# API Interface DLL Programmer Guide

# Content

## Relevant devices

This application note applies Neophotonics S500 Monitor device.

## 1. Introduction

The API Interface Dynamic Link Library (DLL) provides the following functionality: connect and disconnect device serial port, read device factory information including serial number and model number, read current measured power, read or set current device's status including wavelength, sample rate and etc.

The DLL is under license control, on the other hand, the DLL can only access some fixed S500 Monitor devices whose serial number is specified by license.

The procedures and guidelines presented in this document illustrate how to link the DLL to a client executable.

An example application (including program source code) developed by VC++2010 is provided along with the DLL as a means of testing or using the DLL as a base application.

## 2. Files and Compatibility

The DLL is developed by Microsoft Visual studio C++ 2010, meaning it use Visual Studio 2010 library (msvcr100.dll, msvcp100.dll). The DLL export two sets of interface: one is the a C++ class named "PowerMonitor", the other is standard "C" standard style functions. To use the "C++ class" interface of the DLL for development, client application also must use Visual Studio 2010 development environment or later version. But for the "C" style interface, it can be loaded by any Win32 programming environment. The two sets of interface are separated respectively in a same DLL file.

The whole Lib contains 4 files : S500API_NEO.h, S500API_NEO.lib, S500API_NEO.vb, and S500API_NEO.dll. For more information about linking to the interface DLL for development, Please refer to section 5 or 6 on Page 11

## 3. Class member functions

Following lists all the member functions of class “PowerMonitor” describing detailed functionality and usage.

### 3.1 PowerMonitor()

**Description:** Constructor of class PowerMonitor, it will open the specified serial port. And will reopen if current port has been open status

**Parameters:** **1.** char * port - Specified the ANSI string of target serial port to establish a connection. This port string will be stored in current object for subsequent open operation. e.g., “COM1”.

**Return Value:** None.

### 3.2 ~PowerMonitor()

**Description:** Destructor of class “PowerMonitor”, close the connection to serial port

**Parameters:** None

**Return Value:** None

### 3.3 setLicenseFilePath ()

**Description:** the member function set the directory in which license file located. This member function should be called right after calling constructor ()

**Parameters:** 1. char* pPath - an ANSI char string pointer that represents directory in which license files located.

**Return Value:** = true (license directory has been set successfully)
= false (error occurred).

### 3.4 isLicensed ()

**Description:** the member function detect whether the current device is licensed or not.

**Parameters:** None

**Return Value:** =true (means licensed)
=false (means not licensed)

### 3.5 open ()

**Description:** the member function used current serial port specified in parameter of constructor to open port. And will reopen if current serial port has been open status. It’s not necessary to call this member because this member has been already called in constructor.

**Parameters:** None

**Return Value:** = true (connected)
= false (not connected)

### 3.6 close()

**Description:** the member function closes the serial port. It’s not necessary to call this member because this member would be called in destructor.

**Parameters:** None

**Return Value:** = true (serial port has been closed successfully)
= false (error occurred)

### 3.7 isOpen()

**Description:** the member function detect whether the current serial port is open or not.

**Parameters:** None

**Return Value:** = true (current serial port is opened)
= false (current serial port is close)

### 3.8 flush()

**Description:** the member function flushes the buffer of current serial port.

**Parameters:** None

**Return Value:** = true (the buffer of current serial port has been flushed)
= false (error occurred)

### 3.9 getPower()

**Description:** the member function gets current reading for the device.

**Parameters:** None

**Return Value:** double

1. return a double type number with the power reading for current measurement setting (dB or dBm)
2. return 100, means error occurred or not licensed
3. return 101, means having no power (below the lowest limited power range)

### 3.10 getSampleRate()

**Description:** the member function gets sample rate of the device.

**Parameters:** None

**Return Value:** char
'f' means fast mode
's' means slow mode
'e' means error occurred or not licensed

### 3.11 setSampleRate()

**Description:** the member function sets the sample rate of the device.

**Parameters:** **1.** char – 'f' is for fast mode; 's' is for slow mode

**Return Value:** = true (command was sent successfully)
= false (error occurred or not licensed)

### 3.12 getWavelength()

**Description:** the member function gets current wavelength setting for the device.

**Parameters:** None

**Return Value:** int
Not 0, return an integer number in nanometer describing wavelength.
0, means error occurred or not licensed.

### 3.13 setWavelength()

**Description:** the member function sets the wavelength for the device.

**Parameters:** **1.** int – wavelength setting as integer, in nanometer

**Return Value:** = true (command was sent successfully)
= false (error occurred or not licensed)

### 3.14 getPolarization()

**Description:** the member function gets the current polarization mode for the device. It is only available for the device with Polarization Mode function, otherwise return "error".

