,"V",3.0,"V")
```