Command Line Interface

Manual

The Pure Photonics Command Line Interface is a utility to communicate with OIF MSA based tunable lasers over a serial port. It includes commands to address all standardized registers and also specific commands to access Pure Photonics' specific functionality.

This manual describes the commands with a focus on the use with Pure Photonics lasers.

1. Table of Contents

2. Software usage

The software can be downloaded from the Pure Photonics website (purephotonics.com) under the support section.

No installation is needed. The unzipped directory can be placed at any location.

The software is run by double clicking the ‘Pure Photonics CLI.exe’ file.

This manual is based on the software version 3.2.10. This version is written in the Python 3 language and operates with the Windows 11 operating system (previous versions did not operate with Windows 11) and earlier versions. This version does not work with a Linux operating system.

In addition, this version works with COM ports with numbers larger than 9 and it can handle both textual com-ports (such as ‘com8’) and numbered ones (e.g. 8).

Also, a help function has been added by typing ‘it.help()’ after connecting to a serial port.

The software has a built in feature to check for updates every 30 days. This can be delayed to the next use or for a further 30 days at each occasion You can also manually check the latest release at https://purephotonics.com/CLI_VERSION_STATUS . If a new version is available the user can manually download this from https://purephotonics.com/support/ under ‘software’ and ‘CLI’.

3. Usage

A connection with a serial port is set up with the it.connect(port,baud) command or the it.connectCoBrite(port)/it.connectDX(port,chassis,slot,device) command (in case of a CoBrite unit). Port is the COM port number of the serial port. Entries such as 8 or ‘COM8’ are both accepted. Baud is the baudrate. For most units it would typically be 9600, but could be as high as 115200. For the CoBrite unit the default communication baudrate is 115200 (though note that internally in the CoBrite unit the communication rate is 9600).

The connection is disconnected through the it.disconnect() command. Note that a serial port can only be accessed by one client at a time, so the disconnect command needs to be used before accessing with a different application. Closing a CLI window also closes the connection to the serial port.

Several CLI windows can be opened at the same time to access different serial connections. Only one serial connection can be accessed at a time in the same window.

4. Commands

The following commands are available through the CLI, after connecting to the serial port. Each command starts with it. .

Command	Return
General commands	
connect(port=1, baud=None)	General; Connect to a serial port. When baud is not give, it will try to auto detect.
connectCoBrite(port=1)	General; Connect to a CoBrite port.
connectDX(port=1, chassis=1, slot=1, device=1)	General; Connect to a DX port.
disconnect()	General; Disconnect from port.
help()	General; Provide help index
flushBuffer()	General; Read all bytes in receive queue and discard
reset()	General; Resynchronize the transmit and receive buffer
upgrade(target, filename, version='Interrupting')	General; Upgrade firmware
script(filename, echo=False)	Opens a filename and runs it line by line; if echo is True the window will also echo the command.
OIF commands	
aeaEa()	Read only; AEA EA register
aeaEac()	Read only; AEA EAC register
aeaEar(write=False)	Read only; AEA EAR register
age()	Read only; Get laser age in percentage
almT(wvsf=None, wfreq=None, wtherm=None, wpwr=None, fvsf=None, ffreq=None, ftherm=None, fpwr=None)	Read/Write; Set/Get alarm triggers
baudrate(baudrate=None)	Read/Write; Compound command; Set/Get (integer) baud rate and reconnect with new baud rate, return tuple (status_string, baud_rate)
channel(channel=None)	Read/Write; Compound command; Set/Get channel1 and channel2
channel1(channel=None)	Read/Write; Set/Get lower 2 bytes of channel
channel2(channel=None)	Read/Write; Set/Get upper 2 bytes of channel (MSA1.3)
ctemp()	Read only; Get current temperature in degreeC*100
currents()	Read only; Device currents in mA*10
devTyp()	Read only; Device Type
ditherA(gain=None)	Read/Write; Get/Set dither gain in percent
ditherE(wf=None, de=None)	Read/Write; Get/Set dither enable
ditherF(width=None)	Read/Write; Get/Set dither width in GHz * 10
ditherR(rate=None)	Read/Write; Get/Set dither rate in KHz
dlConfig(init_write=None, abrt=None, done=None, init_read=None, init_check=None, init_run=None, runv=None, type=None)	Read/Write; dlConfig register

