THORLABS

PM200
PM100A/D/USB
PM160

Write Your Own Application

- Drivers
- Samples
- SCPI Command Reference

Contents

1 Write Your Own Application

In order to write your own application, you need a specific instrument driver and some tools for use in different programming environments. The driver and tools are included in the installer package and cannot be found as separate files on the installation CD.

In this section the location of drivers and files, required for programming in different environments, are given for installation under Windows XP (32 bit) and Windows 7 (32 and 64 bit)

Note

PM200 software and drivers contains 32 bit and 64 bit applications.

In 32 bit systems, only the 32 bit components are installed to

```
C:\Program Files\...
```

In 64 bit systems the 64 bit components are being installed to

```
C:\Program Files\...
```

while 32 bit components can be found at

```
C:\Program Files (x86)\...
```

In the table below you will find a summary of what files you need for particular programming environments.

Programming environment	Necessary files
C, C++, CVI	*.fp (function panel file; CVI IDE only) *.h (header file) *.lib (static library) *.dll (dynamic linked library)
C#	.net wrapper dll
Visual Studio	*.h (header file) *.lib (static library) or .net wrapper dll
LabView	*.fp (function panel) and NI VISA instrument driver Beside that, LabVIEW driver vi's are provided with the *.llb container file

Note

All above environments require also the NI VISA instrument driver dll !

During NI-VISA Runtime installation, a system environment variable `VXIPNPPATH` for including files is created. It contains the information where the drivers are installed to, usually to `C:\Program Files\IVI Foundation\VISA\WinNT\`.

This is the reason, why after installation of a NI-VISA Runtime a system reboot is required: This environment variable is necessary for installation of the instrument driver software components.

In the next sections the location of above files is described in detail.

1.1 Windows XP 32bit

NI VISA Instrument Driver 32bit

```
C:\Program Files\IVI Foundation\VISA\WinNT\Bin\PM100D_32.dll
```

Online Help for NI VISA Instrument Driver

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\Manual\PM100D.html
```

NI LabVIEW Driver

The LabVIEW version of this driver was generated with the "NI LabVIEW Instrument Driver Import Wizard 2.0" in conjunction with LabVIEW 2011.

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\LabVIEW\PM100D.llb
```

Header File

```
C:\Program Files\IVI Foundation\VISA\WinNT\include\PM100D.h
```

Static Library

```
C:\Program Files\IVI Foundation\VISA\WinNT\lib\msc\PM100D_32.lib
```

Function Panel

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\PM100D.fp
```

.net wrapper dll

```
C:\Program Files\IVI Foundation\VISA\VisaCom\...
...Primary Interop Assemblies\Thorlabs.PM100D.dll
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\DotNet\...
...Thorlabs.PM100D.dll
C:\Program Files\Microsoft.NET\Primary Interop Assemblies\...
...Thorlabs.PM100D.dll
```

Examples

ANSI-C

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\Samples\C\sample.c
```

Thorlabs PM100x/PM160/PM200 Instrument Driver Sample Application with console interface. Read the comment text in the sample.c file for more information.

C# Visual Studio 2010

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\Samples\...
...DotNet\DotNetSample.csproj
```

Very basic C# sample project. The application opens a session to a device takes one measurement and closes the device again.

1.2 Windows Vista / 7 / 8 - 32bit

NI VISA Instrument Driver

```
C:\Program Files\IVI Foundation\VISA\WinNT\Bin\PM100D_32.dll
```

Online Help for NI VISA Instrument Driver:

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\Manual\PM100D.html
```

NI LabVIEW Driver

The LabVIEW version of this driver was generated with the "NI LabVIEW Instrument Driver Import Wizard 2.0" in conjunction with LabVIEW 2011.

