



# PM101x

## Write Your Own Application

- **Drivers**
- **PC Interface**
- **Programmer's Reference**

**Version: 1.0**  
**Date: 12-Aug-2019**

---

# 1 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 PM101x, all information can be found in the documentation for the OPM software, the Thorlabs software to steer PM101x. The OPM and respective information can be downloaded from the [OPM website](#).

## 2 Computer Interface

For communication between the PM101x and the PC with custom made software, the following interfaces can be used:

All PM101x optical power meters have a USB 2.0 interface that allows to send commands from a host computer to the instrument and vice versa. The connection between PC and PM101x is accomplished by a USB cable with a male type 'A' connector at the PC side and a type 'Mini-B' connector on the instrument side.

### PM101:

The PM101 further has a DA15 port with a UART interface that allows to send commands from a host computer to the instrument and vice versa.

### PM101R:

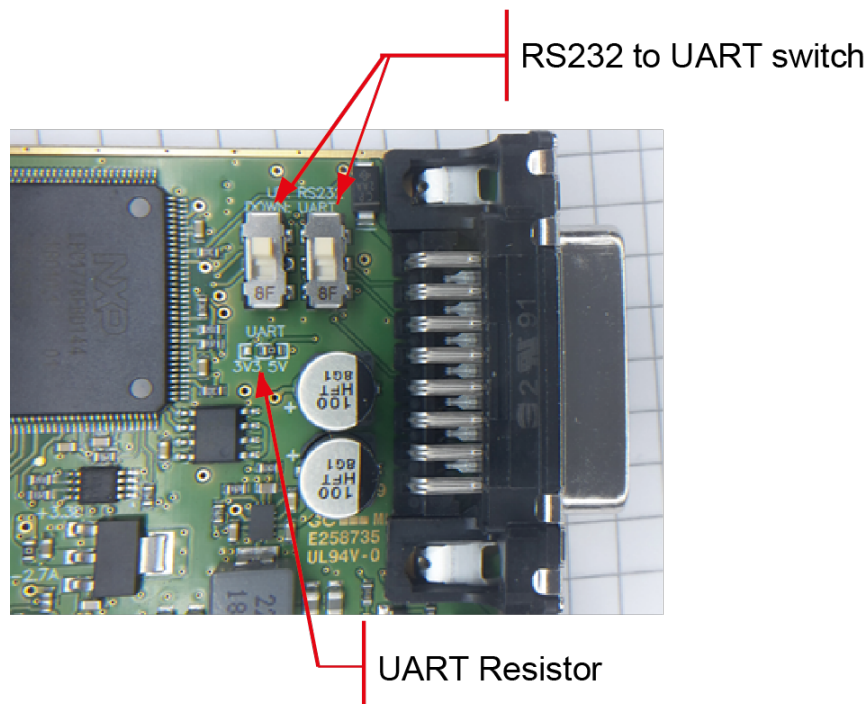
The PM101R further has a serial DE9 interface that allows to send commands from a host computer to the instrument and vice versa.

### 2.1 Serial Interface with UART

To use the PM101 with UART, the unit must be switched from RS232 operation with  $\pm 5$  V to UART operation.

To do so, the PM101 must be opened to perform the following operations:

1. Remove the bolts from the 15pin sub-d connector.
2. Remove the 4 TX9 screws in the sensor front panel.
3. Pull out the PCB.
4. Move both switches from PC RS232 Operation (default - both switches in upper position) to  $\pm 5$  V level UART (both switches in lower position)



#### Note

For instruction on how to switch to operation with 3.3 V, please contact [Thorlabs](https://www.thorlabs.com)<sup>[17]</sup>.

## 3 Programmers Reference PM101x

### 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 manual 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 PM101x Specific SCPI Command Reference