dlStatus()	Read only; dlStatus register
ea()	Read only; AEA EA register
eac()	Read only; AEA EAC register
ear(value=None)	Read only; AEA EAR register
fAgeTh(threshold=None)	Read/Write; Get/Set fatal laser age threshold in percentage
fFreqTh()	Read/Write; Get fatal frequency threshold in GHz*10
fPowTh(dB100=None)	Read/Write; Set/Get fatal power threshold in dB*100
fThermTh()	Read/Write; Get fatal thermal threshold in degC*100
fatalT(wvsfl=None, wfreql=None, wtherml=None, wpwrl=None, mrl=None, fvsfl=None, ffreql=None, ftherml=None, fpwrl=None)	Read/Write; Set/Get fatal triggers
fcf(frequency=None)	Read/Write; Compound command; Get/Set first channel frequency in THz (fcf1, fcf2, fcf3)
fcf1(fTHz=None)	Read/Write; Get/Set first channel frequency THz portion
fcf2(fGHz10=None)	Read/Write; Get/Set first channel frequency 100MHz portion
fcf3(MHz=None)	Read/Write; Get/Set first channel frequency MHz portion
ftf(MHz=None)	Read/Write; Get/Set the fine tune frequency in MHz
ftfR()	Read only; Get maximum range for FTF (MHz)
genCfg(sdc=None)	Read/Write; Set/Get General Module Configuration
grid(frequency=None)	Read/Write; Get/Set grid spacing in GHz*10
grid2(frequency=None)	Read/Write; Get/Set grid2 spacing in MHz (MSA1.3)
health()	Read only; Get the health status (status_string, 16bit status report)
ioCap(baudrate=None, module_select_no_reset=True)	Read/Write; Set/Get ioCap register (baudrate)
isLocked(timeout=25.0)	Read only; Query NOP unitl pending is cleared, timeout in seconds. Return (boolean, lock_time_in_seconds)
lf()	Read only; Compound command; Get channel frequency in THz (lf1, lf2, lf3)
lfh()	Read only; Compound command; Get laser last frequency in THz in THz (lfh1, lfh2, lfh3)
lfl()	Read only; Compound command; Get laser first frequency in THz (lfl1, lfl2, lfl3)
lgrid()	Read only; Get laser minimum supported grid spacing in GHz*10
lgrid2()	Read only; Get laser minimum supported grid spacing, MHz portion
lstResp()	Read only; Last response register
mcb(sdf=None, adt=None, autostart=None, whisperstart=None, ditherreduce=None)	Read/Write; Get/Set module configuration behavior
mfgDate()	Read only; Manufacturing Date

mfgr()	Read only; Manufacturer
model()	Read only; Model
nop()	Read only; NOP register
oop()	Read only; Get optical output power in dBm*100
opsh()	Read only; Get maximum power setting in dBm*100
opsl()	Read only; Get minimum power setting in dBm*100
pwr(power=None)	Read/Write; Get/Set power set point in dBm*100
read_string(byte_count=1)	Read/Write; Read string directly from serial port
relBack()	Read only; Release backwards compatibility
release()	Read only; Release information
resena(sena=None, sr=None, mr=None)	Read/Write; Get/Set reset/enable
serNo()	Read only; Serial Number
srqT(dis=None, wvsfl=None, wfreql=None, wtherml=None, wpwrl=None, xel=None, cel=None, mrl=None, crl=None, fvsfl=None, ffreql=None, ftherml=None, fpwrl=None)	Read/Write; Set/Get SRQ triggers
statusF(xel=0, cel=0, mrl=0, crl=0, fvsfl=0, ffreql=0, ftherml=0, fpwrl=0)	Read/Write; Get/Set status fatal
statusW(xel=0, cel=0, mrl=0, crl=0, wvsfl=0, wfreql=0, wtherml=0, wpwrl=0)	Read/Write; Get/Set status warning
temps()	Read only; Device temperatures in C*100
toModulePacket()	Read Only; Return last packet sent to module.
wAgeTh(threshold=None)	Read/Write; Get/Set warning laser age threshold in percentage
wFreqTh()	Read/Write; Get warning frequency threshold in GHz*10
wPowTh(dB100=None)	Read/Write; Set/Get warning power threshold in dB*100
wThermTh()	Read/Write; Get warning thermal threshold in degC*100
write_string(string)	Read/Write; Write string directly on serial port
Pure Photonics Specific Registers	
cleanMode(cleanmode=None)	Read/Write; Get/Set Clean Mode
cleanJumpEnable(enable=None, setchannel=0)	Read/Write; Enable/Disable Clean Jump (1/0) and select channel.
CleanJumpOffset()	Read only; Provides clean jump offset.
CleanJumpCalibrate(channels=None)	Read/Write; Starts calibration for number of channels or return the current channel that is being calibrated.
cleanSweepAmplitude(amplitude=None)	Read/Write; Get/Set Clean Sweep Amplitude (GHz)
cleanSweepEnable(enable=None)	Read/Write; Enable/Disable Clean Sweep (1/0)
cleanSweepOffset()	Read/Write; Get frequency offset (GHz * 10)
cleanSweepRate(rate=None)	Read/Write; Set maximum sweep rate (GHz/sec)
cleanSweepTriggers(triggers=None)	Read/Write; Set triggers for clean Sweep
ICR PPEB076 Registers (activate with it.setICR(True)	
setICR(value)	Write only; activates ICR commands.
ICRGain(ch=0, volts=None)	Read/Write; Get/Set outut gain value

