PM103, PM103A, PM103U, PM103E

Write Your Own Application

- Drivers
- PC Interface
- Programmer's Reference

Revision: B
Date: September 25, 2025

1 Safety

Attention

The safety of any system incorporating the equipment is the responsibility of the assembler of the system.

All statements regarding safety of operation and technical data in this user guide will only apply when the unit is operated correctly as it was designed for.

The PM103x must not be operated in explosion endangered environments!

Only with written consent from Thorlabs may changes to single components be made or components not supplied by Thorlabs be used.

All modules including control inputs / outputs and the sensor must be connected with duly shielded connection cables.

Attention

The following statement applies to the products covered in this user guide, unless otherwise specified herein. The statement for other products will appear in the accompanying documentation.

Note

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules and meets all requirements of the Canadian Interference-Causing Equipment Standard ICES-003 for digital apparatus. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio or television technician for help.

Thorlabs GmbH is not responsible for any radio or television interference caused by modifications of this equipment or the substitution or attachment of connecting cables and equipment other than those specified by Thorlabs GmbH. The correction of interference caused by such unauthorized modification, substitution or attachment will be the responsibility of the user.

The use of shielded I/O cables is required when connecting this equipment to any and all optional peripheral or host devices. Failure to do so may violate FCC and ICES rules.

Attention

Mobile telephones, cellular phones or other radio transmitters are not to be used within the range of three meters of this unit since the electromagnetic field intensity may then exceed the maximum allowed disturbance values according to IEC 61326-1.

This product has been tested and found to comply with the limits according to IEC 61326-1 for using connection cables shorter than 3 meters (9.8 feet).

2 Write Your Own Application

Thorlabs provides all information to write custom made applications for Thorlabs products.

In order to write a custom made application, a specific instrument driver and some tools for use in different programming environments are required.

For PM103x, all information can be found in the documentation WYOA for PM103, PM103A, PM103E, and PM103U, which can be downloaded from the Thorlabs website.

3 Programmers Reference PM103x

3.1 SCPI Commands

SCPI (Standard Commands for Programmable Instruments) is an ASCII-based instrument command language designed for test and measurement instruments.

Note

The commands listed in this section are supported by a USBTMC protocol and can be used with the instrument driver TLPM.dll.

3.2 Introduction to the SCPI Language

SCPI commands are based on a hierarchical structure, also known as a tree system. In this system, associated commands are grouped together under a common node or root, thus forming subsystems. A portion of the SENSE subsystem is shown below to illustrate the tree system.

```
[SENSe:]
  CORRection
       :COLLect
          :ZERO
             [:INITiate]
             :ABORt
             :STATe?
             :MAGNitude?
       :BEAMdiameter {MINimum|MAXimum|DEFault|<numeric_value>[mm]}
       :BEAMdiameter? [{MINimum|MAXimum|DEFault}]
       :WAVelength {MINimum|MAXimum|<numeric_value>[nm]}
       :WAVelength? [{MINimum|MAXimum}]
       :POWer
           [:PDIode]
              [:RESPonse] MINimum|MAXimum|DEFault|<numeric_value>[A]}
              [:RESPonse]? [{MINimum|MAXimum|DEFault}]
           :THERmopile
              [:RESPonse] {MINimum|MAXimum|DEFault|<numeric_value>[V]}
              [:RESPonse]? [{MINimum|MAXimum|DEFault}]
```

SENSe is the root keyword of the command, CORRection is the second-level keyword, and COLLect and BEAMdiameter are third-level keywords, and so on.

A colon (**:**) separates a command keyword from a lower-level keyword.

Command Format

The format used to show commands in this user guide is shown below:

```
CURRent[:DC]:RANGe {MINimum|MAXimum|<numeric_value>[A]}
CORRection:BEAMdiameter {MINimum|MAXimum|DEFault|<numeric_value>[mm]}
```

The command syntax shows most commands (and some parameters) as a mixture of upper- and lower-case letters. The upper-case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, send the long form.