Operation Status Register			Operationable Data/Signal Status Register			Auxiliary Status Register		
Bit	Name	PM 101	Bit	Name	PM 101	Bit	Name	PM 101
0	CALibrating		0	Summary of VOLTage		0	NTC connected	X
1	SETTLing		1	Summary of CURRent		1	EMM connected	
2	RANGing	X	2	Summary of TIME		2	User Power calibration supported	X
3	SWEEping		3	Summary of POWer	X	3	User Power calibration active	X
4	MEASuring	X	4	Summary of TEMPerature		4	Ext. power supply connected	
5	Waiting for TRIG		5	Summary of FREQuency		5	Battery charging	
6	Waiting for ARM		6	Summary of PHASe		6	Battery low	
7	CORREcting	X	7	Summary of MODulation		7	-	
8	SENSor connected and operable	X	8	Summary of CALibration	X	8	-	
9	DATA ready	X	9	Summary of ENERgy		9	-	
10	THACcelerating	X	10	-		10	-	
11	-		11	-		11	-	
12	-		12	-		12	-	
13	INSTrument Summary Bit	X	13	INSTrument Summary Bit		13	-	
14	PROGram running		14	Command Warning		14	-	
15	always zero		15	Not used		15	-	

Command	Driver
<b>Required IEEE488.2 Common Commands</b>	
*CLS	viClear
*ESE	PM100D_writeRegister
*ESE?	PM100D_readRegister
*ESR?	PM100D_readRegister
*IDN?	PM100D_identificationQuery
*OPC	
*OPC?	
*RST	PM100D_reset
*SRE	PM100D_writeRegister
*SRE?	PM100D_readRegister
*STB?	PM100D_readRegister
*TST?	PM100D_selfTest
*WAI	
<b>SYSTEM subsystem. Some are required SCPI commands</b>	

SYSTem:ERRor[:NEXT]?	PM100D_errorQuery, PM100D_errorMessage
SYSTem:VERSion?	PM100D_revisionQuery
SYST:SER:TRAN:BAUD?	TLPM_getDeviceBaudrate
SYST:SER:TRAN:BAUD	TLPM_setDeviceBaudrate
<b>STATus subsystem. Some are required SCPI commands</b>	
STATus:OPERation[:EVENT]?	PM100D_readRegister
STATus:OPERation:CONDition?	PM100D_readRegister
STATus:OPERation:ENABLE	PM100D_writeRegister
STATus:OPERation:ENABLE?	PM100D_readRegister
STATus:OPERation:PTRansition	PM100D_writeRegister
STATus:OPERation:PTRansition?	PM100D_readRegister
STATus:OPERation:NTRansition	PM100D_writeRegister
STATus:OPERation:NTRansition?	PM100D_readRegister
STATus:QUEStionable[:EVENT]?	PM100D_readRegister
STATus:QUEStionable:CONDition?	PM100D_readRegister
STATus:QUEStionable:ENABLE	PM100D_writeRegister
STATus:QUEStionable:ENABLE?	PM100D_readRegister
STATus:QUEStionable:PTRansition	PM100D_writeRegister
STATus:QUEStionable:PTRansition?	PM100D_readRegister
STATus:QUEStionable:NTRansition	PM100D_writeRegister
STATus:QUEStionable:NTRansition?	PM100D_readRegister
STATus:AUXiliary[:EVENT]?	PM100D_readRegister
STATus:AUXiliary:CONDition?	PM100D_readRegister
STATus:AUXiliary:ENABLE	PM100D_writeRegister
STATus:AUXiliary:ENABLE?	PM100D_readRegister