**Parameters:** None

**Return Value:** char *, a pointer to DLL internal memory null-terminated string that would be override after calling next function.
"p1", polarization 1 mode
"p2", polarization 2 mode
"error", error occurred or not licensed

### 3.15 setPolarization()

**Description:** the member function sets the polarization mode for the device. It is only available for the device with polarization mode function, otherwise return false.

**Parameters:** char *, a ANSI null-terminated string describing the polarization mode. "p1" means polarization 1 mode, "p2" means polarization 2 mode.

**Return Value:** = true (command was sent successfully)
= false (error occurred or not licensed)

### 3.16 getMode()

**Description:** the member function gets the current power measurement mode for the device.

**Parameters:** None

**Return Value:** char *, a pointer to DLL internal memory null-terminated string that would be override after calling next function.
"abs", means absolute power measurement mode(dBm)
"rel", means relative power measurement mode(dB)
"error", means error occurred
"not licensed", means not licensed

### 3.17 setMode()

**Description:** the member function sets the power measurement mode for the device.

**Parameters:** char *, an ANSI null-terminated string describing the power measurement mode. "abs" means absolute mode, "rel" means relative mode.

**Return Value:** = true (command was sent successfully)
= false (error occurred or not licensed)

### 3.18 getSerial ()

**Description:** the member function gets the serial number for the device. This function is not under license control.

**Parameters:** None

**Return Value:** char *, a pointer to DLL internal memory null-terminated string that would be override after calling next function.
Return a string with 9-digit serial number
"error", means error occurred

### 3.19 getModel()

**Description:** the member function gets the model number for the device.

**Parameters:** None

**Return Value:** char *, a pointer to DLL internal memory null-terminated string that would be override after calling next function.
Return a string with 10-digit model number
"error", means error occurred

"not licensed", means not licensed

### 3.20 getRevInfo()

**Description:** the member function gets the versions of hardware and firmware for the device.

**Parameters:** None

**Return Value:** char *, a pointer to DLL internal memory null-terminated string that would be override after calling next function.
Return a string with hardware and firmware version
"error", means error occurred
"not licensed", means not licensed

### 3.21 getChannel()

**Description:** the member function gets the current channel number for array devices, return 1 if the current device is a single S500 device and is licensed.

**Parameters:** None

**Return Value:** int
Return a the channel num for array devices
1, if the device is a single S500 device and is licensed
0, means error occurred or not licensed

### 3.22 getBatteryPower()

**Description:** the member function gets the battery power for the device.

**Parameters:** None

**Return Value:** int
Return the battery power, in unit mV, as an integer
0, means error occurred or not licensed

### 3.23 getLastError()

**Description:** the member function returns a string which contains a date/timestamp and a description of error.

**Parameters:** None

**Return Value:** char *, a pointer to DLL internal memory null-terminated string that would be override after calling next function.

## 4. "C" Standard style interface

Following lists all the "C" standard style functions.

### 4.1 openMonitor ()

**Description:** Open the serial port. This function must be called before calling any other functions.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall openMonitor(char* port);*

**Parameters:** **1.** char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** 1, means serial port opened successfully
0, means error occurred.

### 4.2 setLicenseFilePath ()

**Description:** Set the directory in which license file located. This function should be called right after calling openMonitor()

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall setLicenseFilePath (char* port, char * pPath);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. char* pPath - an ANSI char string pointer that represents directory in which license files located.

**Return Value:** 1, means set directory successfully

0, means error occurred.

### 4.3 isLicensed ()

**Description:** Detect whether the current device is licensed or not.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall isLicensed (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** 1, means license.

0, means not opened.

### 4.4 closeMonitor ()

**Description:** Close the serial port. This function must be called for releasing allocated resource before exiting your program.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall closeMonitor (char* port);*

**Parameters:** **1.** char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** 1, means serial port closed successfully

0, means error occurred.

### 4.5 isOpen ()

**Description:** Detect whether the serial port is opened or not.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall isOpen(char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** 1, means opened.

0, means not opened.

### 4.6 flush ()

**Description:** Flush the buffer of the serial port, it is not necessary to call for most applications, since the buffer is flushed automatically

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall flush (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** 1, means the buffer has been flushed.

0, means error occurred.