For example, in the above syntax statement, CURR and CURRENT are both acceptable forms. You can use upper- or lower-case letters. Therefore, CURRENT, `current` and `Current` are all acceptable. Other forms, such as CUR and CURREN, will generate an error.

Braces (**{ }**) enclose the parameter choices for a given command string. The braces are not sent with the command string. A vertical bar (**|**) separates multiple parameter choices for a given command string.

Triangle brackets (**< >**) indicate that you must specify a value for the enclosed parameter. For example, the above syntax statement shows the range parameter enclosed in triangle brackets. The brackets are not sent with the command string. You must specify a value for the parameter (such as `"CURR:DC:RANG 50E-6"`).

Some parameters are enclosed in square brackets (**[ ]**). The brackets indicate that the parameter is optional and can be omitted. The brackets are not sent with the command string. In this example `[:DC]` can be omitted, so the command string can be shortened to `"CURR:RANG 50E-6"`. If you do not specify a value for an optional parameter, the power/energy meter chooses a default value.

Command Separators

A colon (**:**) is used to separate a command keyword from a lower-level keyword. You must insert a blank space to separate a parameter from a command keyword. If a command requires more than one parameter, you must separate adjacent parameters using a comma as shown below:

`"SYST:TIME 10, 34, 48"`

A semicolon (**;**) is used to separate commands within the same subsystem, and can also minimize typing. For example, sending the following command string:

`"CORR:BEAM 1; WAVE 1310"`

... is the same as sending the following two commands:

`"CORR:BEAM 1"`

`"CORR:WAVE 1310"`

Use a colon and a semicolon to link commands from different subsystems. For example, in the following command string, an error is generated if you do not use both the colon and semicolon:

`"CORR:BEAM 1;:AVER 300"`

Using the MIN and MAX Parameters

You can substitute `MINimum` or `MAXimum` in place of a parameter for many commands. For example, consider the following command:

`CURRent[:DC]:RANGe {MINimum|MAXimum|<numeric_value>[A]}`

Instead of selecting a specific current range, you can substitute MIN to set the range to its minimum value or MAX to set the range to its maximum value.

Querying Parameter Settings

You can query the current value of most parameters by adding a question mark (?) to the command. For example, the following command sets the operating wavelength to 1550 nm: `"CORR:WAVE 1550"`

You can query the operating wavelength by executing: `"CORR:WAVE?"`.

You can also query the minimum or maximum operating wavelength allowed as follows:

`"CORR:WAVE? MIN"`

`"CORR:WAVE? MAX"`

Caution

If you send two query commands without reading the response from the first, and then attempt to read the second response, you may receive some data from the first response followed by the complete second response. To avoid this, do not send a query command without reading the response. When you cannot avoid this situation, send a device clear before sending the second query command.

SCPI Command Terminators

A command string sent to the power/energy meter must terminate with a <new line> character. The IEEE-488 EOI (end-or-identify) message is interpreted as a <new line> character and can be used to terminate a command string in place of a <new line> character. A <carriage return> followed by a <new line> is also accepted. Command string termination will always reset the current SCPI command path to the root level.

IEEE488.2 Common Commands

The IEEE-488.2 standard defines a set of common commands that perform functions like reset, self-test, and status operations. Common commands always begin with an asterisk (*), are four to five characters in length, and may include one or more parameters. The command keyword is separated from the first parameter by a blank space. Use a semicolon (;) to separate multiple commands as shown below:

```
"*RST; *CLS; *ESE 32; *OPC?"
```

SCPI Parameter Types

The SCPI language defines several different data formats to be used in program messages and response messages.

Numeric Parameters

Commands that require numeric parameters will accept all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation.