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\LabVIEW\PM100D.llb
```

Header File

```
C:\Program Files\IVI Foundation\VISA\WinNT\include\PM100D.h
```

Static Library

```
C:\Program Files\IVI Foundation\VISA\WinNT\lib\msc\PM100D_32.lib
```

Function Panel

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\PM100D.fp
```

.net wrapper dll

```
C:\Program Files\IVI Foundation\VISA\VisaCom\...
...Primary Interop Assemblies\Thorlabs.PM100D.dll
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\DotNet\...
...Thorlabs.PM100D.dll
C:\Program Files\Microsoft.NET\Primary Interop Assemblies\...
...Thorlabs.PM100D.dll
```

Examples:

ANSI-C

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\Samples\C\sample.c
```

Thorlabs PM100x/PM160/PM200 Driver Sample Application with console interface. Read the comment text in the sample.c file for more information.

C# Visual Studio 2010

```
C:\Program Files\IVI Foundation\VISA\WinNT\PM100D\Samples\...
...DotNet\DotNetSample.csproj
```

Very basic C# sample project. The application opens a session to a device takes one measurement and closes the device again.

1.3 Windows Vista / 7 / 8 - 64bit

NI VISA Instrument Driver 32bit

```
C:\Program Files (x86)\IVI Foundation\VISA\WinNT\Bin\PM100D_32.dll
```

NI VISA Instrument Driver 64bit

```
C:\Program Files\IVI Foundation\VISA\Win64\Bin\PM100D_64.dll
```

Online Help for NI VISA Instrument Driver

```
C:\Program Files\IVI Foundation\VISA\Win64\PM100D\Manual\PM100D.html
```

NI LabVIEW Driver 32bit

```
C:\Program Files (x86)\IVI Foundation\VISA\WinNT\PM100D\...
...LabVIEW\PM100D.llb
```

NI LabVIEW Driver 64bit

```
C:\Program Files\IVI Foundation\VISA\Win64\PM100D\LabVIEW\PM100D.llb
```

Header File 32bit

```
C:\Program Files (x86)\IVI Foundation\VISA\WinNT\include\PM100D.h
```

Header File 64bit

```
C:\Program Files\IVI Foundation\VISA\Win64\include\PM100D.h
```

Static Library 32bit

```
C:\Program Files (x86)\IVI Foundation\VISA\WinNT\lib\msc\PM100D_32.lib
```

Static Library 64bit

```
C:\Program Files\IVI Foundation\VISA\Win64\Lib_x64\msc\PM100D_64.lib
```

Function Panel 32bit

```
C:\Program Files (x86)\IVI Foundation\VISA\WinNT\PM100D\PM100D.fp
```

Function Panel 64bit

```
C:\Program Files\IVI Foundation\VISA\Win64\PM100D\PM100D.fp
```

.net wrapper dll 32bit

```
C:\Program Files (x86)\IVI Foundation\VISA\VisaCom\...
...Primary Interop Assemblies\Thorlabs.PM100D.dll
```

```
C:\Program Files (x86)\Microsoft.NET\Primary Interop Assemblies\...
...Thorlabs.PM100D.dll
```

.net wrapper dll 64bit

```
C:\Program Files\IVI Foundation\VISA\VisaCom64\...
...Primary Interop Assemblies\Thorlabs.PM100D.dll
```

Examples:

ANSI-C

```
C:\Program Files\IVI Foundation\VISA\Win64\PM100D\Samples\C\sample.c
```

Thorlabs PM100x/PM160/PM200 Driver Sample Application with console interface. Read the comment text in the sample.c file for more information.

C# Visual Studio 2010

