TLS120Xe Communications Manual

Version 1.7.0

The TLS120Xe is a high power tuneable light source that delivers superlative stability, high wavelength agility and the power of continuous tuning.

Bringing together high optical performance and the convenience of a single, plug-and-play unit, the TLS120Xe gives you the power of monochromatic light at your fingertips.

Key features:

- Ultra-quiet 75W short-arc xenon light source with starter and internal constant current PSU_610 power supply
- Fixed slit high throughput fast scanning monochromator using a single concave holographic reflectance grating with an effective focal length of 119mm
- Manual wavelength control through front panel (1nm resolution)
- Fully programmable via USB using intuitive SCPI protocol (wavelength resolution 0.1nm)

Quick Start

To begin controlling the TLS120Xe via USB first turn it on. At startup, the TLS120Xe parks the monochromator. The TLS120Xe is set up by default to automatically turn on the Xenon lamp at boot time, but this is user-configurable: the lamp can also be lit by pushing the output button (in local mode) or by sending the command `:LAMP 1` or `:LAMP ON` (in remote mode).

During the lighting process the output LED will be in a red flashing state, and the screen will display device information. Once this process has finished, the LED will turn to solid red, the display changes to show operational information, and the device is ready to be controlled via its SCPI interface (see Physical

Interface).

Here are some example commands

- Set the TLS120Xe to remote mode, set the target output wavelength to 500nm, set the target filter wheel position to one appropriate for a wavelength of 500nm, move the monochromator to target
  - `:SYST:REM`
  - `:MONO 500`
  - `:MONO:FILT:WAVE 500`
  - `:MONO:MOVE`
- Close the shutter by setting the target filter position to 1 (the shutter position), and then move to target
  - `:MONO:FILT 1`
  - `:MONO:MOVE`
- Read the lamp sense voltage and current
  - `:VOLT?` returns `15.3`
  - `:CURR?` returns `5.4`
  - `:IV?` returns `5.4,15.3`
- Turn off the lamp
  - `:LAMP 0`
- Return the TLS120Xe to local mode for control via the front panel
  - `:SYST:LOC`

Communication Protocol

Many of Bentham's instruments, this one included, expose a SCPI command interface. SCPI is chosen as it is a well established standard that provides a good compromise between capability and ease-of-use for the system user or programmer. We provide a short introduction to SCPI below, but much more information is available online, for example:

- Standard Commands for Programmable Instruments (SCPI) Wikipedia page
- SCPI Specification
- Agilent/Keysight SCPI Learning Centre

Physical Interface

The SCPI specification does not specify a particular physical interface (e.g. RS232, GPIB, USB). The majority of Bentham's instruments use a USB connection configured as an HID device: this is readily supported by all operating systems and suitable libraries are available for all major programming and scripting languages. It does however put an upper limit of 64 bytes per line sent to the instrument, although, in practice, this is rarely a problem as SCPI commands are typically around 8-30 bytes long. More care has to be taken if you are sending multiple commands on a single line (which is possible with SCPI, see below).

To issue a command to an instrument:

1. encode the command as an ASCII sequence of bytes;
2. ensure the string is terminated with either a null or newline byte (nothing after this termination will be read)
3. transmit the text as a HID message to the instrument (depending on the library/ language used Windows hosts may require a leading null byte).

Some (but not all) SCPI commands return one or more values (whether they do is explicitly detailed in the command descriptions in this document). When data is returned, it is in ASCII format within a 64-byte HID buffer.

SCPI Basics

SCPI commands are text commands and are designed to be human-readable. Some examples of SCPI commands are:

1. `:SOUR:CURR 5.0` - sets a power source to generate 5.0A DC
2. `:SOUR:VOLT 16.5` - sets a power source to generate 16.5V DC
3. `:SOUR:VOLT?` - gets the current voltage setting from a power source
4. `:MONO:WAVE 654.0` - set a monochromator to 654 nm
5. `:MONO:WAVE?` - get the current wavelength from a monochromator
6. `:MONO:SLIT:BAND 2,3.2` - sets slit 2 of a monochromator to 3.2 nm bandwidth
7. `*IDN?` - get system identification

These examples also show some other key points about SCPI commands:

- Commands that end with a `?` are 'query commands' and will return one or more (comma-separated) values;
- Commands that take a value have a space separating the command from the value;
- Some commands take multiple values, which should be comma-separated;
- There are a number of SCPI standard commands that are supported by all SCPI compliant instruments, these start with an asterisk (`*`).

Parameters & Data Types

Where a SCPI command takes parameters these are well-defined in both their meaning and type. Each command described in this document lists the required parameters, along with their type. Common types are:

- boolean - this can either be sent as OFF or ON, or numerically with 0 for false and 1 for true;
- 32-bit integer - integer value as decimal ASCII text;
- text - an ASCII string, which should be quoted in double quotation marks (");
- IEEE-754 32-bit float - a floating point value as decimal ASCII text.

Returned values

Some SCPI commands return no data, and in this case no buffer is returned by the instrument to the controlling device. These commands are usually the commands that set a variable.

Other SCPI commands return one or more values. These are all returned on a single line, are comma-separated (with no spaces) and terminated with a null byte. Each command description below lists the returned values and their type. Boolean parameters are returned as `0` or `1`, and text is contained within double quotation marks, `"`.

Command abbreviation

If you look through the command descriptions in this document you will see that each command contains a mixture of upper and lower case characters and some contain sections between square brackets (`[` and `]`). In fact, the lower case characters and any section between a `[]` pair are optional. This allows the user to choose between human-readability and command length. Here we show two commands to illustrate, showing the command description followed by some valid abbreviations in use.

- `:MONOchromator[:WAVElength][:SET] <wavelength>`
  - `:MONOchromator:WAVElength:SET 800`
  - `:MONO:WAVE:SET 800`
  - `:MONO:WAVElength 800`
  - `:MONO:WAVE 800`

    - `:MONO 800`
- `[:MEASure]:CURRent?`
    - `:MEASure:CURRent?`
    - `:MEAS:CURR?`
    - `:CURR?`

Multiple commands per line

A single line of text sent to the instrument can contain multiple SCPI commands. Each command just needs to be separated with a `;` (semicolon). For example:

1. `:SOUR:CURR 5.0; :OUTP ON` - sets a power source to generate 5.0A DC and switches it on
2. `:MONO:GRAT 1; :MONO:WAVE 654.0; :MONO:MOVE` - set a monochromator to grating 1 at 654 nm

Hierarchical Commands

SCPI commands are hierarchical in structure. Similar commands are grouped in the tree like structure, just like files in a filesystem. Whereas a filesystem uses a slash character to separate the layers, SCPI uses a `:` (colon).

A leading `:` indicates the root of the command structure, and a command that starts with a `:` is fully-qualified, just like an absolute file path.

Commands without a leading `:` inherit their position from the previous command and hence the fully-qualified name does not need to be sent every time. However, we would recommend that this feature is not widely used when programming for the automated use of the interface as it can lead to difficulty porting to different instruments. The one case where it may be considered is when putting multiple commands on a line, which leads to our recommendation that all lines sent to a SCPI device start with a fully-qualified command.

Synchronisation

As with all machine-to-machine interfaces, some care has to be taken to ensure that the controlling device and instrument remain synchronised or that the controlling device is not sending the instrument commands faster than it can cope with them. We make these recommendations to avoid these issues:

- Do not send more than two consecutive lines of SCPI commands where none of the commands return a value.
- When a SCPI command returns a value wait for it before continuing.
- It is good practice to check the instruments has no pending errors by using the `SYST:ERR:COUNT?` command after setting some values.
- A very robust way to ensure synchronisation is to set a variable and then query it on the same line.