Special values for numeric parameters like `MINimum`, `MAXimum` and `DEFault` are also accepted. You can also send engineering unit suffixes with numeric parameters (e.g., `M`, `K`, or `U`). If only specific numeric values are accepted, the power/energy meter will automatically round the input numeric parameters. The following command uses a numeric parameter:

```
POWer:REFerence {MINimum|MAXimum|DEFault|<numeric_value>[W]}
```

Discrete Parameters

Discrete parameters are used to program settings that have a limited number of values (like `W`, `DBM`). They can have a short form and a long form just like command keywords. You can mix upper- and lower-case letters. Query responses will always return the short form in all upper-case letters. The following command uses discrete parameters:

```
POW:UNIT {W|DBM}
```

Boolean Parameters

Boolean parameters represent a single binary condition that is either true or false. For a false condition, the power/energy meter will accept `"OFF"` or `"0"`. For a true condition, the meter will accept `"ON"` or `"1"`. When you query a boolean setting, the instrument will always return `"0"` or `"1"`. The following command uses a boolean parameter:

```
CURRent:RANGe:AUTO {OFF|0|ON|1}
```

String Parameters

String parameters can contain virtually any set of ASCII characters. A string must begin and end with matching quotes; either with a single quote or with a double quote. You can include the quote delimiter as part of the string by typing it twice without any characters in between. The following command uses a string parameter:

```
DIAG:CALString <quoted string>
```

3.3 IEEE488.2 Common Commands

Common commands are device commands that are common to all devices according to the IEEE488.2 standard. These commands are designed and defined by this standard. Most of the commands are described in detail in this section. The following common commands associated with the status structure are covered in the "Status Structure" section: *CLS, *ESE, *ESE?, *ESR?, *SRE, *SRE?, *STB?

3.3.1 Command Summary

Mnemonic	Name	Description
***CLS**	Clear status	Clears all event registers and Error Queue
***ESE <NRf>**	Event enable command	Program the Standard Event Enable Register
***ESE?**	Event enable query	Read the Standard Event Enable Register
***ESR?**	Event status register query	Read and clear the Standard Event Register
***IDN?**	Identification query	Read the unit's identification string
***OPC**	Operation complete command	Set the Operation Complete bit in the Standard Event Register
***OPC?**	Operation complete query	Places a "1" into the output queue when all device operations have been completed
***RST**	Reset command	Returns the unit to the *RST default condition
***SRE <NRf>**	Service request enable command	Programs the Service Request Enable Register
***SRE?**	Service request enable query	Reads the Service Request Enable Register
***STB?**	Status byte query	Reads the Status Byte Register
***TST?**	Self-test query	Performs the unit's self-test and returns the result.
***WAI**	Wait-to-continue command	Wait until all previous commands are executed

3.3.2 Command Reference

***IDN? – identification query - read identification code**

The identification code includes the manufacturer, model code, serial number, and firmware revision levels and is sent in the following format: THORLABS,MMM,SSS,X.X.X

Where: MMM is the model code

SSS is the serial number

X.X.X is the instrument firmware revision level

***OPC – operation complete - set OPC bit**

***OPC? – operation complete query – places a "1" in output queue**

When *OPC is sent, the OPC bit in the Standard Event Register will set after all pending command operations are complete. When *OPC? is sent, an ASCII "1" is placed in the Output Queue after all pending command operations are complete.

Typically, either one of these commands is sent after the INITiate command. The INITiate command is used to take the instrument out of idle in order to perform measurements. While operating within the trigger model layers, many sent commands will not execute. After all programmed operations are completed, the instrument returns to the idle state at which time all pending commands (including *OPC and/or *OPC?) are executed. After the last pending command is executed, the OPC bit and/or an ASCII "1" is placed in the Output Queue.

When *OPC is sent, the OPC bit in the Standard Event Register will set after all pending command operations are complete. When *OPC? is sent, an ASCII "1" is placed in the Output Queue after all pending command operations are complete.

***RST – reset – return instrument to defaults**

When the *RST command is sent, the instrument performs the following operations:

- Returns the instrument to the default conditions
- Cancels all pending commands.
- Cancels response to any previously received *OPC and *OPC? commands.