```
C:\Program Files\IVI Foundation\VISA\Win64\PM100D\Samples\DotNet\...
...DotNetSample_64.csproj
```

Very basic C# sample project. The application opens a session to a device takes one measurement and closes the device again.

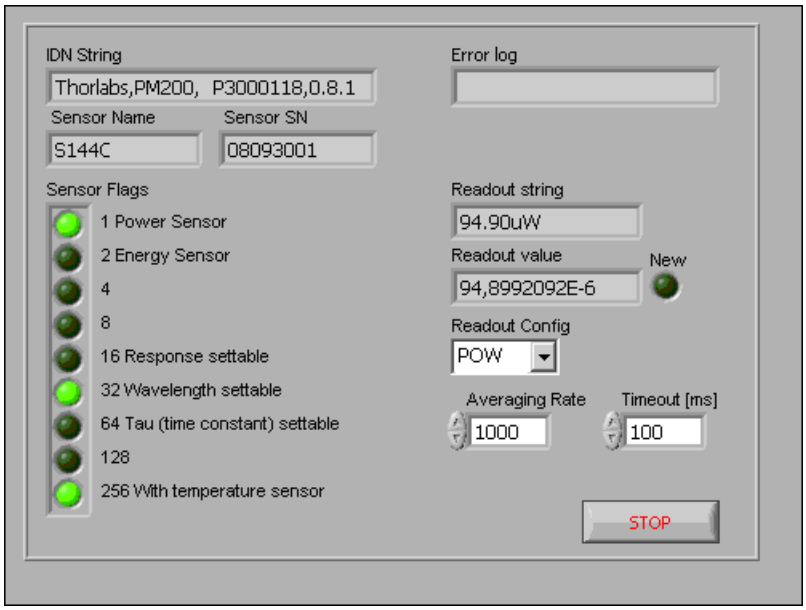
1.4 Simple LabVIEW Example using SCPI commands

The PM200 Instrument driver is not required for this LabVIEW example.

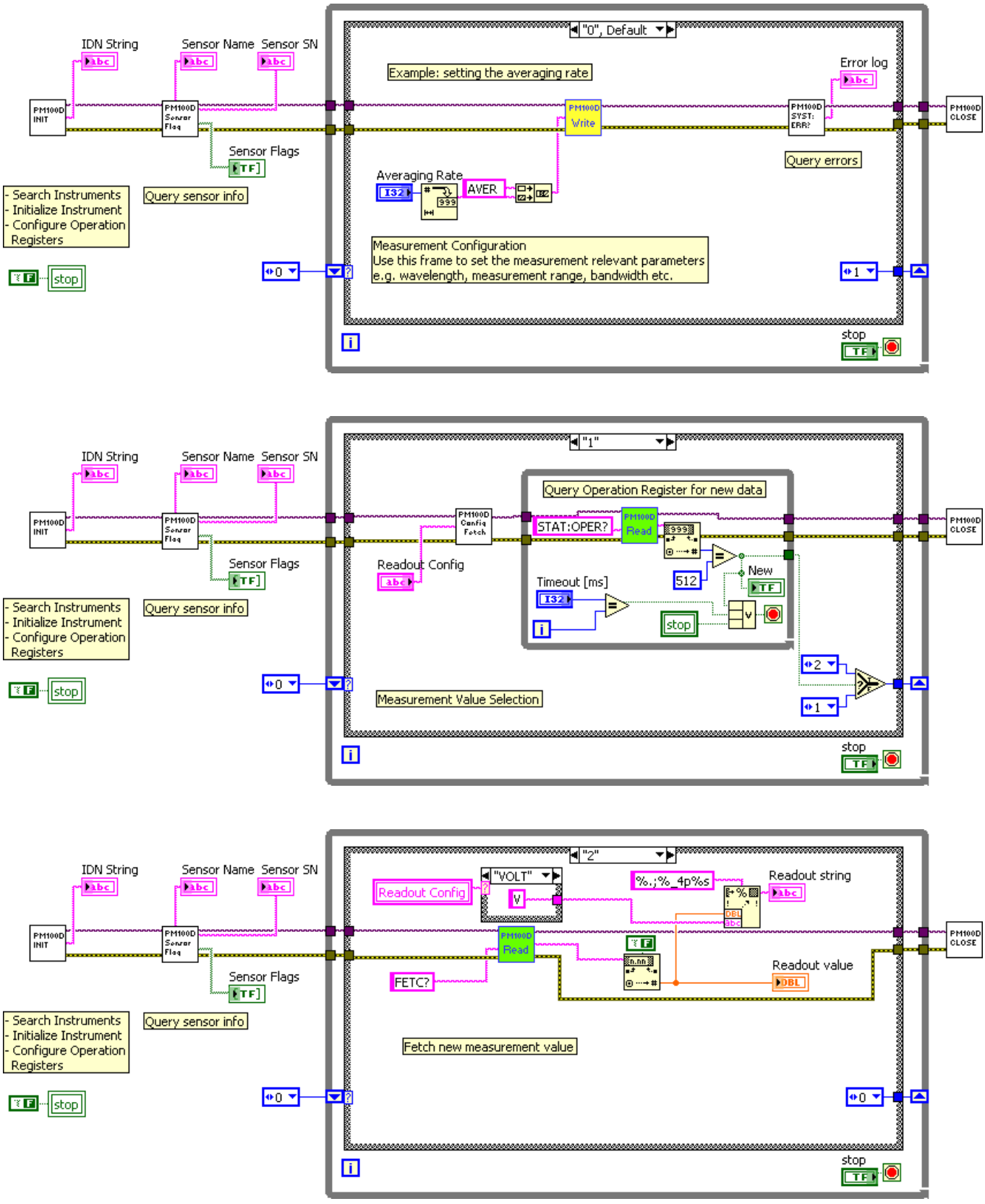
PM100D Simple Example.vi

This VI shows how to communicate with a PM200 optical power/energy meter with SCPI commands. The following steps are demonstrated within this application:

- Initializing the instrument
- Getting system info
- Setting parameters
- Measurement configuration
- Measuring queue
- Fetching and displaying a measurement value
- Closing the application



Block diagram



Averaging Rate
Sets the averaging rate - 3000 averages take approximately 1s for performing a new measurement value

Readout Config
Selects the measurement parameter

- POW power measurement in W
- CURR current measurement in A
- VOLT voltage measuremetn in V
- ENER energy measurement in J
- FREQ frequency / repetition rate measurement in Hz
- PDEN power density measurement in W/cm²
- EDEN energy density measurement in J/cm²
- RES thermistor resistance measurement in Ohm
- TEMP temperature measurement in °C

Stop
Stops application

Timeout [ms]
Sets a timeout value in ms that allows the instrument to sample.

The timeout must be longer than it takes to perform a new measurement. This has especially to be considered when performing single shot energy measurements.

Error log
Error indicator, 'no error' is suppressed

Readout string
Formatted measurement value

- limitation to 4 significant digits
- SI notation
- '.' decimal separator
- appended unit according readout configuration

New
Indicator lights up, when a new measurement value is processed

Sensor Flags

Sensor flag bitmap:
1 Is power sensor
2 Is energy sensor
16 Response settable
32 Wavelength settable