Command Index

```
*CLS
*IDN? ⇒ "<manufacturer>","<model>","<serial_number>","<revision>"
*RCL "profilename"
[:DIAGnostic]:ECHO[:TEXT]? "<echo_to_device>" ⇒ "<echo_from_device>"
:DISPlay:ACTive:BRIGhtness <brightness>
:DISPlay:ACTive:BRIGhtness? ⇒ <brightness>
:DISPlay[:DIMmed]:BRIGhtness <brightness>
:DISPlay[:DIMmed]:BRIGhtness? ⇒ <brightness>
```

:DISPlay[:DIMmed]:DELAY <time>
:DISPlay[:DIMmed]:DELAY? ⇒ <time>
:DISPlay[:ENABle] <state>
:DISPlay[:ENABle]? ⇒ <state>
:FETCh:BURNtime? ⇒ <burntime>
:LAMP <state>
:LAMP:BOOT <boot_mode>
:LAMP:BOOT? ⇒ <boot_mode>
:LAMP? ⇒ <state>
:LAMP:TIMEout <time>
:LAMP:TIMEout? ⇒ <time>
[:MEASure]:BANDWidth? ⇒ <bandwidth>
[:MEASure][:CURRent]:PD:MAX? ⇒ <photocurrent>
[:MEASure][:CURRent]:PD:MEDian? ⇒ <photocurrent>
[:MEASure][:CURRent]:PD? ⇒ <photocurrent>
[:MEASure]:CURRent? ⇒ <measured_current>
[:MEASure]:IV? ⇒ <measured_current>,<measured_voltage>
[:MEASure]:POWer? ⇒ <measured_power>
[:MEASure]:POWer:STDev? ⇒ <power_std>
[:MEASure]:RESistance? ⇒ <measured_resistance>
[:MEASure]:VOLTage? ⇒ <measured_voltage>
:MONOchromator:FILTer:PARK? ⇒ <success>
:MONOchromator:FILTer[:POSition][:GET]? ⇒ <filter_position>,<target>
:MONOchromator:FILTer[:POSition][:SET] <filter_position>
:MONOchromator:FILTer:TABle:DELete <filter_position>
:MONOchromator:FILTer:TABle[:GET]? <filter_position> ⇒
<min_wavelength>,<max_wavelength>
:MONOchromator:FILTer:TABle:SAVE
:MONOchromator:FILTer:TABle[:SET]
<filter_position>,<min_wavelength>,<max_wavelength>
:MONOchromator:FILTer:WAVElength[:SET] <wavelength>
:MONOchromator:GOTO? <wavelength> ⇒ <success>,"<status>"
:MONOchromator:GRATing:CALIBration[:GET]? <grating_number> ⇒
<zord>,<cosk>,<alpha>
:MONOchromator:GRATing:CALIBration:SAVE <grating_number>
:MONOchromator:GRATing:CALIBration[:SET] <grating_number>,<zord>,<cosk>,<alpha>
:MONOchromator:GRATing[:GET]? ⇒
<current_grating_number>,<target_grating_number>
:MONOchromator:GRATing:INFO? <grating_number> ⇒
<ruling_density>,<blaze>,<max_wl>
:MONOchromator:GRATing[:SET] <grating_number>
:MONOchromator:GRATing:TABle:DELete <turret_number>,<grating_number>
:MONOchromator:GRATing:TABle[:GET]? <grating_number> ⇒
<min_wavelength>,<max_wavelength>
:MONOchromator:GRATing:TABle:SAVE
:MONOchromator:GRATing:TABle[:SET]

<grating_number>,<min_wavelength>,<max_wavelength>
:MONOchromator:GRATing:WAVElength[:SET] <wavelength>
:MONOchromator:MOVE:ASYNC
:MONOchromator:MOVE? ⇒ <success>
:MONOchromator:PARK:ASYNC
:MONOchromator:PARK? ⇒ <success>
:MONOchromator:SPEED[:GET]? ⇒ <percent>
:MONOchromator:SPEED[:SET] <percent>
:MONOchromator:STATus? ⇒ <state>
:MONOchromator:TURRet:GRATing:CALIBration[:GET]?
<turret_number>,<grating_number> ⇒ <zord>,<cosk>,<alpha>
:MONOchromator:TURRet:GRATing:CALIBration:SAVE <turret_number>,<grating_number>
:MONOchromator:TURRet:GRATing:CALIBration[:SET]
<turret_number>,<grating_number>,<zord>,<cosk>,<alpha>
:MONOchromator:TURRet:GRATing[:GET]? <turret_number> ⇒
<current_grating_number>,<target_grating_number>
:MONOchromator:TURRet:GRATing:INFO? <turret_number>,<grating_number> ⇒
<ruling_density>,<blaze>,<max_wl>
:MONOchromator:TURRet:GRATing[:SET] <turret_number>,<grating_number>
:MONOchromator:TURRet:GRATing:TABle:DELete <turret_number>,<grating_number>
:MONOchromator:TURRet:GRATing:TABle[:GET]? <turret_number>,<grating_number> ⇒
<min_wavelength>,<max_wavelength>
:MONOchromator:TURRet:GRATing:TABle:SAVE <turret_number>
:MONOchromator:TURRet:GRATing:TABle[:SET]
<turret_number>,<grating_number>,<min_wavelength>,<max_wavelength>
:MONOchromator:TURRet:GRATing:WAVElength[:SET] <turret_number>,<wavelength>
:MONOchromator[:WAVElength][:GET]? ⇒ <wavelength>,<target>
:MONOchromator[:WAVElength][:SET] <wavelength>
[:OUTPut]:ATTarget? ⇒ <state>
[:OUTPut]:WHITe <intensity>
[:PARAmeter]:BANDWidth:CORRection <a0>,<a1>
[:PARAmeter]:BANDWidth:CORRection? ⇒ <a0>,<a1>
[:PARAmeter]:DIMming <knob_deadzone_fraction>,<bandwidth_scaling>
[:PARAmeter]:DIMming? ⇒ <knob_deadzone_fraction>,<bandwidth_scaling>
[:PARAmeter]:WIRE:RESistance <wire_resistance>
[:PARAmeter]:WIRE:RESistance? ⇒ <wire_resistance>
:RESEt:BURNtime
:SYSTem:ERRor:COUNt? ⇒ <number_of_errors>
:SYSTem:ERRor[:NEXT]? ⇒ <error_code>,"<error_msg>"
SYSTem:LOCal
SYSTem:LOCal? ⇒ <local_mode>
[:SYSTem]:OPERating:STATe? ⇒ <state>
SYSTEM:REBOOT
SYSTem:REMote
SYSTem:REMote? ⇒ <remote_mode>

Command Syntax

`*CLS`

This command clears the error buffer.

See Also `SYSTem:ERRor[:NEXT]?`

`*IDN? ⇒ "<manufacturer>","<model>","<serial_number>","<revision>"`

This command returns meta data information from the device.

Return	Type	Description
"<manufacturer>"	text	"Bentham Instruments Ltd."
"<model>"	text	The device's model designation.
"<serial_number>"	text	The device's serial number.
"<revision>"	text	The device's revision number.

`*RCL "profilename"`

This command loads a particular configuration profile and restarts the device.

`[:DIAGnostic]:ECHO[:TEXT]? "<echo_to_device>" ⇒ "<echo_from_device>"`

This command is used to send a string to the device and have the device send it back as a communication diagnostic.

Parameter	Type	Description
"<echo_to_device>"	text	The text to be returned (in quotation marks).

Return	Type	Description
"<echo_from_device>"	text	The returned text.

Examples `:ECHO? "hello!"` returns `"hello!"`

:DISPlay:ACTive:BRIGhtness <brightness>

This command sets the brightness of the illuminated, active display. The brightness parameter has a range of 0 to 1. It is worth noting that a brightness of zero does not disable the screen, this can be achieved using the :DISPlay[:ENABle] command.