***TST? – self-test query – run self test and read result**

Use this query command to perform the instrument self-test routine. The command places the coded result in the Output Queue. A returned value of zero (0) indicates that the test passed, other values indicate that the test failed.

***WAI – wait-to-continue – wait until previous commands are completed**

The *WAI command is a no operation command for the instrument and thus, does not need to be used. It is there for conformance to IEEE488.2.

3.4 PM103x Specific SCPI Command Reference

Operation Status Register		
Bit	**Name**	**PM103**
0	CALibrating	
1	SETTing	
2	RANGing	
3	SWEeping	X
4	MEASuring	
5	Waiting for TRIG	
6	Waiting for ARM	
7	CORRecting	X
8	SENSor connected and operable	X
9	DATA ready	X
10	THACcelerating	X
11	-	
12	-	X
13	INSTrument Summary Bit	
14	PROGram running	
15	always zero	

Operationable Data/Signal Status Register		
Bit	**Name**	
0	Summary of VOL Tage	
1	Summary of CURRent	**PM103**
2	Summary of TIME	
3	Summary of POWer	X
4	Summary of TEMPerature	
5	Summary of FREQuency	
6	Summary of PHASe	
7	Summary of MODulation	
8	Summary of CALibration	x
9	Summary of ENERgy	
10	-	
11	-	
12	-	
13	INSTrument Summary Bit	
14	Command Warning	
15	Not used	

Auxiliary Status Register		
Bit	**Name**	**PM103**
0	NTC connected	X
1	EMM connected	
2	User Power calibration supported	X
3	User Power calibration active	X
4	Ext. power supply connected	
5	Battery charging	
6	Battery low	
7		
8		
9		
10		
11		
12		
13		
14		
15		

Required IEEE488.2 Common Commands	
Command	**Driver**
*CLS	viClear
*ESE	writeRegister
*ESE?	readRegister
*ESR?	readRegister
*IDN?	identificationQuery
*OPC	
*OPC?	
*RST	reset
*SRE	writeRegister
*SRE?	readRegister
*STB?	readRegister
*TST?	selfTest
*WAI	

SYSTem Subsystem. Some are required SCPI Commands.	
SYSTem:ERRor[:NEXT]?	errorQuery, errorMessage
SYSTem:VERSion?	revisionQuery
SYST:SER:TRAN:BAUD?	getDeviceBaudrate
SYST:SER:TRAN:BAUD	setDeviceBaudrate
STATus Subsystem. Some are required SCPI Commands.	
STATus:OPERation[:EVENt]?	readRegister
STATus:OPERation:CONDition?	readRegister
STATus:OPERation:ENABle	writeRegister
STATus:OPERation:ENABle?	readRegister
STATus:OPERation:PTRansition	writeRegister
STATus:OPERation:PTRansition?	readRegister
STATus:OPERation:NTRansition	writeRegister
STATus:OPERation:NTRansition?	readRegister
STATus:QUEStionable[:EVENt]?	readRegister
STATus:QUEStionable:CONDition?	readRegister
STATus:QUEStionable:ENABle	writeRegister
STATus:QUEStionable:ENABle?	readRegister
STATus:QUEStionable:PTRansition	writeRegister
STATus:QUEStionable:PTRansition?	readRegister
STATus:QUEStionable:NTRansition	writeRegister
STATus:QUEStionable:NTRansition?	readRegister
STATus:AUXiliary[:EVENt]?	readRegister
STATus:AUXiliary:CONDition?	readRegister
STATus:AUXiliary:ENABle	writeRegister
STATus:AUXiliary:ENABle?	readRegister
STATus:AUXiliary:PTRansition	writeRegister
STATus:AUXiliary:PTRansition?	readRegister
STATus:AUXiliary:NTRansition	writeRegister
STATus:AUXiliary:NTRansition?	readRegister
STATus:PRESet	presetRegister
CALibration Subsystem	
CALibration:STRing?	getCalibrationMsg