64 Tau settable
256 Has temperature sensor

Sensor Name
Name of connected power/energy sensor

Sensor SN
Serial number of connected power/energy sensor

IDN String
Answer from instrument on *idn? command:

- manufacturer

- model number
- serial number
- firmware version

Readout value
Plain readout value in full resolution

PM100D_Initialize.vi
This VI scans for connected devices, that can be selected in a dialog box. Next steps

- setting timeout
- performing identification query
- configuring the operation register to '512'; flag gets to HI when a new measurement value is ready to fetch
- clear operation register

PM100D_Write.vi
Writes a SCPI command to the connected instrument

PM100D_Read.vi
Reads data from the connected instrument. All query commands according the SCPI command table are terminated by a question mark (?)

PM100D_SensorFlag.vi
This VI queries all relevant sensor info with SYST:SENS:IDN?

- sensor name
- sensor serial number
- calibration message
- sensor type
- sensor sub-type
- sensor flags

PM100D_ConfMeas.vi

- CONFigure measurement; CONF:<parameter> (POW, ENER, etc.)
- ABORt measurement
- Clear operation register with STAT:OPER?
- INITiate measurement

PM100D_SYST-ERR.vi
This VI lists all errors coming from the instrument with the command SYST:ERR?

'no error' is suppressed

PM100D_Close.vi
Closes the VISA session

Sets the connected instrument in local mode (default option)

2 SCPI Commands

This section describes in detail the SCPI command set with respect to PM200 power meter console. However, these commands are compatible with PM100A, PM100D and PM100USB consoles as well as with the PM160 Hand-held Power Meter, except some console and sensor related functions.

2.1 An Introduction to the SCPI language

The PM200 interface commands use the SCPI (Standard Commands for Programmable Instruments), an ASCII-based command language that was designed for test and measurement instruments.

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; WAV 1310"
```