Parameter	Type	Description
<brightness>	IEEE-754 32-bit float	The brightness of the active display (where 0 ≤ brightness ≤ 1).

Examples
:DISP:ACT:BRIG 0.5 sets the brightness to 50%
:DISP:ACT:BRIG 1.0 sets the brightness to its maximum.

See Also
:DISPlay[:ENABle]
:DISPlay[:DIMmed]:BRIGhtness
:DISPlay:ACTive:BRIGhtness?

:DISPlay:ACTive:BRIGhtness? ⇒ <brightness>

This command returns the brightness of the illuminated, active display. The brightness parameter has a range of 0 to 1. It is worth noting that a brightness of zero does not disable the screen, this can be achieved using the :DISPlay[:ENABle] command.

Return	Type	Description
<brightness>	IEEE-754 32-bit float	The brightness of the active display (where 0 ≤ brightness ≤ 1).

See Also
:DISPlay[:DIMmed]:BRIGhtness?
:DISPlay:ACTive:BRIGhtness

:DISPlay[:DIMmed]:BRIGhtness <brightness>

This command sets the brightness of the dimmed display. The brightness parameter has a range of 0 to 1. It is worth noting that a brightness of zero does not disable the screen, this can be achieved using the :DISPlay[:ENABle] command.

Parameter	Type	Description
<brightness>	IEEE-754 32-bit float	The brightness of the dimmed display (where 0 ≤ brightness ≤ 1).

Examples
:DISP:BRIG 0.5 sets the brightness to 50%
:DISP:DIMMED:BRIG 1.0 sets the brightness to its maximum.

See Also
:DISPlay[:ENABle]
:DISPlay:ACTive:BRIGhtness
:DISPlay[:DIMmed]:BRIGhtness?

:DISPlay[:DIMmed]:BRIGhtness? ⇒ <brightness>

This command returns the brightness of the dimmed display. The brightness parameter has a range of 0 to 1. It is worth noting that a brightness of zero does not disable the screen, this can be achieved using the :DISPlay[:ENABle] command.

Return	Type	Description
<brightness>	IEEE-754 32-bit float	The brightness of the dimmed display (where 0 ≤ brightness ≤ 1).

See Also :DISPlay[:DIMmed]:BRIGhtness
:DISPlay[:ENABle]
:DISPlay:ACTive:BRIGhtness?

:DISPlay[:DIMmed]:DELAY <time>

This command sets the time the display stays at its full brightness after waking.

Parameter	Type	Description
<time>	IEEE-488.2 Number, optional suffix	Time for which the display stays at its full brightness before it dims again. In seconds, or units can be specified.

Examples :DISP:DELAY 500ms sets the delay to 500ms
:DISP:DELAY 0.5 sets the delay to 500ms

See Also :DISPlay[:DIMmed]:BRIGhtness
:DISPlay:ACTive:BRIGhtness
:DISPlay[:DIMmed]:DELAY?

:DISPlay[:DIMmed]:DELAY? ⇒ <time>

This command returns the time the display stays at its full brightness after waking.

Return	Type	Description
<time>	IEEE-754 32-bit float	Time for which the display stays at its full brightness before it dims again, in seconds

See Also :DISPlay[:DIMmed]:DELAY
:DISPlay[:DIMmed]:BRIGhtness
:DISPlay:ACTive:BRIGhtness

:DISPlay[:ENABle] <state>

This command enables or disables the illuminated display.

Parameter	Type	Description
<state>	boolean (0 or 1)	The target state for the display.

Examples :DISP 0 disables the display.
:DISP 1 enables the display.

See Also :DISPlay[:ENABle]?

:DISPlay[:ENABle]? ⇒ <state>

This command returns the enabled state of the display.

Return	Type	Description
<state>	boolean (0 or 1)	Whether or not the display is enabled

See Also :DISPlay[:ENABle]

:FETCh:BURNtime? ⇒ <burntime>

This command returns the cumulative time in hours for which the lamp has been illuminated since the last burntime reset.

Return	Type	Description
<burntime>	IEEE-754 32-bit float	The cumulative time in hours the output has spent at target since the last burntime reset.

See Also :RESEt:BURNtime

:LAMP <state>

This command begins turns the Xenon lamp on or off.

Parameter	Type	Description
<state>	boolean (0 or 1)	0 turns the lamp off, 1 turns it on.

See Also :LAMP:BOOT
:LAMP?

:LAMP:BOOT <boot_mode>

This command can be used to instruct the TLS120Xe to turn the lamp on or to leave it off at system boot time.

Parameter	Type	Description
<boot_mode>	boolean (0 or 1)	0 leaves the lamp off, 1 turns it on.

See Also :LAMP:BOOT?
:LAMP

:LAMP:BOOT? ⇒ <boot_mode>

This command can be used to query whether the system is set to turn the lamp on upon system boot.

Return	Type	Description
<boot_mode>	boolean (0 or 1)	0 leaves the lamp off, 1 turns it on.

See Also :LAMP:BOOT
:LAMP

:LAMP? ⇒ <state>

This command returns 0 if the TLS120Xe lamp is off, 1 otherwise.

Return	Type	Description
<state>	boolean (0 or 1)	0 if the TLS120Xe lamp is off, 1 otherwise.

See Also :LAMP:BOOT
:LAMP

:LAMP:TIMEout <time>

Set the permitted duration of the lamp lighting process, after which, if the lamp has not yet lit, the attempt is considered unsuccessful. Persistent.

Parameter	Type	Description
<time>	unsigned 32-bit integer	timeout in milliseconds

See Also :LAMP:TIMEout?

:LAMP:TIMEout? ⇒ <time>

Return the permitted duration of the lamp lighting process, after which, if the lamp has not yet lit, the attempt is considered unsuccessful.

Return	Type	Description
<time>	unsigned 32-bit integer	timeout in milliseconds

See Also :LAMP:TIMEout

[:MEASure]:BANDWidth? ⇒ <bandwidth>

If the TLS120Xe is at the target wavelength, this command returns the bandwidth of the light being emitted, otherwise pushes an execution error to the SCPI error queue.

Return	Type	Description
<bandwidth>	IEEE-754 32-bit float	the bandwidth of the TLS120Xe at the current wavelength, in nanometers.

[:MEASure][:CURRent]:PD:MAX? ⇒ <photocurrent>

This command returns the maximum photocurrent measured with the TLS120Xe's built-in feedback photodetector during the current session.

Return	Type	Description
<photocurrent>	IEEE-754 32-bit float	The maximum measured photocurrent, Amperes.

See Also [:MEASure][:CURRent]:PD?
[:MEASure][:CURRent]:PD:MEDian?

[:MEASure][:CURRent]:PD:MEDian? ⇒ <photocurrent>

This command returns the median of the 50 most-recent photocurrent samples measured with the TLS120Xe's built-in feedback photodetector during the last 0.5s.

Return	Type	Description
<photocurrent>	IEEE-754 32-bit float	The median measured photocurrent, Amperes.

See Also [:MEASure][:CURRent]:PD?
[:MEASure][:CURRent]:PD:MAX?

[:MEASure][:CURRent]:PD? ⇒ <photocurrent>

This command returns the most recent of the photocurrent samples measured with the TLS120Xe's built-in feedback photodetector.

Return	Type	Description
<photocurrent>	IEEE-754 32-bit float	The measured photocurrent, Amperes.

See Also [:MEASure][:CURRent]:PD:MEDian?
[:MEASure][:CURRent]:PD:MAX?

[:MEASure]:CURRent? ⇒ <measured_current>

This command returns the measured output current in Amps. This current may differ from the target current if the power supply is in the process of ramping the output to achieve the set target.

Return	Type	Description
<measured_current>	IEEE-754 32-bit float	The actual output current in Amps.

Examples CURRENT? will return 3.004 if the output current was 3.004A.