STATus:AUXiliary:PTRansition	PM100D_writeRegister
STATus:AUXiliary:PTRansition?	PM100D_readRegister
STATus:AUXiliary:NTRansition	PM100D_writeRegister
STATus:AUXiliary:NTRansition?	PM100D_readRegister
STATus:PRESet	PM100D_presetRegister
<b>CALibration subsystem</b>	
CALibration:STRing?	PM100D_getCalibrationMsg
<b>SENSe subsystem</b>	
SENSe[1]:AVERage[:COUNT]	PM100D_setAvgTime
	PM100D_setAvgCnt
SENSe[1]:AVERage[:COUNT]?	PM100D_getAvgTime
	PM100D_getAvgCnt
SENSe[1]:CORRection[:LOSS[:INPut[:MAGNitude]]] {MINimum   MAXimum   DEFault   <numeric_value>[dB]}	PM100D_setAttenuation
SENSe[1]:CORRection[:LOSS[:INPut[:MAGNitude]]]? [{MINimum   MAXimum   DEFault}]	PM100D_getAttenuation
SENSe[1]:CORRection:COLLect:ZERO[:INITiate]	PM100D_startDarkAdjust
SENSe[1]:CORRection:COLLect:ZERO:ABORT	PM100D_cancelDarkAdjust
SENSe[1]:CORRection:COLLect:ZERO:STATE?	PM100D_getDarkAdjustState
SENSe[1]:CORRection:COLLect:ZERO:MAGNitude?	PM100D_getDarkOffset
SENSe[1]:CORRection:CSET[1...5]:STATE?	TLPM_getPowerCalibrationPoint- sState
SENSe[1]:CORRection:CSET[1...5]:STATE true	TLPM_setPowerCalibrationPoint- sState
SENSe[1]:CORRection:CSET[1...5]:PREamble?	TLPM_getPowerCalibrationPoint- sInformation
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]}	PM100D_setBeamDia
SENSe[1]:CORRection:BEAMdiameter? [{MINimum   MAXimum   DEFault}]	PM100D_getBeamDia
SENSe[1]:CORRection:WAVelength {MINimum   MAXimum   <numeric_value>[m]}	PM100D_setWavelength
SENSe[1]:CORRection:WAVelength? [{MINimum   MAXimum}]	PM100D_getWavelength
SENSe[1]:CORRection:POWER[:PDIode][:RESPonse] {MINimum   MAXimum   DEFault   <numeric_value>[A]}	PM100D_setPhotodiodeResponsivity
SENSe[1]:CORRection:POWER[:PDIode][:RESPonse]? [{MINimum   MAXimum   DEFault}]	PM100D_getPhotodiodeResponsivity
SENSe[1]:CORRection:POWER:THERmopile[:RESPonse] {MINimum   MAXimum   DEFault   <numeric_value>[V]}	PM100D_setThermopileResponsivity
SENSe[1]:CORRection:POWER:THERmopile[:RESPonse]? [{MINimum   MAXimum   DEFault}]	PM100D_getThermopileResponsivity
SENSe[1]:CURRent[1][:DC]:RANGe:AUTO {OFF   0   ON*   1}	PM100D_setCurrentAutoRange
SENSe[1]:CURRent[1][:DC]:RANGe:AUTO?	PM100D_getCurrentAutoRange
SENSe[1]:CURRent[1][:DC]:RANGe[:UPPer] {MINimum   MAXimum   <numeric_value>[A]}	PM100D_setCurrentRange
SENSe[1]:CURRent[1][:DC]:RANGe[:UPPer]? [{MINimum   MAXimum}]	PM100D_getCurrentRange
SENSe[1]:CURRent[1][:DC]:REFerence {MINimum   MAXimum   DEFault   <numeric_value>[A]}	PM100D_setCurrentRef
SENSe[1]:CURRent[1][:DC]:REFerence? [{MINimum   MAXimum   DEFault}]	PM100D_getCurrentRef
SENSe[1]:CURRent[1][:DC]:REFerence:STATe {OFF*   0   ON   1}	PM100D_setCurrentRefState
SENSe[1]:CURRent[1][:DC]:REFerence:STATe?	PM100D_getCurrentRefState
SENSe[1]:POWER[:DC]:RANGe:AUTO {OFF   0   ON*   1}	PM100D_setPowerAutoRange