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

```
"CORR:BEAM 1"
"CORR:WAV 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:WAV 1550"
You can query the operating wavelength by executing: "CORR:WAV?"
You can also query the minimum or maximum operating wavelength allowed as follows:

```
"CORR:WAV? MIN"
"CORR:WAV? 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>`

2.2 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?

2.2.1 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

2.2.2 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.

2.2.3 PM200 specific SCPI Command Reference

See also SCPI Specification, Version 1999.0, May, 1999, http://www.scpiconsortium.org . All commands with a 'SCPI' checkmark are described in the SCPI specification.

All described commands work also with the PM100D, PM100A, PM100USB and PM160 instruments (with some limitations due to the hardware capabilities).

2.2.3.1 SYSTem subsystem commands

Command	Description
SYSTem	Path to SYSTem subsystem. (SCPI Vol.2 §21)
:BEEPer	
[:IMMediate]	Issue an audible signal. (SCPI Vol.2 §21.2.2)
:STATe {ON|1|OFF|0}	Activate/deactivate the beeper. (SCPI Vol.2 §21.2.3)
:STATe?	Return the state of the the beeper (SCPI Vol.2 §21.2.3)
:ERRor	
[:NEXT]?	Returns the latest <error code, "message">. (SCPI Vol.2 §21.8.8)
:VERSion?	Query level of SCPI standard (1999.0) . (SCPI Vol.2 §21.21)
:DATE <year>,<month>,<day>	Sets the instrument's calendar. (SCPI Vol.2 §21.7)
:DATE?	Query the instrument's calendar. (SCPI Vol.2 §21.7)
:TIME <hour>,<min>,<sec>	Sets the instrument's clock. (SCPI Vol.2 §21.19)
:TIME?	Query the instrument's clock. (SCPI Vol.2 §21.19)
:LFRequency <numeric value>	Sets the instrument's line frequency setting to 50 or 60Hz. (SCPI Vol.2 §21.13)
:LFRequency?	Query the instrument's line frequency setting. (SCPI Vol.2 §21.13)
:SENSor	
:IDN?	Query information about the connected sensor. This is a query only command. The response consists of the following fields: <name>,<sn>,<cal_msg>,<type>,<subtype>,<flags> <name> Sensor name in string response format <sn> Sensor serial number in string response format <cal_msg> calibration message in string response format <type> Sensor type in NR1 format <subtype> Sensor subtype in NR1 format <flags> Sensor flags as bitmap in NR1 format. Flag: / Dec.value: Is power sensor 1 Is energy sensor 2 Response settable 16 Wavelength settable 32 Tau settable 64 Has temperature sensor 256

2.2.3.2 STATus subsystem commands

Command	Description
STATus	Path to STATus subsystem. (SCPI Vol.2 §20)
:MEASurement	Path to control measurement event registers
[:EVENt]?	Read the event register
:CONDition?	Read the condition register
:PTRansition <value>	Program the positive transition filter
:PTRansition?	Read the positive transition filter
:NTRansition <value>	Program the negative transition filter
:NTRansition?	Read the negative transition filter
:ENABle <value>	Program the enable register
:ENABle?	Read the enable register
:AUXillary	Path to control measurement event registers
[:EVENt]?	Read the event register
:CONDition?	Read the condition register
:PTRansition <value>	Program the positive transition filter
:PTRansition?	Read the positive transition filter
:NTRansition <value>	Program the negative transition filter
:NTRansition?	Read the negative transition filter
:ENABle <value>	Program the enable register
:ENABle?	Read the enable register
:OPERation	Path to control operation event registers
[:EVENt]?	Read the event register
:CONDition?	Read the condition register
:PTRansition <value>	Program the positive transition filter
:PTRansition?	Read the positive transition filter
:NTRansition <value>	Program the negative transition filter
:NTRansition?	Read the negative transition filter
:ENABle <value>	Program the enable register
:ENABle?	Read the enable register
:QUEStionable	Path to control questionable event registers
[:EVENt]?	Read the event register
:CONDition?	Read the condition register
:PTRansition <value>	Program the positive transition filter
:PTRansition?	Read the positive transition filter
:NTRansition <value>	Program the negative transition filter
:NTRansition?	Read the negative transition filter
:ENABle <value>	Program the enable register
:ENABle?	Read the enable register
:PRESet	Return status registers to default states.