See Also [:MEASure]:VOLTage?
[:MEASure]:IV?

[:MEASure]:IV? ⇒ <measured_current>,<measured_voltage>

This command returns both the measured output current in Amps and the measured output voltage in Volts. Calling this single compound command is equivalent to calling both the [:MEASure]:CURRent? and [:MEASure]:VOLTage? commands.

Return	Type	Description
<measured_current>	IEEE-754 32-bit float	The measured output current in Amps.
<measured_voltage>	IEEE-754 32-bit float	The output voltage in Volts measured across the sense pins (PSU_610_4WS model) or corrected for the series resistance of the supply wires (PSU_610 model).

Examples :IV? will return 8.251,12 if 8.251A is flowing through the circuit resulting in a potential difference of 12V across ⊕ and ⊖ pins (or the sense pins if available).

See Also [:MEASure]:CURRent?
[:MEASure]:VOLTage?

[:MEASure]:POWer? ⇒ <measured_power>

This command returns the measured output power in Watts. It is calculated using $P = I \times V$
where
V is returned by [:MEASure]:VOLTage?
I is returned by [:MEASure]:CURRent?.

Return	Type	Description
<measured_power>	IEEE-754 32-bit float	The measured output power in Watts.

Examples :POW? will return 150.0 if 10.0A is flowing through the circuit resulting in a potential difference of 15.0V across ⊕ and ⊖ pins (or the sense pins if available).

See Also [:MEASure]:CURRent?
[:MEASure]:VOLTage?

[:MEASure]:POWer:STDev? ⇒ <power_std>

This command returns the standard deviation of the measured power taken over the last 10 internal samples. This command will result in a SCPI execution error if used when the output is off as no samples are available.

Return	Type	Description
<power_std>	IEEE-754 64-bit float	the standard deviation of measured power, Watts.

See Also [:MEASure]:POWer?

[:MEASure]:RESistance? ⇒ <measured_resistance>

This command returns the measured resistance of the load in Ohms. It is calculated using $R = V / I$
where
V is returned by [:MEASure]:VOLTage?
I is returned by [:MEASure]:CURRent?.

Return	Type	Description
<measured_resistance>	IEEE-754 32-bit float	The measured load resistance in Ohms.

Examples :RES? will return 1.0 if 5.0A is flowing through the circuit resulting in a potential difference of 5.0V across ⊕ and ⊖ pins (or the sense pins if available).

See Also [:MEASure]:CURRent?
[:MEASure]:VOLTage?

`[:MEASure]:VOLTage? ⇒ <measured_voltage>`

This command returns the output voltage in Volts measured across the sense pins (PSU_610_4WS model) or corrected for the series resistance of the supply wires (PSU_610 model).

Return	Type	Description
<measured_voltage>	IEEE-754 32-bit float	The output voltage in Volts measured across the sense pins (PSU_610_4WS model) or corrected for the series resistance of the supply wires (PSU_610 model).

Examples

`:VOLT?` will return `11.513` if a potential difference of 11.513V is measured across the ⊕ and ⊖ pins (or the sense pins if available) and the wire resistance parameter is zero.
`:VOLT?` will return `11.6` if a raw potential difference of 12V is measured across the ⊕ and ⊖ pins (or the sense pins if available), 4A of current is flowing, and the wire resistance parameter is 0.1Ω (*11.6V = 12V - 4A*0.1Ω*).

See Also

`[:MEASure]:CURRent?`
`[:MEASure]:IV?`
`[:PARAmeter]:WIRE:RESistance?`
`[:PARAmeter]:WIRE:RESistance`

`:MONOchromator:FILTer:PARK? ⇒ <success>`

not currently implemented

This command parks the monochromator filter wheel. Once the parking procedure has been completed the command returns either true or an error message to indicate the success of the command.
If this command fails then an execution error (code 200) is pushed to the error stack. This indicates the system was not be able to perform a park. This will be because the system was already moving or parking. In this case the following error message is returned instead of the specified return values.
`'Error: System busy'`
Currently this method returns the following error
`'Error: Command not implemented'`

Return	Type	Description
<success>	boolean (0 or 1)	The success of the park command.

Examples

`:MONO:FILT:PARK?` parks the filter wheel and returns true if the park was successful.

See Also

`:MONOchromator:SLIT:PARK?`
`:MONOchromator:SLITS:PARK?`
`:MONOchromator:MOVE?`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

:MONOchromator:FILTer[:POSition][:GET]? ⇒ <filter_position>,<target>

This command returns the current and target filter position of the filter wheel.
If the filter wheel is not enabled then an execution error (code 200) is pushed to the error stack. In this case the following error message is returned instead of the specified return values.
'Error: Filter wheel not enabled'

Return	Type	Description
<filter_position>	unsigned 32-bit integer	The current position of the filter wheel.
<target>	unsigned 32-bit integer	The target position of the filter wheel.

Examples if :MONO:FILT? returned 1,1 this would indicate that the current and target filter position was 1.

See Also :MONOchromator:FILTer[:POSition][:SET]

:MONOchromator:FILTer[:POSition][:SET] <filter_position>

This command sets a new target filter position for the filter wheel. The filter position can take values of 1,...,N depending on the number of positions N of the filter wheel fitted.
If an invalid filter position is specified an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<filter_position>	unsigned 32-bit integer	The target filter position of the filter wheel.

Examples :MONO:FILT 2;:MONO:MOVE? set the target filter position to 2 and triggers the move.

See Also
:MONOchromator:FILTer[:POSition][:GET]?
:MONOchromator:MOVE?
:MONOchromator:FILTer:WAVElength[:SET]
:SYSTem:ERRor:COUNt?
:SYSTem:ERRor[:NEXT]?

:MONOchromator:FILTer:TABle:DELete <filter_position>

This command deletes the wavelength insertion criteria values for a filter with a specific filter position. The new values will be effective, immediately however will not be saved to the monochromator unless a save command is issued.
If an invalid filter position is specified then an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<filter_position>	unsigned 32-bit integer	The position of the filter in the filter wheel (1,...,N).

See Also :MONOchromator:FILTer:TABle:SAVE

:MONOchromator:FILTer:TABle[:GET]? <filter_position> ⇒ <min_wavelength>,<max_wavelength>

This command returns the current wavelength insertion criteria for a filter with a specific filter position. If no wavelength insertion criteria are set for the specified filter position then wavelength bounds of [0, 0) are returned.
If an invalid filter position is specified then an execution error (code 200) is pushed to the error stack. In this case one of the following error messages is returned instead of the specified return values.
'Error: Invalid filter position'
'Error: Filter wheel not found'

Parameter	Type	Description
<filter_position>	unsigned 32-bit integer	The position of the filter in the filter wheel (1,...,N).

Return	Type	Description
<min_wavelength>	IEEE-754 32-bit float	The minimum wavelength (nm) for the filter to be selected (inclusive).
<max_wavelength>	IEEE-754 32-bit float	The maximum wavelength (nm) for the filter to be selected (exclusive).

Examples If :MONO:FILT:TAB? 1 returned 400, 700 then this would indicate that filter position 1 would be used in the wavelength range [400nm, 700nm).

See Also :MONOchromator:FILTer:TABle[:SET]

:MONOchromator:FILTer:TABle:SAVE

This command persistently saves the filter wheel insertion table used by the current filter wheel.

Examples `:MONO:FILT:TAB:SET 1,400,700;:MONO:FILT:TAB:SAVE` sets the wavelength bounds for the filter in filter position 1 and then saves the entire insertion table.