SENSe Subsystem	
SENSe[1]:AVERage[:COUNt]	setAvgTime
	setAvgCnt
SENSe[1]:AVERage[:COUNt]?	getAvgTime
	getAvgCnt
SENSe[1]:CORRection[:LOSS[:INPut[:MAGNitude]]] {MINimum|MAXimum|DEFault|<numeric_value>[dB]}	setAttenuation
SENSe[1]:CORRection[:LOSS[:INPut[:MAGNitude]]]? [{MINimum|MAXimum|DEFault}]	getAttenuation
SENSe[1]:CORRection:COLLect:ZERO[:INITiate]	startDarkAdjust
SENSe[1]:CORRection:COLLect:ZERO:ABORt	cancelDarkAdjust
SENSe[1]:CORRection:COLLect:ZERO:STATe?	getDarkAdjustState
SENSe[1]:CORRection:COLLect:ZERO:MAGNitude?	getDarkOffset
SENSe[1]:CORRection:CSET[1...5]:STATe?	TLPM_getPowerCalibrationPointsState
SENSe[1]:CORRection:CSET[1...5]:STATe true	TLPM_setPowerCalibrationPointsState
SENSe[1]:CORRection:CSET[1...5]:PREamble?	TLPM_getPowerCalibrationPointsInformation
SENSe[1]:CORRection:CSET:PREamble <serial of sensor> <cal date> <author> <sensorPos: 0, 1>	
SENSe[1]:CORRection:CSET[1...5]:POINts?	TLPM_getPowerCalibrationPoints
SENSe[1]:CORRection:CSET:POINts <point list>	TLPM_setPowerCalibrationPoints
SENSe[1]:CORRection:CSET[1...5]:DEFine	
SENSe[1]:Reinit	TLPM_reinitSensor
SENSe[1]:CORRection:BEAMdiameter {MINimum|MAXimum|DEFault|<numeric_value>[m]}	setBeamDia
SENSe[1]:CORRection:BEAMdiameter? [{MINimum|MAXimum|DEFault}]	getBeamDia
SENSe[1]:CORRection:WAVelength {MINimum|MAXimum|<numeric_value>[nm]}	setWavelength
SENSe[1]:CORRection:WAVelength? [{MINimum|MAXimum}]	getWavelength
SENSe[1]:CORRection:POWer[:PDIode][:RESPonse] {MINimum|MAXimum|DEFault|<numeric_value>[A]}	setPhotodiodeResponsivity
SENSe[1]:CORRection:POWer[:PDIode][:RESPonse]? [{MINimum|MAXimum|DEFault}]	getPhotodiodeResponsivity
SENSe[1]:CORRection:ENERgy[:PYRO][:RESPonse] {MINimum|MAXimum|DEFault|<numeric_value>[V]}	setPyrosensorResponsivity
SENSe[1]:CORRection:ENERgy[:PYRO][:RESPonse]? [{MINimum|MAXimum|DEFault}]	getPyrosensorResponsivity