2.2.3.3 DISPlay subsystem commands

Command	Description
DISPlay	Path to DISPlay subsystem. (SCPI Vol.2 §8)
:BRIGhtness <value>	Set the display birghtness. (SCPI Vol.2 §8.2)
:BRIGhtness?	Return the display birghtness value. (SCPI Vol.2 §8.2)
:CONTrast <value>	Set the display contrast. (SCPI Vol.2 §8.4)
:CONTrast?	Return the display conrast value. (SCPI Vol.2 §8.4)

2.2.3.4 CALibration subsystem commands

Command	Description
CALibration	Path to CALibration subsystem. (SCPI Vol.2 §5)
:STRing?	Returns a human readable calibration string. This is a query only command. The response is formatted as string response data.

2.2.3.5 SENSe subsystem commands

Command	Description
SENSe	Path to SENSe subsystem. (SCPI Vol.2 §18)
AVERage	
[:COUNt] <value	Sets the averaging rate (1 sample takes approx. 3ms)
[:COUNt]?	Queries the averaging rate
CORRection	
[:LOSS[:INPut[:MAGNitude]]] {MINimum|MAXimum|DEFault| <numeric_value>}	Sets a user attenuation factor in dB
[:LOSS[:INPut[:MAGNitude]]]? [{MINimum|MAXimum|DEFault}]	Queries the user attenuation factor
COLLect	
ZERO	
[:INITiate]	Performs zero adjustment routine
ABORt	Aborts zero adjustment routine
STATe?	Queries the zero adjustment routine state
MAGNitude?	Queries the zero value
BEAMdiameter {MINimum| MAXimum|DEFault| <numeric_value>[mm]}	Sets the beam diameter in mm
BEAMdiameter? [{MINimum| MAXimum|DEFault}]	Queries the beam diameter
WAVelength {MINimum| MAXimum| <numeric_value>[nm]}	Sets the operation wavelength in nm
WAVelength? [{MINimum| MAXimum}]	Queries the operation wavelength
POWer	
[:PDIOde]	Sets the photodiode response value in A/W

Command	Description
[:RESPonse] {MINimum|MAXimum|DEFault|<numeric_value>[A]}	
[:RESPonse]?[{MINimum|MAXimum|DEFault}]	Queries the photodiode response value
:THERmopile	
[:RESPonse] {MINimum|MAXimum|DEFault|<numeric_value>[V]}	Sets the thermopile response value in V/W
[:RESPonse]?[{MINimum|MAXimum|DEFault}]	Queries the thermopile response value
ENERgy	
[:PYRO]	
[:RESPonse] {MINimum|MAXimum|DEFault|<numeric_value>[V]}	Sets the pyro-detector response value in V/J
[:RESPonse]?[{MINimum|MAXimum|DEFault}]	Queries the pyro-detectro response value
CURRent[:DC]	
RANGe	
AUTO {OFF|0|ON|1}	Switches the auto-ranging function on and off
AUTO?	Queries the auto-ranging function state
[:UPPer] {MINmum|MAXimum|<numeric_valuje>[A]}	Sets the current range in A
[:UPPer]? [{MINimum|MAXimum}]	Queries the current range
REFerence {MINimum|MAXimum|DEFault|<numeric_value>[A]}	Sets a delta reference value in A
REFerence? [{MINimum|MAXimum|DEFault}]	Queries the delta reference value
STATe {OFF|0|ON|1}	Switches to delta mode
STATe?	Queries the delta mode state
ENERgy	
RANGe	
[:UPPer] {MINmum|MAXimum|<numeric_valuje>[J]}	Sets the energy range in J
[:UPPer]? [{MINimum|MAXimum}]	Queries the energy range
REFerence {MINimum|MAXimum|DEFault|<numeric_value>[J]}	Sets a delta reference value in J
REFerence? [{MINimum|MAXimum|DEFault}]	Queries the delta reference value
STATe {OFF|0|ON|1}	Switches to delta mode
STATe?	Queries the delta mode state
FREQuency	
Range	