See Also `:MONOchromator:FILTer:TABle[:SET]`
`:MONOchromator:FILTer:TABle:DELete`

:MONOchromator:FILTer:TABle[:SET] <filter_position>,<min_wavelength>,<max_wavelength>

This command sets new values for the wavelength insertion criteria for a filter with a specific filter position. The new values will be effective immediately, however will not be saved to the monochromator unless a save command is issued.
If the wavelength range is invalid (max ≤ min) or an invalid filter position is specified then an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<filter_position>	unsigned 32-bit integer	The position of the filter in the filter wheel (1,...,N).
<min_wavelength>	IEEE-754 32-bit float	The minimum wavelength (nm) for the filter to be selected (inclusive).
<max_wavelength>	IEEE-754 32-bit float	The maximum wavelength (nm) for the filter to be selected (exclusive).

See Also `:MONOchromator:FILTer:TABle:SAVE`

:MONOchromator:FILTer:WAVElength[:SET] <wavelength>

This command chooses a suitable filter for the target wavelength from the insertion table and sets it as the new target filter.
If an invalid wavelength is specified an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<wavelength>	IEEE-754 32-bit float	The target wavelength (nm).

Examples `:MONO:FILT:WAVE 200;:MONO:MOVE?` set the target filter position for 200nm and triggers the move.

See Also `:MONOchromator:FILTer[:POSition][:GET]?`
`:MONOchromator:FILTer[:POSition][:SET]`
`:MONOchromator:MOVE?`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

`:MONOchromator:GOTO? <wavelength> ⇒ <success>,"<status>"`

Set new targets for all components of the monochromator in order to achieve a particular wavelength, and if successful trigger the move. Restores previous targets if unsuccessful.
If this command triggers a change in grating, the monochromator bandwidth may change significantly.
If the wavelength is unachievable, the command reports which components were unable to be configured for the requested wavelength and restores previous targets.

Parameter	Type	Description
<wavelength>	IEEE-754 32-bit float	The target wavelength (nm).

Return	Type	Description
<success>	boolean (0 or 1)	Whether or not the goto command completed successfully.
"<status>"	text	"OK" to indicate that a configuration appropriate for the wavelength was found and the move was attempted, or a message indicating which component could not be configured for the target wavelength. (surrounded by quotation marks).

Examples `:MONO:GOTO? 500.0` selects gratings and filters (if applicable) compatible with a wavelength of 500nm. If successful, trigger the move to 500nm.

See Also
`:MONOchromator:FILTer:TABle[:SET]`
`:MONOchromator:GRATing:TABle[:SET]`
`:MONOchromator[:WAVElength][:SET]`
`:MONOchromator:MOVE?`

:MONOchromator:GRATing:CALIBration[:GET]? <grating_number> ⇒ <zord>,<cosk>,<alpha>

This command delegates to :MONOchromator:TURRet:GRATing:CALIBration[:GET]? with turret number set to 1.
If an invalid grating is specified or turret 1 is invalid then an execution error (code 200) is pushed to the error stack. In this case one of the following error messages is returned instead of the specified return values.
'Error: Invalid turret number'
'Error: Invalid grating number'

Parameter	Type	Description
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3) on turret 1.

Return	Type	Description
<zord>	IEEE-754 32-bit float	The current value of zero order used by the calibration.
<cosk>	IEEE-754 32-bit float	The current value of cos(k) used by the calibration.
<alpha>	IEEE-754 32-bit float	The current value of alpha used by the calibration.

See Also :MONOchromator:TURRet:GRATing:CALIBration[:GET]?

:MONOchromator:GRATing:CALIBration:SAVE <grating_number>

This command delegates to :MONOchromator:TURRet:GRATing:CALIBration:SAVE with turret number set to 1.

Parameter	Type	Description
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3) on turret 1.

See Also :MONOchromator:TURRet:GRATing:CALIBration:SAVE

```
:MONOchromator:GRATing:CALIBration[:SET]
<grating_number>,<zord>,<cosk>,<alpha>
```

This command delegates to `:MONOchromator:TURRet:GRATing:CALIBration[:SET]` with turret number set to 1.

Parameter	Type	Description
`<grating_number>`	unsigned 32-bit integer	The grating number (1, 2, or 3) on turret 1.
`<zord>`	IEEE-754 32-bit float	The value of zero order to be used by the calibration.
`<cosk>`	IEEE-754 32-bit float	The value of cos(k) to be used by the calibration.
`<alpha>`	IEEE-754 32-bit float	The value of alpha to be used by the calibration.

See Also `:MONOchromator:TURRet:GRATing:CALIBration[:SET]`

```
:MONOchromator:GRATing[:GET]? ⇒
<current_grating_number>,<target_grating_number>
```

This command delegates to `:MONOchromator:TURRet:GRATing[:GET]?` with turret number set to 1.
If turret 1 is invalid then an execution error (code 200 is pushed to the error stack. In this case the following error message is returned instead of the specified return values.
`'Error: Invalid turret number'`

Return	Type	Description
`<current_grating_number>`	unsigned 32-bit integer	The current grating number (1, 2, or 3) on turret 1.
`<target_grating_number>`	unsigned 32-bit integer	The target grating number (1, 2, or 3) on turret 1.

See Also `:MONOchromator:TURRet:GRATing[:GET]?`

:MONOchromator:GRATing:INFO? <grating_number> ⇒ <ruling_density>,<blaze>,<max_wl>

This command delegates to :MONOchromator:TURRet:GRATing:INFO? with turret number set to 1. If an invalid grating is specified or turret 1 is invalid then an execution error (code 200) is pushed to the error stack. In this case one of the following error messages is returned instead of the specified return values.

'Error: Invalid turret number'
'Error: Invalid grating number'

Parameter	Type	Description
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3) on turret 1.

Return	Type	Description
<ruling_density>	IEEE-754 32-bit float	The grating's ruling density (lines per mm).
<blaze>	IEEE-754 32-bit float	The grating's blaze wavelength (nm).
<max_wl>	IEEE-754 32-bit float	The grating's maximum usable wavelength (nm).

See Also :MONOchromator:TURRet:GRATing:INFO?

:MONOchromator:GRATing[:SET] <grating_number>

This command delegates to :MONOchromator:TURRet:GRATing[:SET] with turret number set to 1.

Parameter	Type	Description
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3) on turret 1.

See Also :MONOchromator:TURRet:GRATing[:SET]

:MONOchromator:GRATing:TABle:DELete <turret_number>,<grating_number>

This command delegates to :MONOchromator:TURRet:GRATing:TABle:DELete with turret number set to 1.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).

See Also :MONOchromator:TURRet:GRATing:TABle:DELete

:MONOchromator:GRATing:TABle[:GET]? <grating_number> ⇒ <min_wavelength>,<max_wavelength>

This command delegates to :MONOchromator:TURRet:GRATing:TABle[:GET]? with turret set to 1.
If an invalid grating is specified or turret 1 is invalid then an execution error (code 200) is pushed to the error stack. In this case one of the following error messages is returned instead of the specified return values.

'Error: Invalid turret number'
'Error: Invalid grating number'

Parameter	Type	Description
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).

Return	Type	Description
<min_wavelength>	IEEE-754 32-bit float	The minimum wavelength (nm) for the grating to be selected (inclusive).
<max_wavelength>	IEEE-754 32-bit float	The maximum wavelength (nm) for the grating to be selected (exclusive).

See Also :MONOchromator:TURRet:GRATing:TABle[:GET]?

:MONOchromator:GRATing:TABle:SAVE

This command delegates to :MONOchromator:TURRet:GRATing:TABle:SAVE with turret number set to 1.

See Also :MONOchromator:TURRet:GRATing:TABle:SAVE

:MONOchromator:GRATing:TABle[:SET] <grating_number>,<min_wavelength>,<max_wavelength>

This command delegates to :MONOchromator:TURRet:GRATing:TABle[:SET] with turret number set to 1.