SENSe[1]:CURRent[1][:DC]:RANGe:AUTO {OFF|0|ON*|1}	setCurrentAutoRange
SENSe[1]:CURRent[1][:DC]:RANGe:AUTO?	getCurrentAutoRange
SENSe[1]:CURRent[1][:DC]:RANGe[:UPPer] {MINimum|MAXimum|<numeric_value>[A]}	setCurrentRange
SENSe[1]:CURRent[1][:DC]:RANGe[:UPPer]? [{MINimum|MAXimum}]	getCurrentRange
SENSe[1]:CURRent[1][:DC]:REFerence {MINimum|MAXimum|DEFault|<numeric_value>[A]}	setCurrentRef
SENSe[1]:CURRent[1][:DC]:REFerence? [{MINimum|MAXimum|DEFault}]	getCurrentRef
SENSe[1]:CURRent[1][:DC]:REFerence:STATe {OFF*|0|ON|1}	setCurrentRefState
SENSe[1]:CURRent[1][:DC]:REFerence:STATe?	getCurrentRefState
SENSe[1]:POWer[:DC]:RANGe:AUTO {OFF|0|ON*|1}	setPowerAutoRange
SENSe[1]:POWer[:DC]:RANGe:AUTO?	getPowerAutorange
SENSe[1]:POWer[:DC]:RANGe[:UPPer] {MINimum|MAXimum|<numeric_value>[W]}	setPowerRange
SENSe[1]:POWer[:DC]:RANGe[:UPPer]? [{MINimum|MAXimum}]	getPowerRange
SENSe[1]:POWer[:DC]:REFerence {MINimum|MAXimum|<numeric_value>[W]}	setPowerRef
SENSe[1]:POWer[:DC]:REFerence? [{MINimum|MAXimum}]	getPowerRef
SENSe[1]:POWer[:DC]:REFerence:STATe {OFF*|0|ON|1}	setPowerRefState
SENSe[1]:POWer[:DC]:REFerence:STATe?	getPowerRefState
SENSe[1]:ENERgy:RANGe[:UPPer] {MINimum|MAXimum|<numeric_value>[J]}	setEnergyRange
SENSe[1]:ENERgy:RANGe[:UPPer]? [{MINimum|MAXimum}]	getEnergyRange
SENSe[1]:ENERgy:REFerence {MINimum|MAXimum|DEFault|<numeric_value>[J]}	setEnergyRef
SENSe[1]:ENERgy:REFerence? [{MINimum|MAXimum|DEFault}]	getEnergyRef
SENSe[1]:ENERgy:REFerence:STATe {OFF*|0|ON|1}	setEnergyRefState
SENSe[1]:ENERgy:REFerence:STATe?	getEnergyRefState
SENSe[1][:PDIode]:FREQuency:MODE {CW|PULSE}	setFreqMode
SENSe[1][:PDIode]:FREQuency:MODE?	getFreqMode
SENSe[1]:ILLUminance:UNIT	setIlluminanceUnit
SENSe[1]:ILLUminance:UNIT?	getIlluminanceUnit
SENSe[1]:PEAKdetector:SEARch	startPeakDetection

SENSe[1]:PEAKdetector:SEARch?	isPeakDetectorRunning
SENSe[1]:POWer[:DC]:UNIT {DBM|W}	setPowerUnit
SENSe[1]:POWer[:DC]:UNIT?	getPowerUnit
SENSe[1]:VOLTage[:DC]:RANGe:AUTO {OFF|0|ON*|1}	setVoltageAutoRange
SENSe[1]:VOLTage[:DC]:RANGe:AUTO?	getVoltageAutorange
SENSe[1]:VOLTage[:DC]:RANGe[:UPPer] {MINimum|MAXimum|<numeric_value>[V]}	setVoltageRange
SENSe[1]:VOLTage[:DC]:RANGe[:UPPer]? [{MINimum|MAXimum}]	getVoltageRange
SENSe[1]:VOLTage[:DC]:REFerence {MINimum|MAXimum|DEFault|<numeric_value>[V]}	setVoltageRef
SENSe[1]:VOLTage[:DC]:REFerence? [{MINimum|MAXimum|DEFault}]	getVoltageRef
SENSe[1]:VOLTage[:DC]:REFerence:STATe {OFF*|0|ON|1}	setVoltageRefState
SENSe[1]:VOLTage[:DC]:REFerence:STATe?	getVoltageRefState
SENSe5:CORRection:COEFficient:RESistance {MINimum|MAXimum|DEFault|<numeric_value>[Ohm]}	setExtNtcParameter
SENSe5:CORRection:COEFficient:RESistance? [{MINimum|MAXimum|DEFault}]	getExtNtcParameter
SENSe5:CORRection:COEFficient:BETA {MINimum|MAXimum|DEFault|<numeric_value>[K]}	setExtNtcParameter
SENSe5:CORRection:COEFficient:BETA? [{MINimum|MAXimum|DEFault}]	getExtNtcParameter
SENSe5: RESistance:DATA?	measExtNtcResistance
SENSe5: TEMPerature:DATA?	measExtNtcTemperature
SENSe#:CONDition?	getMeasPinMode
SENSe#:POWer[:DC]:CONDition:LEVel? Res:<float min [W]>, <float max [W]>	getMeasPinPowerLevel
SENSe#:POWer[:DC]:CONDition:LEVel <float min [W]>, <float max [W]>	setMeasPinPowerLevel
SENSe#:ENERgy[:DC]:CONDition:LEVel? Res:<float min [J]>, <float max [J]>	getMeasPinEnergyLevel
SENSe#:ENERgy[:DC]:CONDition:LEVel <float min [J]>, <float max [J]>	setMeasPinEnergyLevel