Command	Description
[UPPer]?	Queries the frequency range
LOWer?	
POWer[:DC]	
RANGe	
AUTO {OFF|0|ON|1}	Switches the auto-ranging function on and off
AUTO?	Queries the auto-ranging function state
[:UPPer] {MINmum|MAXimum|<numeric_valuje>[W]}	Sets the current range in W
[:UPPer]? [{MINimum|MAXimum}]	Queries the current range
REFerence {MINimum|MAXimum|DEFault|<numeric_value>[W]}	Sets a delta reference value in W
REFerence? [{MINimum|MAXimum|DEFault}]	Queries the delta reference value
STATe {OFF|0|ON|1}	Switches to delta mode
STATe?	Queries the delta mode state
UNIT {W|DBM}	Sets the power unit W or dBm
UNIT?	Queries the power unit
VOLTage[:DC]	
RANGe	
AUTO {OFF|0|ON|1}	Switches the auto-ranging function on and off
AUTO?	Queries the auto-ranging function state
[:UPPer] {MINmum|MAXimum|<numeric_valuje>[V]}	Sets the current range in V
[:UPPer]? [{MINimum|MAXimum}]	Queries the current range
REFerence {MINimum|MAXimum|DEFault|<numeric_value>[V]}	Sets a delta reference value in V
REFerence? [{MINimum|MAXimum|DEFault}]	Queries the delta reference value
STATe {OFF|0|ON|1}	Switches to delta mode
STATe?	Queries the delta mode state
PEAKdetector	
[:THReshold] {MINimum|MAXimum|DEFault|<numeric_value>	Sets the trigger level in % for the energy mode
[:THReshold]? [{MINimum|MAXimum|DEFault}	Queries the trigger level setting

2.2.3.6 INPut subsystem commands

Command	Description
INPut [:PDIode] :FILTer [:LPASs] [STATe] {OFF|0|ON|1} [STATe]?	 Sets the bandwidth of the photodiode input stage Queries the bandwidth of the photodiode inut stage
:THERmopile :ACCelerator [STATe] {OFF|0|ON|1} [STATe]? :AUTO {OFF|0|ON|1} ? :TAU {MINimum| MAXimum|DEFault| <numeric_value>[s]} :TAU? [{MINimum| MAXimum|DEFault}]	 Sets the thermopile accelerator state Queries the thermopile accelerator state Sets the thermopile accelerator to auto mode Queries thermopile accelerator auto mode Sets thermopile time constant 0-63% in s Queries the thermopile time constant in s
:ADAPter [:TYPE] {PHOTodiode| THERmal|PYRo} [:TYPE]?	 Sets default sensor adapter type Queries default sensor adapter type

2.2.3.7 Measurement commands

Command	Description
INITiate[:IMMediate]	Start measurement
ABORt	Abort measurement
CONFigure[:SCALar]	
[:POWer]	Configure for power measurement
:CURRent[:DC]	Configure for current measurement
:VOLTage[:DC]	Configure for voltage measurement
:ENERgy	Configure for energy measurement
:FREQuency	Configure for frequency measurement
:PDENsity	Configure for power density measurement
:EDENsity	Configure for energy density measurement
:RESistance	Configure for sensor presence resistance measurement
:TEMPerature	Configure for sensor temperature measurement
MEASure[:SCALar]	
[:POWer]	Performs a power measurement
:CURRent[:DC]	Performs a current measurement
:VOLTage[:DC]	Performs a voltage measurement
:ENERgy	Performs a energy measurement
:FREQuency	Performs a frequency measurement
:PDENsity	Performs a power density measurement
:EDENsity	Performs a energy density measurement
:RESistance	Performs a sensor presence resistance measurement
:TEMPerature	Performs a sensor temperature measurement
FETCh?	Read last measurement data (SCPI Vol.2 §3.2)
READ?	Start new measurement and read data (SCPI Vol.2 §3.3)
CONFigure?	Query the current measurement configuration.