Parameter	Type	Description
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).
<min_wavelength>	IEEE-754 32-bit float	The minimum wavelength (nm) for the grating to be selected (inclusive).
<max_wavelength>	IEEE-754 32-bit float	The maximum wavelength (nm) for the grating to be selected (exclusive).

See Also :MONOchromator:TURRet:GRATing:TABle[:SET]

:MONOchromator:GRATing:WAVElength[:SET] <wavelength>

This command delegates to :MONOchromator:TURRet:GRATing:WAVElength[:SET] with turret number set to 1.

Parameter	Type	Description
<wavelength>	IEEE-754 32-bit float	The target wavelength.

See Also :MONOchromator:TURRet:GRATing:WAVElength[:SET]

:MONOchromator:MOVE:ASYNC

This command will cause all aspects of the monochromator to start moving to their current respective targets. If this command is issued before a park command has been issued, then an automatic park command is performed first, followed by the move command.
If this command fails then an then an execution error (code 200) is pushed to the error stack. This will either be because the monochromator's target position is inconsistent or impossible, or the monochromator hasn't finished a previous MOVE or PARK command.

Examples :MONO:MOVE:ASYNC moves the monochromator to current targets.
:MONO:WAVE 500.0;:MONO:MOVE:ASYNC sets the target wavelength of the monochromator to 500nm and then triggers the move to 500nm.

See Also
:MONOchromator:MOVE?
:MONOchromator:PARK?
:MONOchromator[:WAVElength][:SET]
:SYSTem:ERRor:COUNt?
:SYSTem:ERRor[:NEXT]?

:MONOchromator:MOVE? ⇒ <success>

This command simultaneously moves all aspects of the monochromator to their current respective targets. If this command is issued before a park command has been issued, then an automatic park command is performed first, followed by the move command. The command returns either true or an error message to indicate the success of the command.
If this command fails then an execution error (code 200) is pushed to the error stack. This is because either some aspects of the monochromator do not have targets set, or the monochromator is busy moving. In this case one of the following errors will be returned
`'Error: Targets not set'`
`'Error: System busy'`
Note: for backwards compatibility the command `:MONOchromator:MOVE` behaves identically to this command. However, it should not be used in new designs.

Return	Type	Description
<success>	boolean (0 or 1)	The success of the move command.

Examples `:MONO:MOVE?` moves the monochromator to current targets.
`:MONO:WAVE 500.0;:MONO:MOVE?` sets the target wavelength of the monochromator to 500nm and then triggers the move to 500nm.

See Also
`:MONOchromator:MOVE:ASYNC`
`:MONOchromator:PARK?`
`:MONOchromator[:WAVElength][:SET]`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

:MONOchromator:PARK:ASYNC

This command commences the simultaneous parking of all aspects of the monochromator including the turret, filter wheel and slits.
If this command fails then an execution error (code 200) is pushed to the error stack. This indicates the system was not be able to perform a park. This will be because the system was already moving or parking.

Examples `:MONO:PARK:ASYNC` starts the parking of the monochromator.

See Also
`:MONOchromator:PARK?`
`:MONOchromator:MOVE:ASYNC`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

:MONOchromator:PARK? ⇒ <success>

This command simultaneously parks all aspects of the monochromator including the turret, filter wheel and slits. Once the parking procedure is complete the command returns either true or an error message to indicate the success of the command.
If this command fails then an execution error (code 200) is pushed to the error stack. This indicates the system was not be able to perform a park. This will be because the system was already moving or parking. In this case the following error message is returned instead of the specified return values.
`'Error: System busy'`
Note: For backwards compatibility the command `:MONOchromator:PARK` behaves identically to this command. However, it should not be used in new designs.

Return	Type	Description
<success>	boolean (0 or 1)	The success of the park command.

Examples `:MONO:PARK?` parks the monochromator and returns true if the park was successful.

See Also
`:MONOchromator:PARK:ASYNC`
`:MONOchromator:SLIT:PARK?`
`:MONOchromator:SLITS:PARK?`
`:MONOchromator:FILTER:PARK?`
`:MONOchromator:MOVE?`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

:MONOchromator:SPEED[:GET]? ⇒ <percent>

This command retrieves the monochromator's current speed setting relative to default, in percent. As not all speeds are available the value returned may not match the value set with `:MONO:SPEED` precisely.

Return	Type	Description
<percent>	IEEE-754 32-bit float	the percentage of the default speed, 1-100

Examples `:MONO:SPEED?` will report the monochromator's current speed.

See Also `:MONOchromator:SPEED`

:MONOchromator:SPEED[:SET] <percent>

This command sets the speed of the monochromator's turret(s) to a percentage of the default speed. Not all speeds are available, in which case the nearest possible speed to the requested speed will be applied. The value can be queried with :MONO:SPEED?

Parameter	Type	Description
<percent>	IEEE-754 32-bit float	the percentage of the default speed, 1-100

Examples :MONO:SPEED 50 will cause the monochromator to move at half speed.

See Also :MONOchromator:SPEED?

:MONOchromator:STATus? ⇒ <state>

This command returns the current status of the monochromator as one of the strings 'not_initialized', 'moving', 'idle' or 'error'.

Return	Type	Description
<state>	string	The current state of the monochromator.

`:MONOchromator:TURRet:GRATing:CALIBration[:GET]?`
`<turret_number>,<grating_number>` ⇒ `<zord>,<cosk>,<alpha>`

This command returns the current calibration settings for a grating with specific turret and grating numbers, for the active ports (if applicable).
In general the grating number can take values of 1, 2 or 3. However the maximum allowed number will depend on how many gratings are physically fitted in the monochromator.
If an invalid grating or turret is specified then an execution error (code 200) is pushed to the error stack. In this case one of the following error messages is returned instead of the specified return values.
`'Error: Invalid turret number'`
`'Error: Invalid grating number'`

Parameter	Type	Description
`<turret_number>`	unsigned 32-bit integer	The turret number (1 or 2).
`<grating_number>`	unsigned 32-bit integer	The grating number (1, 2, or 3).

Return	Type	Description
`<zord>`	IEEE-754 32-bit float	The current value of zero order used by the calibration.
`<cosk>`	IEEE-754 32-bit float	The current value of cos(k) used by the calibration.
`<alpha>`	IEEE-754 32-bit float	The current value of alpha used by the calibration.

See Also `:MONOchromator:TURRet:GRATing:CALIBration[:SET]`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

`:MONOchromator:TURRet:GRATing:CALIBration:SAVE`
`<turret_number>,<grating_number>`

This command sets new values for the calibration settings for a grating with specific turret and grating numbers, for the active ports (if applicable). The new values will be effective immediately however will not be saved to the monochromator unless a save command is issued. The grating number must match the currently active grating.

Parameter	Type	Description
`<turret_number>`	unsigned 32-bit integer	The turret number (1 or 2).
`<grating_number>`	unsigned 32-bit integer	The grating number (1, 2, or 3).

See Also `:MONOchromator:TURRet:GRATing:CALIBration[:GET]?`
`:MONOchromator:TURRet:GRATing:CALIBration[:SET]`

```
:MONOchromator:TURRet:GRATing:CALIBration[:SET]
<turret_number>,<grating_number>,<zord>,<cosk>,<alpha>
```

This command sets new values for the calibration settings for a grating with specific turret and grating numbers, for the active ports (if applicable). The new values will be effective immediately however will not be saved to the monochromator unless a save command is issued. The grating number must match the currently active grating.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).
<zord>	IEEE-754 32-bit float	The value of zero order to be used by the calibration.
<cosk>	IEEE-754 32-bit float	The value of cos(k) to be used by the calibration.
<alpha>	IEEE-754 32-bit float	The value of alpha to be used by the calibration.