Source Subsystem	
SOURce:DIGital:DATA <numeric_value>	setDigIoOutput
SOURce:DIGital:DATA?	getDigIoOutput
SOURce:DIGital:ENABle <numeric_value>	setDigIoDirection
SOURce:DIGital:ENABle?	getDigIoDirection
SOURce:DIGital:MODE?	getDigIoMode
SOURce:DIGital:MODE <IO,Alt>	setDigIoMode
SOURce:DIGital:PIN#:FUNC?	getDigIoPinMode
SOURce:DIGital:PIN#:FUNC <INP,OUT,IALT,OALT>	setDigIoPinMode
SOUR:DIG:OUTP:DATA?	getDigIoPinOutput
SOUR:DIG:OUTP:DATA <numeric_value>	setDigIoPinOutput
SOUR:DIG:OUTP:DATA?	getDigIoPinInput
SOURce2:VOLTage[:LEVel][:IMMediate][:AMPLitude]? {MIN | MAX}	getAnalogOutputVoltage
SOURce2:VOLTage:CORRection:SLOPe[:OUTPut][:MAGNitude] {<numeric value> [V/W] MIN | MAX |DEF}	setAnalogOutputSlope
SOURce2:VOLTage:CORRection:SLOPe[:OUTPut][:MAGNitude]? {MIN | MAX | DEF}	getAnalogOutputSlope
INPut Subsystem	
INPut[:PDIode]:FILTer[:LPASs][:STATe] {OFF*|0|ON|1}	setInputFilterState
INPut[:PDIode]:FILTer[:LPASs][:STATe]?	getInputFilterState
INPut:ADAPter[:TYPE] {PHOTodiode|THERmal}	setInputAdapterType
INPut:ADAPter[:TYPE]?	getInputAdapterType
Measurement Commands	
INITiate[:IMMediate]	
INITiate:CONTinuous	
ABORt	
CONFigure[:SCALar][:POWer]	cfgPower
CONFigure[:SCALar]:CURRent[:DC]	cfgCurrent
CONFigure[:SCALar]:PDENsity	cfgPowerDensity
CONFigure[:SCALar]:RESistance	cfgResistance
CONFigure[:SCALar]:TEMPerature	cfgTemperature
CONFigure[:SCALar]:ILLUminance	cfgIgIlluminance
CONFigure[:SCALar]:NWIDth	cfgNegativePulseWidth
CONFigure[:SCALar]:PWIDth	cfgPositivePulseWidth
CONFigure[:SCALar]:PDUTycycle	cfgNegativeDutyCycle
CONFigure[:SCALar]:NPDUTycycle	cfgPositiveDutyCycle
MEASure[:SCALar][:POWer]?	measPower