ICRMGCAGC(value=None)	Read/Write; Get/Set manual and automatic gain mode
ICROutputAdjust(ch=0, volts=None)	Read/Write; Get/Set outut adjust value
ICRPDMode(value=None)	Read/Write; Get/Set Photodiode Mode
ICRPDValue(channel=0)	Read only; Get photodiode current
ICRPeakV(channel=0)	Read only; Get peak value
ICRShutdown(value=None)	Read/Write; Get/Set shutdown
ICRTIA(channel=None)	Read/Write; Get/Set TIA enable
ICRTIACurrent(channel=0)	Read only; Get peak value
ICRVOA(value=None)	Read/Write; Get/Set VOA voltage in V
ICRDEBUGGAIN(ch=0)	Read only; Get ouptut gain settng (debug register)
ICRDEBUGOUTADJUST(ch=0)	Read only; Get ouptut adjust settng (debug register)
ICRDEBUGRESISTANCE(ch=0, value=None)	Read/Write; Get/Set resistance reading (debug register)
ICRDEBUGSAMPLE(value=0)	Read only; Get sample reading (debug register)
Analog Array Registers (activate with it.setAnalogArray(True))	
AAPower(ch=None)	Read only; read power on current channel or of specific channel
AAVOAMode(ch=None,mode=None)	Read/Write; Get/Set VOA mode (0 for constant absorption and 1 for constant power) for current channel or for specific channel
AAVOAAbs(ch=None,dB=None)	Read/Write; Get/Set VOA absorption value (in dB) for current channel or for specific channel
AAVOAPowerTarget(ch=None,dBm=None)	Read/Write; Get/Set power target (in dBm) for current channel or for specific channel
AAPassthrough(array=0)	Write only; only for a system to bypass the controller and speak with the array directly.
Legacy Registers (activate with it.setLegacy(True))	
setLegacy(value)	Write only; Set Legacy status. By default False, True for ITLA commands.
cleanJumpCurrent(mA10=None)	Read/Write; Set sled temperature of the next Clean Jump step (1000*C)
cleanJumpGHz(GHz=None)	Read/Write; Set GHz portion of the next Clean Jump step (10*GHz)
cleanJumpSled(Cdeg=None)	Read/Write; Set sled temperature of the next Clean Jump step (1000*C)
cleanJumpTHz(THz=None)	Read/Write; Set THz portion of the next Clean Jump step (THz)
cleanScanCalibration(factor1, factor2=None)	Write only; Load the calibration factors for Clean Scan (2)
cleanScanEnable(enable=None)	Read/Write; Enable/Disable Clean Scan (1/0)
cleanScanOffset()	Read/Write; Get frequency offset (GHz * 10)
cleanScanSetF1(degC=None)	Read/Write; Set target filter1 temperature for next center point (C)
cleanScanSetF2(degC=None)	Read/Write; Set target filter2 temperature for next center point (C)

cleanScanSetSled(degC=None)	Read/Write; Set target sled temperature for next center point (C)
cleanSweepConstants(Tminus10, T0, T10, T20, T30, T40, T50, T60, T70, lowtempvalue, hightempvalue)	Write only; Provide calibration constants to the laser for extended sweep (9 current values at different temperatures [mA] and 2 correction factors in C/GHz)
noDriftMode(enable=None)	Read/Write; Enable/Disable NoDrift Mode (1/0)
PPCL590 Registers (activate with it.setPPCL590(True))	
PPCL590RMSValue(longterm=False)	Reads register 0x96 for the RMS frequency offset value; False for integration over 1 second; True for integration over 20 seconds
PPCL590Lockstate()	Reads register 0xfd with value 0x9000; provides the lockstate of the PPCL590 lock
PPCL590PZTsignal()	Reads register 0xfd with value 0x9001; provides the DAC value to the PZT which controls the fast frequency correction; should be around 0x8000 when locked.
PPCL590Locksignal()	Reads register 0x93; provides the feedback signal received from the photodiode. Target is 0x8000 in the PPCL590 and could be a specified value in the external lock situation
PPCL590Outputsignal()	Reads register 0x93; provides the analog output signal; For complex operation (such as the PPCL590 and complex external lock) this will be larger than 0.
PPCL590Setfeedback(value=50)	Reads register 0xfd with value 0xe000+value; This command should only be used by experienced users and for external lock applications.