See Also :MONOchromator:TURRet:GRATing:CALIBration[:GET]?
:MONOchromator:TURRet:GRATing:CALIBration:SAVE

```
:MONOchromator:TURRet:GRATing[:GET]? <turret_number> ⇒
<current_grating_number>,<target_grating_number>
```

This command returns the current and target grating numbers (1, 2 or 3) for a specified turret number (1 or 2). When the monochromator is power cycled the grating number is set to 1 as this is always a valid grating number.
If an turret number is specified then an execution error (code 200) is pushed to the error stack. In this case the following error message is returned instead of the specified return values.
`'Error: Invalid turret number'`

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).

Return	Type	Description
<current_grating_number>	unsigned 32-bit integer	The current grating number (1, 2, or 3).
<target_grating_number>	unsigned 32-bit integer	The target grating number (1, 2, or 3).

See Also :MONOchromator:TURRet:GRATing[:SET]

`:MONOchromator:TURRet:GRATing:INFO? <turret_number>,<grating_number> ⇒ <ruling_density>,<blaze>,<max_wl>`

Retrieve grating ruling density, blaze, maximum wavelength for specified grating and turret numbers.
If an invalid grating or turret is specified then an execution error (code 200) is pushed to the error stack.
In this case one of the following error messages is returned instead of the specified return values.
`'Error: Invalid turret number'`
`'Error: Invalid grating number'`

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).

Return	Type	Description
<ruling_density>	IEEE-754 32-bit float	The grating's ruling density (lines per mm).
<blaze>	IEEE-754 32-bit float	The grating's blaze wavelength (nm).
<max_wl>	IEEE-754 32-bit float	The grating's maximum usable wavelength (nm).

:MONOchromator:TURRet:GRATing[:SET] <turret_number>,<grating_number>

This command is used to specify which grating to use for for a specified turret for all subsequent :MONOchromator[:WAVElength][:SET] commands. Once this command is sent the wavelength position and target are set to nan. Therefore to move the monochromator to a wavelength position on a new grating the :MONO:TURR:GRAT, :MONO:WAVE and :MONO:MOVE? commands need to be sent.
In general the grating number can take values of 1, 2 or 3. However the maximum allowed number will depend on how many gratings are physically fitted in the monochromator.
If an invalid grating number is specified an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).

Examples

:MONO:GRAT 1, 1;:MONO 500.0;:MONO:MOVE? will cause the monochromator to move to a wavelength target of 500.0nm on grating 1.
:MONO:TURR:GRAT 1, 1;:MONO 500.0;:MONO:GRAT 2;:MONO:MOVE? will **also** cause the monochromator to move to a wavelength target of 500.0nm on grating 1. Subsequent wavelength changes will occur on grating 2.
:MONO:TURR:GRAT 1, 2;:MONO 500.0;:MONO:MOVE? will cause the monochromator to move to a wavelength target of 500.0nm on grating 2.
:MONO:TURR:GRAT 1, 2;:MONO:WAVE? will return nan,nan.

See Also

:MONOchromator:TURRet:GRATing[:GET]?
:MONOchromator[:WAVElength][:SET]
:MONOchromator:MOVE?
:SYSTem:ERRor:COUNt?
:SYSTem:ERRor[:NEXT]?

:MONOchromator:TURRet:GRATing:TABle:DELete <turret_number>,<grating_number>

This command deletes the wavelength bounds criteria values for a grating with a specific grating number on the specified turret. The new values will be effective immediately, however will not be saved to the monochromator unless a save command is issued.
If an invalid grating position is specified then an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1, or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).

See Also

:MONOchromator:TURRet:GRATing:TABle:SAVE

:MONOchromator:TURRet:GRATing:TABle[:GET]? <turret_number>,<grating_number> ⇒ <min_wavelength>,<max_wavelength>

This command returns the current wavelength bounds criteria for the specified grating on the specified turret. If no wavelength criteria are set for the specified grating then wavelength bounds of [0, 0) are returned.
If an invalid grating or turret is specified then an execution error (code 200) is pushed to the error stack. In this case one of the following error messages is returned instead of the specified return values.
'Error: Invalid turret number'
'Error: Invalid grating number'

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).

Return	Type	Description
<min_wavelength>	IEEE-754 32-bit float	The minimum wavelength (nm) for the grating to be selected (inclusive).
<max_wavelength>	IEEE-754 32-bit float	The maximum wavelength (nm) for the grating to be selected (exclusive).

Examples If :MONO:TURR:GRAT:TAB? 1,1 returned 400,700 then this would indicate that grating position 1 would be used on turret 1 in the wavelength range [400nm, 700nm).

See Also :MONOchromator:TURRet:GRATing:TABle[:SET]

:MONOchromator:TURRet:GRATing:TABle:SAVE <turret_number>

This command persistently saves the grating bounds table by which MONO:GOTO chooses a grating for a specified turret number.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).

Examples :MONO:TURR:GRAT:TAB:SET 1,1,400,700;:MONO:TURR:GRAT:TAB:SAVE 1 sets the wavelength bounds for the grating in grating position 1 on turret 1 and then saves the entire bounds table of turret 1.

See Also :MONOchromator:TURRet:GRATing:TABle[:SET]
:MONOchromator:TURRet:GRATing:TABle:DELete

:MONOchromator:TURRet:GRATing:TABle[:SET] <turret_number>,<grating_number>,<min_wavelength>,<max_wavelength>

This command sets new values for the wavelength bounds criteria for a grating with a specific grating position and turret number. The new values will be effective immediately however, will not be saved to the monochromator unless a save command is issued.
If the wavelength range is invalid (max ≤ min) or an invalid grating position is specified then an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<grating_number>	unsigned 32-bit integer	The grating number (1, 2, or 3).
<min_wavelength>	IEEE-754 32-bit float	The minimum wavelength (nm) for the grating to be selected (inclusive).
<max_wavelength>	IEEE-754 32-bit float	The maximum wavelength (nm) for the grating to be selected (exclusive).

See Also :MONOchromator:TURRet:GRATing:TABle:SAVE

:MONOchromator:TURRet:GRATing:WAVElength[:SET] <turret_number>,<wavelength>

This command using the current grating wavelength table to choose a suitable grating for the specified turret given a target wavelength. If a suitable choice is found the grating target is set, otherwise an execution error (code 200) is pushed to the error stack.

Parameter	Type	Description
<turret_number>	unsigned 32-bit integer	The turret number (1 or 2).
<wavelength>	IEEE-754 32-bit float	The target wavelength in nm.

Examples :MONO:TURR:GRAT:WAVE 1, 200;:MONO:MOVE? sets the target grating for 200nm and triggers the move.

See Also
:MONOchromator:TURRet:GRATing[:GET]?
:MONOchromator:TURRet:GRATing[:SET]
:MONOchromator:MOVE?
:SYSTem:ERRor:COUNt?
:SYSTem:ERRor[:NEXT]?

:MONOchromator[:WAVElength][:GET]? ⇒ <wavelength>,<target>

This command returns the current and target wavelength of the monochromator in nm. These two values may not be the same if a new target has been set but not moved to.

Return	Type	Description
<wavelength>	IEEE-754 32-bit float	The current wavelength in nm.
<target>	IEEE-754 32-bit float	The target wavelength in nm.

Examples if `:MONO:WAVE?` returned `250.0,500.0` this would indicate the current wavelength was 250.0nm and the target was 500.0nm. In this state, issuing a `:MONO:MOVE?` command would cause the monochromator to move from 250.0nm to 500.0nm.

See Also
`:MONOchromator[:WAVElength][:SET]`
`:MONOchromator:MOVE?`

:MONOchromator[:WAVElength][:SET] <wavelength>

This command sets a new target wavelength in nm for the monochromator. It does not cause the monochromator to move to the specified target as this done by the move command.
If an invalid wavelength is specified an execution error (code 200) is pushed to the error stack. An invalid wavelength can be caused by an out of bounds exception when being compared to the current grating wavelength limits.