SENSe[1]:POWer[:DC]:RANGe:AUTO?	PM100D_getPowerAutorange
SENSe[1]:POWer[:DC]:RANGe[:UPPer] {MINimum MAXimum <numeric_value>[W]}	PM100D_setPowerRange
SENSe[1]:POWer[:DC]:RANGe[:UPPer]? [{MINimum MAXimum}]	PM100D_getPowerRange
SENSe[1]:POWer[:DC]:REFerence {MINimum MAXimum <numeric_value>[W]}	PM100D_setPowerRef
SENSe[1]:POWer[:DC]:REFerence? [{MINimum MAXimum}]	PM100D_getPowerRef
SENSe[1]:POWer[:DC]:REFerence:STATe {OFF* 0 ON 1}	PM100D_setPowerRefState
SENSe[1]:POWer[:DC]:REFerence:STATe?	PM100D_getPowerRefState
SENSe[1]:POWer[:DC]:UNIT {DBM W}	PM100D_setPowerUnit
SENSe[1]:POWer[:DC]:UNIT?	PM100D_getPowerUnit
SENSe[1]:VOLTage[:DC]:RANGe:AUTO {OFF 0 ON* 1}	PM100D_setVoltageAutoRange
SENSe[1]:VOLTage[:DC]:RANGe:AUTO?	PM100D_getVoltageAutorange
SENSe[1]:VOLTage[:DC]:RANGe[:UPPer] {MINimum MAXimum <numeric_value>[V]}	PM100D_setVoltageRange
SENSe[1]:VOLTage[:DC]:RANGe[:UPPer]? [{MINimum MAXimum}]	PM100D_getVoltageRange
SENSe[1]:VOLTage[:DC]:REFerence {MINimum MAXimum DEFAULT <numeric_value>[V]}	PM100D_setVoltageRef
SENSe[1]:VOLTage[:DC]:REFerence? [{MINimum MAXimum DEFAULT}]	PM100D_getVoltageRef
SENSe[1]:VOLTage[:DC]:REFerence:STATe {OFF* 0 ON 1}	PM100D_setVoltageRefState
SENSe[1]:VOLTage[:DC]:REFerence:STATe?	PM100D_getVoltageRefState
SENSe5:CORRection:COEFFicient:RESistance {MINimum MAXimum DEFAULT <numeric_value>[Ohm]}	PM100D_setExtNtcParameter
SENSe5:CORRection:COEFFicient:RESistance? [{MINimum MAXimum DEFAULT}]	PM100D_getExtNtcParameter
SENSe5:CORRection:COEFFicient:BETA {MINimum MAXimum DEFAULT <numeric_value>[K]}	PM100D_setExtNtcParameter

SENSe5:CORRection:COEFFicient:BETA? [{MINimum MAXimum DEFault}]	PM100D_getExtNtcParameter
SENSe5: RESistance:DATA?	PM100D_measExtNtcResistance
SENSe5: TEMPerature:DATA?	PM100D_measExtNtcTemperature
<b>Source subsystem</b>	
SOURce:DiGital:DATA <numeric_value>	PM100D_setDigIoOutput
SOURce:DiGital:DATA?	PM100D_getDigIoOutput
SOURce:DiGital:ENABle <numeric_value>	PM100D_setDigIoDirection
SOURce:DiGital:ENABle?	PM100D_getDigIoDirection
SOURce2:VOLTage[:LEVel][:IMMediate][:AMPLitude] {<numeric value>   MIN   MAX}	PM100D_setAnalogOutput
SOURce2:VOLTage[:LEVel][:IMMediate][:AMPLitude]? {<numeric value>   MIN   MAX}	PM100D_getAnalogOutput
SOURce2:VOLTage:CORRection:SLOPe[:OUTPut][:MAGNitude] {<numeric value>   MIN   MAX}	PM100D_setAnalogOutputSlope
SOURce2:VOLTage:CORRection:SLOPe[:OUTPut][:MAGNitude]? [{MIN   MAX}]	PM100D_getAnalogOutputSlope
<b>INPut subsystem</b>	
INPut[:PDiode]:FILTer[:LPASS][:STATe] {OFF* 0 ON 1}	PM100D_setInputFilterState
INPut[:PDiode]:FILTer[:LPASS][:STATe]?	PM100D_getInputFilterState
INPut:THERmopile:ACCElerator[:STATe] {OFF* 0 ON 1}	PM100D_setAccelState
INPut:THERmopile:ACCElerator[:STATe]?	PM100D_getAccelState
INPut:THERmopile:ACCElerator:TAU {MINimum MAXimum DEFault <numeric_value>[s]}	PM100D_setAccelTau
INPut:THERmopile:ACCElerator:TAU? [{MINimum MAXimum DEFault}]	PM100D_getAccelTau
INPut:THERmopile:ACCElerator:AUTO {OFF 0 ON* 1}	PM100D_setAccelMode
INPut:THERmopile:ACCElerator:AUTO?	PM100D_getAccelMode
INPut:ADAPter[:TYPE] {PHOTodiode THERmal}	PM100D_setInputAdapterType
INPut:ADAPter[:TYPE]?	PM100D_getInputAdapterType