5. Registers

The OIF MSA defines the following registers:

Command	Register Name	Read / Write	AEA	Non-volatile (NV)	Description
General Module Commands					
0x00	NOP	R/W			Provide a way to read a pending response as from an interrupt, to determine if there is pending operation, and/or determine the specific error condition for a failed command.
0x01	DevTyp	R	AEA		Returns device type (tunable laser source, filter, modulator, etc) as a null terminated string.
0x02	MFGR	R	AEA		Returns manufacturer as a null terminated string in AEA mode (vendor specific format)
0x03	Model	R	AEA		Returns a model null terminated string in AEA mode (vendor specific format)
0x04	SerNo	R	AEA		Returns the serial number as null terminated string in AEA mode
0x05	MFGDate	R	AEA		Returns the mfg date as a null terminated string.
0x06	Release	R	AEA		Returns a manufacturer specific firmware release as a null terminated string in AEA mode
0x07	RelBack	R	AEA		Returns manufacturer specific firmware backwards compatibility as a null terminated string
0x08	GenCfg	RW			General module configuration
0x09	AEA-EAC	R			Automatic extended address configuration register
0x0A	AEA-EA	R			Automatic extended address (16 bits)
0x0B	AEA-EAR	RW			Location accessed "thru" AEA-EA and AEA-EAC
0x0C	Reserved				
0x0D	IOCap	RW		NV	Physical interface specific information (such as data rate, etc.)
0x0E	EAC	RW			Extended address configuration register - auto incr/decr flag on read and on write and additional address bits
0x0F	EA	RW			Extended address (16 bits)
0x10	EAR	RW			Location accessed "thru" EA and EAC
0x13[24]	LstResp	R			Returns last response
0x14	DLConfig	RW			Download configuration register
0x15	DLStatus	R			Download status register
0x17 – 0x1F	Reserved	–	–		

Module Status Commands					
0x20	StatusF	RW			Contains reset status, optical faults and alarms, and enable status.
0x21	StatusW	RW			Contains reset status, warning optical faults and alarms, and enable status.
0x22	FPowTh	RW		NV	Returns/Sets the threshold for the output power FATAL condition encoded as ±dB*100
0x23	WPowTh	RW		NV	Returns/Sets the threshold for the power warning encoded as ±dB*100
0x24	FFreqTh	RW		NV	Returns/Sets the threshold for the frequency FATAL condition encoded as ±GHz*10. Also see the optional MHz resolution FFreqTh2 register 0x63
0x25	WFreqTh	RW		NV	Returns/Sets the threshold for the frequency error warning encoded as ±GHZ*10. Also see the optional MHz resolution WFreqTh2 register 0x64.
0x26	FThermTh	RW		NV	Returns/Sets the threshold for thermal deviations (> ±°C*100) at which FATAL is asserted.
0x27	WThermTh	RW		NV	Returns/Sets the threshold for thermal deviations (> ±°C*100) at which a warning is asserted.
0x28	SRQT	RW		NV	Indicates which bits in the Fatal & Warning status registers, 0x20-0x21, cause a SRQ condition and asserts the SRQ* line.
0x29	FatalT	RW		NV	Indicates which bits in the Fatal & Warning status register, 0x20-0x21, assert a FATAL condition
0x2A	ALMT	RW		NV	Indicates which bits in the status registers, 0x20, 0x21, cause an alarm condition. (Default behavior asserted whether laser is LOCKED on frequency.
0x2B – 0x2F	Reserved				