Parameter	Type	Description
<wavelength>	IEEE-754 32-bit float	The target wavelength in nm.

Examples `:MONO:WAVE 500.0` sets the current target wavelength to 500.0nm.

See Also
`:MONOchromator[:WAVElength][:GET]?`
`:MONOchromator:MOVE?`
`:SYSTem:ERRor:COUNt?`
`:SYSTem:ERRor[:NEXT]?`

[:OUTPut]:ATTarget? ⇒ <state>

This command returns the at-target state of the TLS120Xe indicating whether or not the system is at the target wavelength, with light being emitted from the exit port. This is also indicated by the output LED, where solid green light is equivalent to at-target being true.

Return	Type	Description
<state>	boolean (0 or 1)	1 indicates that the system is emitting light at the target wavelength, 0 otherwise.

[:OUTPut]:WHITe <intensity>

This command sets the output of the TLS120Xe to white light with the intensity reduced to approximately match the brightness parameter, by turning the grating to near its zero order reflection angle.

Parameter	Type	Description
<intensity>	IEEE-754 32-bit float	the target brightness between 0 and 1, where 1 represents the maximum brightness.

See Also [:PARAmeter]:DIMming
[:PARAmeter]:DIMming?

[:PARAmeter]:BANDWidth:CORRection <a0>,<a1>

This command sets the parameters to the linear function that is used to correct the theoretical bandwidth:
$Bandwidth = a0 + a1 \times Bandwidth_{theo}$

Parameter	Type	Description
<a0>	IEEE-754 32-bit float	Offset term, in nanometers, defaults to zero.
<a1>	IEEE-754 32-bit float	Scaling term, unitless, defaults to one.

See Also [:PARAmeter]:BANDWidth:CORRection?

[:PARAmeter]:BANDWidth:CORRection? ⇒ <a0>,<a1>

This command returns the parameters to the linear function that is used to correct the theoretical bandwidth:
$Bandwidth = a0 + a1 \times Bandwidth_{theo}$

Return	Type	Description
<a0>	IEEE-754 32-bit float	Offset term, in nanometers, defaults to zero.
<a1>	IEEE-754 32-bit float	Scaling term, unitless, defaults to one.

See Also [:PARAmeter]:BANDWidth:CORRection

`[:PARAmeter]:DIMming <knob_deadzone_fraction>,<bandwidth_scaling>`

This command sets the parameters controlling the dimmable white light output functionality.

Parameter	Type	Description
<knob_deadzone_fraction>	IEEE-754 32-bit float	Sensitivity parameter indicating what fraction of the total value range the knob needs to be turned before the value changes. Typically 0.1.
<bandwidth_scaling>	IEEE-754 32-bit float	Dimming is achieved by linearly de-tuning the monochromator from Zero Order output. The magnitude of the de-tuning is proportional to the desired dimming, the bandwidth, and the bandwidth_scaling parameter, which allows the dimming to be fine-tuned.

See Also `[:PARAmeter]:DIMming?`
`[:OUTPut]:WHITe`

`[:PARAmeter]:DIMming? ⇒ <knob_deadzone_fraction>,<bandwidth_scaling>`

This command returns the parameters controlling the dimmable white light output functionality.

Return	Type	Description
<knob_deadzone_fraction>	IEEE-754 32-bit float	Sensitivity parameter indicating what fraction of the total value range the knob needs to be turned before the value changes. Typically 0.1.
<bandwidth_scaling>	IEEE-754 32-bit float	Dimming is achieved by linearly de-tuning the monochromator from Zero Order output. The magnitude of the de-tuning is proportional to the desired dimming, the bandwidth, and the bandwidth_scaling parameter, which allows the dimming to be fine-tuned. Typically 1.0

See Also `[:PARAmeter]:DIMming`
`[:OUTPut]:WHITe`

[:PARAmeter]:WIRE:RESistance <wire_resistance>

This command sets the value of the wire resistance of the lamp wires in Ohms. Once set a resistor of this value will assumed to be in series with the lamp when measured voltage calculations are made.

Parameter	Type	Description
<wire_resistance>	IEEE-754 32-bit float	The wire resistance in Ohms to be stored in persistent memory.

Examples After `WIRE:RES 0.1` the calculation of all voltages will take into account a 0.1Ω in series resistance.

See Also `[:PARAmeter]:WIRE:RESistance?`

[:PARAmeter]:WIRE:RESistance? ⇒ <wire_resistance>

This command returns the value of the wire resistance in Ohms. It is assumed to be in series with the output load and used to correct the measured output voltage returned by the commands like `[:MEASure]:VOLTage?`. The factory default value for this parameter is zero.

Return	Type	Description
<wire_resistance>	IEEE-754 32-bit float	The wire resistance in Ohms.

Examples `WIRE:RES?` returns `0.1` if the wire resistance parameter has been previously set to 0.1Ω.

See Also `[:PARAmeter]:WIRE:RESistance`

:RESEt:BURNtime

This command resets to zero the persistent parameter tracking the cumulative time the lamp has been illuminated since the last burntime reset.

See Also `:FETCh:BURNtime?`

:SYSTem:ERRor:COUNt? ⇒ <number_of_errors>

This command returns the number of errors currently in the error queue.

Return	Type	Description
<number_of_errors>	32-bit integer	The number of errors on the error queue.

See Also `:SYSTEM:ERRor[:NEXT]?`

:SYSTem:ERRor[:NEXT]? ⇒ <error_code>,"<error_msg>"

This command pops the next error from the start of the error queue and returns information about the error.
If there was no error then error code 0 and information "No Error" are returned.

Return	Type	Description
<error_code>	32-bit integer	A numeric value identifying the error.
"<error_msg>"	text	A string of characters describing the error.

Examples `BAD:COMMAND` followed by `:SYST:ERR?` results in `-113,"Undefined header"`
a subsequent `:SYST:ERR?` results in `0,"No error"`

See Also `:SYSTem:ERRor:COUNt?`

SYSTem:LOCal

This command sets the system to local mode, preventing certain functionality from being executed remotely until remote mode is activated again. Not available on all devices. Mutually exclusive with remote mode.

See Also `SYSTEM:REMote`
`SYSTEM:LOCal?`

SYSTem:LOCal? ⇒ <local_mode>

This command returns a boolean identifying whether the system is in local mode or not. Not available on all devices. Mutually exclusive with remote mode.

Return	Type	Description
<local_mode>	boolean (0 or 1)	1 if the system is in local mode, 0 otherwise

See Also `SYSTem:LOCal`
`SYSTem:REMote?`

[:SYSTem]:OPERating:STATe? ⇒ <state>

This command can be used to query the current operating state of the TLS120Xe.

Return	Type	Description
<state>	string	one of "INVALID", "STARTUP", "INITIALIZING", "SYSTEM_SETUP", "OUTPUT_OFF", "MOVING_TO_TARGET", "AT_TARGET", "LAMP_FAILED", "LAMP_OFF" or "UNDEFINED".

SYSTEM:REBOOT

This command restarts the control electronics of the device. This command can not be used in conjunction with other SCPI commands as it is processed separately. It is not in the SCPI command hierarchy and does not use a leading colon (:) but is inherently fully qualified. Unsaved changes will be lost.

SYSTem:REMote

This command sets the system to remote mode, preventing certain functionality from being executed locally until local mode is activated again. Not available on all devices. Mutually exclusive with local mode.

See Also SYSTEM:LOCal
SYSTEM:REMote?

SYSTem:REMote? ⇒ <remote_mode>

This command returns a boolean identifying whether or not the system is in remote mode. Not available on all devices. Mutually exclusive with local mode.

Return	Type	Description
<remote_mode>	boolean (0 or 1)	1 if the system is in remote mode, 0 otherwise

See Also SYSTEM:REMote
SYSTEM:LOCal?