MEASure[:SCALar]:CURRent[:DC]?	measCurrent
MEASure[:SCALar]:VOLTage[:DC]?	measVoltage
MEASure[:SCALar]:FREQuency?	measFreq
MEASure[:SCALar]:PDENsity?	measPowerDens
MEASure[:SCALar]:RESistance?	measResistance
MEASure[:SCALar]:TEMPerature?	measSensTemperature
MEASure[:SCALar]:ILLUminance?	measIlluminance
MEASure[:SCALar]:NWIDth?	measNegPulseWidth
MEASure[:SCALar]:PWIDth?	measPosPulseWidth
MEASure[:SCALar]:PDUTycycle?	measNegDutyCycle
MEASure[:SCALar]:NDUTycycle?	measPosDutyCycle
FETCh?	
READ?	
CONFigure?	
Fast Continuous Array Measurements	
ACQuire[:SCALar]:FAST:RESET	resetArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay][:POWer]	getPowerArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay]:CURRent[:DC]	getCurrentArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay]:VOLTage[:DC]	getVoltageArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay]:PDENsity	getPDensityArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay]:ENERgy	getEnergyArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay]:EDENsity	getEDensityArrayMeasurement
ACQuire[:SCALar]:FAST[:ARRay]:FETCh?	getNextArrayMeasurement
Trigger Array Measurements	
MEASure[:SCALar]:ARRay[:SWTrig] [:POWer]? {<numeric_value>[samples] , <numeric_value>[us]}	getPowerMeasurementSequence
MEASure[:SCALar]:ARRay:HWTrig[:POWer]? {<numeric_value>[samples] , <numeric_value>[us]}	getPowerMeasurementSequenceHWTrigger
MEASure[:SCALar]:ARRay[:SWTrig]:CURRent? {<numeric_value>[samples] , <numeric_value>[us]}	getCurrentMeasurementSequence
MEASure[:SCALar]:ARRay:HWTrig:CURRent? {<numeric_value>[samples] , <numeric_value>[us]}	getCurrentMeasurementSequenceHWTrigger
READ[:SCALar]:ARRay:[SWTrig]?	
READ[:SCALar]:ARRay:HWTrig?	
CONFigure[:SCALar]:ARRay[:SWTrig][:POWer] {<numeric_value>[samples] , <numeric_value>[us]}	
CONFigure[:SCALar]:ARRay:HWTrig[:POWer] {<numeric_value>[samples] , <numeric_value>[us]}	
CONFigure[:SCALar]:ARRay[:SWTrig]:CURRent {<numeric_value>[samples] , <numeric_value>[us]}	
CONFigure[:SCALar]:ARRay:HWTrig:CURRent {<numeric_value>[samples] , <numeric_value>[us]}	

INIT[:SCALar]:ARRay:[SWTrig]	
INIT[:SCALar]:ARRay:HWTrig	
FETCh[:SCALar]:ARRay? <uint32: index>,<uint32: count>	

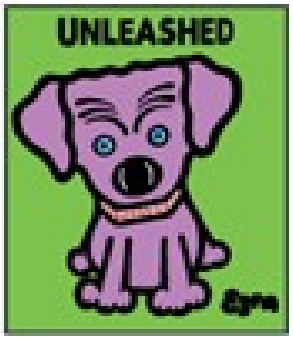
Serial Interface RS232 (Default)

To operate the PM103 via RS232, the RxD, TxD and ground need to be wired from the 15 Pin D-Sub connector to a 9 Pin female connector to perform the PC connection:

PM101 DA-15	DE-9 Male	PC (DE-9 Female)
Pin 7 – TxD	Pin 2	RxD
Pin 8 – RxD	Pin 3	TxD
Pin 15 – Ground	Pin 5	Ground

Set the port setup as following:

Parameter	Setting
Baud Rate	115 200 Bd (Default) Supported Range of Baud Rate: 9600 to 921 600 Bd
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None
Termination Character	Line Feed (x0A; \n) The termination character needs to be enabled.



THORLABS

www.thorlabs.com