Measurement commands	
INITiate[:IMMEDIATE]	
INITiate:CONTinuous	
ABORT	
CONFigure[:SCALar][:POWer]	
CONFigure[:SCALar]:CURRent[:DC]	
CONFigure[:SCALar]:VOLTage[:DC]	
CONFigure[:SCALar]:PDENsity	
CONFigure[:SCALar]:RESistance	
CONFigure[:SCALar]:TEMPerature	
MEASure[:SCALar][:POWer]?	PM100D_measPower
MEASure[:SCALar]:CURRent[:DC]?	PM100D_measCurrent
MEASure[:SCALar]:VOLTage[:DC]?	PM100D_measVoltage
MEASure[:SCALar]:FREQUency?	PM100D_measFreq
MEASure[:SCALar]:PDENsity?	PM100D_measPowerDens
MEASure[:SCALar]:RESistance?	
MEASure[:SCALar]:TEMPerature?	PM100D_measSensTemperature
FETCH?	
READ?	
CONFigure?	



**Serial Interface RS232 (Default)**

To operate the PM101 via RS232, the RxD, TxD and ground need to be wired from the 15pin 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 Bit/s (default) Can be configured between 9.600 and 256.000 Bit/s
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None
Termination Character	LF (x0A; \n) – the termination character needs to be enabled

## 3.5 Thorlabs Worldwide Contact

### USA, Canada, and South America

Thorlabs, Inc.  
56 Sparta Avenue  
Newton, NJ 07860  
USA  
Tel: 973-300-3000  
Fax: 973-300-3600  
[www.thorlabs.com](http://www.thorlabs.com)  
[www.thorlabs.us](http://www.thorlabs.us) (West Coast)  
Email: [sales@thorlabs.com](mailto:sales@thorlabs.com)  
Support: [techsupport@thorlabs.com](mailto:techsupport@thorlabs.com)

### UK and Ireland

Thorlabs Ltd.  
1 Saint Thomas Place, Ely  
Cambridgeshire CB7 4EX  
United Kingdom  
Tel: +44-1353-654440  
Fax: +44-1353-654444  
[www.thorlabs.com](http://www.thorlabs.com)  
Email: [sales.uk@thorlabs.com](mailto:sales.uk@thorlabs.com)  
Support: [techsupport.uk@thorlabs.com](mailto:techsupport.uk@thorlabs.com)

### Europe

Thorlabs GmbH  
Hans-Böckler-Str. 6  
85221 Dachau  
Germany  
Tel: +49-8131-5956-0  
Fax: +49-8131-5956-99  
[www.thorlabs.de](http://www.thorlabs.de)  
Email: [europe@thorlabs.com](mailto:europe@thorlabs.com)

### Scandinavia

Thorlabs Sweden AB  
Bergfotsgatan 7  
431 35 Mölndal  
Sweden  
Tel: +46-31-733-30-00  
Fax: +46-31-703-40-45  
[www.thorlabs.com](http://www.thorlabs.com)  
Email: [scandinavia@thorlabs.com](mailto:scandinavia@thorlabs.com)

### France

Thorlabs SAS  
109, rue des Côtes  
78600 Maisons-Laffitte  
France  
Tel: +33-970 444 844  
Fax: +33-825 744 800  
[www.thorlabs.com](http://www.thorlabs.com)  
Email: [sales.fr@thorlabs.com](mailto:sales.fr@thorlabs.com)

### Brazil

Thorlabs Vendas de Fotônicos Ltda.  
Rua Riachuelo, 171  
São Carlos, SP 13560-110  
Brazil  
Tel: +55-16-3413 7062  
Fax: +55-16-3413 7064  
[www.thorlabs.com](http://www.thorlabs.com)  
Email: [brasil@thorlabs.com](mailto:brasil@thorlabs.com)

### Japan

Thorlabs Japan, Inc.  
3-6-3 Kitamachi  
Nerima-ku, Tokyo 179-0081  
Japan  
Tel: +81-3-6915-7701  
Fax: +81-3-6915-7716  
[www.thorlabs.co.jp](http://www.thorlabs.co.jp)  
Email: [sales@thorlabs.jp](mailto:sales@thorlabs.jp)

### China

Thorlabs China  
Room A101, No. 100  
Lane 2891, South Qilianshan Road  
Putuo District  
Shanghai 200331  
China  
Tel: +86-21-60561122  
Fax: +86-21-32513480  
[www.thorlabs.com](http://www.thorlabs.com)  
Email: [chinasales@thorlabs.com](mailto:chinasales@thorlabs.com)



**THORLABS**  
[www.thorlabs.com](http://www.thorlabs.com)

---