Module Optical Commands					
0x30	Channel	RW		NV	Setting valid channel causes a tuning operation to occur. Also see the optional MHz resolution ChannelH register 0x65.
0x31	PWR	RW		NV	Sets the optical power set point as encoded as dBm*100
0x32	ResEna	RW			Reset/Enable - Enable output, hard and soft reset
0x33	MCB	RW		NV	Various module configurations
0x34	GRID	RW		NV	Allows the grid spacing to be set for channel numbering. Also see the optional MHz resolution GRID2 register 0x66.
0x35	FCF1	RW		NV	Allows the first channel's frequency to be defined for channel numbering. (THz) Also see the optional MHz resolution FCF3 register 0x67.
0x36	FCF2	RW		NV	Allows the first channel's frequency to be defined for channel numbering. (GHz*10) Also see the optional MHz resolution FCF3 register 0x67.
0x37 – 0x3F	Reserved				Reserved for OIF configuration registers
0x40	LF1	R			Returns channel's frequency as THz. Also see the optional MHz resolution LF3 register 0x68.
0x41	LF2	R			Returns channel's frequency as GHZ*10. Also see the optional MHz resolution LF3 register 0x68.
0x42	OOP	R			Returns the optical power encoded as dBm*100
0x43	CTemp	R			Returns the current temperature (monitored by the temperature alarm) encoded as °C*100.
0x44 – 0x4E	Reserved				Reserved for OIF status registers

Module Capabilities					
0x4F	FTFR	R			Returns min/max fine tune frequency range (MHz)
0x50	OPSL	R			Returns the min possible optical power setting
0x51	OPSH	R			Returns the max possible optical power setting
0x52	LFL1	R			Laser's first frequency (THz). Also see the optional MHz resolution LFL3 register 0x69
0x53	LFL2	R			Laser's first frequency (GHz*10). Also see the optional MHz resolution LFL3 register 0x69
0x54	LFH1	R			Laser's last frequency (THz). Also see the optional MHz resolution LFH3 register 0x6A
0x55	LFH2	R			Laser's last frequency (GHz*10). Also see the optional MHz resolution LFH3 register 0x6A
0x56	LGrid	R			Laser's minimum supported grid spacing (GHz*10). Also see the optional MHz resolution LGrid2 register 0x6B

Additional registers that are added on (some) Pure Photonics products are below. Please refer to specific application notes and product capabilities to understand which registers are applicable to specific firmware versions and devices.

Name	R/W	Description
0x90	RW	Enable/disable whispermode
0x93	R	Analog input (Clean Measurement)
0x94	RW	Analog output
0x95	RW	Dither reduction
0x96	R	PPCL590 lock error
0x99	W	Mode-offset in whispermode
0xD0	RW	Select/Enable Clean Jump
0xD1	R	Clean Jump offset
0xD2	RW	Clean Jump Calibration
0xE4	RW	Clean Sweep Range
0xE5	RW	Clean Sweep Enable
0xE6	RW	Clean Sweep Offset
0xE7	RW	Clean Sweep Sweeprate
0xE8	RW	Clean Sweep Triggers
0xF8	RW	Analog FTF
0xFD	RW	Debug register

6. Creating custom commands

The underlying engine of the command line interface is the Python programming language. Variables can be defined, as well as functions. Certain modules, such as e.g. time can be imported.

For example the following line is checking the NOP response until the pending flags drop before moving into whispermode.

```
goon=True
while goon:
    if it.nop()[1].data()&0xff00==0:
        time.sleep(5)
        it.cleanMode(2)
        goon=False
    time.sleep(1)
```

In case a register needs to be accessed that has no custom command, a variable can be generated with a command packet. That packet can then be modified to access a different register.

e.g.:

```
it.oop()
readpacket=it.toModulePacket()
readpacket.register(0x90)
it.packet(readpacket)
```

or

```
it.pwr(1350)
writepacket=it.toModulePacket()
writepacket.register(0x90)
writepacket.data(2)
it,packet(writepacket)
```

7. Running scripts

To make it easier to automate tasks in the CLI the user can define scripts that run a certain task or define additional functions. A script is run with it.script(filename). This command will execute each line in the text file as if it was typed in the interface. If the debug toggle is set (it.script(filename,True)) then the command that is being executed is echo'ed.

The script is automatically split up in segments. That means that function definitions are executed as a whole and e.g. while loops are also operated as a loop. But separate lines (without indent) are all executed separately.

A user can create its own segmentation by adding '#CUSTOMSPLIT' at the top of the file (only having this would make the file executed as a whole) and addeing '#SPLIT' lines at each split point.

Once executing the script a window will pop up with the results of the execution. This window can be closed at any time, but will pop up again after completion of each segment.

Note that the script execution is sensitive to typos and logical errors. Letting the script run in segments will help you to more easily find troublesome segments.