### 4.7 getPower ()

**Description:** Get the power for the device

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getPower (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** double

1. return a double type number with the power reading for current measurement setting (dB or dBm)
2. return 100, means error occurred or not licensed
3. return 101, means having no power (below the lowest limited power

### 4.8 getSampleRate ()

**Description:** Get the sample rate for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) char __stdcall getSampleRate (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** char

'f' means fast mode

's' means slow mode

'e' means error occurred or not licensed

### 4.9 setSampleRate ()

**Description:** Set the sample rate for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall setSampleRate (char* port, char chRate);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. char chRate - 'f' is for fast mode; 's' is for slow mode

**Return Value:** 1, means command has been sent successfully

0, means error occurred.

### 4.10 getWavelength ()

**Description:** Get the wavelength for the device

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getWavelength (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** Not 0, return an integer number in nanometer describing wavelength.

0, means error occurred.

### 4.11 setWavelength ()

**Description:** Set the wavelength for the device

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall setWavelength (char* port, int iWavelength);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. int iWavelength - wavelength setting as integer, in nanometer

**Return Value:** 1, means the command was successfully sent to the device.

0, means error occurred.

### 4.12 getPolarization ()

**Description:** Get the current polarization mode for the device. It is only available for the device with Polarization Mode function, otherwise return 0.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getPolarization (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** int

1, means p1 mode

2,means p2 mode

0, means error occurred

### 4.13 setPolarization ()

**Description:** Set the polarization mode for the device. It is only available for the device with polarization mode function, otherwise return 0.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall setPolarization (char* port, int iPolMode);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. int iPolMode - 1 is for p1 mode; 2 is for p2 mode.

**Return Value:** 1, means command has been sent successfully

0, means error occurred.

### 4.14 getMode ()

**Description:** Get the mode for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) char __stdcall getMode (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** char

'r' means relative mode(dB)

'a' means absolute mode(dBm)

### 4.15 setMode ()

**Description:** Set the mode for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall setMode (char* port, char chRate);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. char chRate - 'r' is for relative mode(dB); 'a' is for absolute mode(dBm).

**Return Value:** 1, means command has been sent successfully

0, means error occurred.

### 4.16 getSerial ()

**Description:** Get the serial number for the device. This function is not under license control.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getSerial(char* port, char * pSN, int iSize);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. char* pSN – a pointer to buffer that receives the serial number

3. int iSize – maximum size of the buffer specified by pSN

**Return Value:** >0, if the function succeeds, the return value is the length of the string copid to pSN.

0, means error occurred.

### 4.17 getModel ()

**Description:** Get the model number for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getModel (char* port, char * pModel, int iSize);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. char* pModel – a pointer to buffer that receives the model number

3. int iSize – maximum size of the buffer specified by pModel

**Return Value:** >0, if the function succeeds, the return value is the length of the string copied to pModel.

0, means error occurred.

### 4.18 getRevInfo ()

**Description:** Get the version information of hardware and firmware for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getRevInfo (char* port, char * pRev, int iSize);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

2. char* pRev– a pointer to buffer that receives the model number

3. int iSize – maximum size of the buffer specified by pRev

**Return Value:** >0, if the function succeeds, the return value is the length of the string copied to pRev.

0, means error occurred.

### 4.19 getChannel ()

**Description:** gets the current channel number for array devices, return 1 if the current device is a single S500 device and is licensed.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getChannel (char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** int

1. Return a the channel num for array devices
2. if the device is a single S500 device and is licensed
3. 0, means error occurred or not licensed

### 4.20 getBatteryPower ()

**Description:** Get the battery power for the device.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getBatteryPower(char* port);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port. Example: "COM8".

**Return Value:** int

Return the battery power, in unit mV, as an integer

0, means error occurred or not licensed

### 4.21 getLastError ()

**Description:** Get information for the last error that occurred.

**C++ Porotype:** *extern "C" __declspec(dllimport) int __stdcall getLastError (char* port, char * pError, int iSize);*

**Parameters:** 1. char* port - an ANSI char string pointer that represents the serial port.

Example: "COM8".

2. char* pError– a pointer to buffer that receives the model number

3. int iSize – maximum size of the buffer specified by pError

**Return Value:** >0, if the function succeeds, the return value is the length of the string copied to pError.

0, means error occurred

## 5. VC++ 2010 Example

To use VC++ develop the client application, you can directly use the C++ interface exported by S500API_NEO.DLL. Following steps is for the reference.

1. "#include" the header file ("NEOPHOTONICSAPI.h").
2. Add the lib file ("C++API_Neo.lib") to the project.
3. Copy the DLL file ("C++API_Neo.dll") to the client application location

Following steps showing the function sequences for calling:

1. Create a PowerMonitor object in heap memory or stack memory, e.g,. PowerMonitor *pPM = new PowerMonitor("COM1");
2. Call member function setLicenseFilePath to specify the license file directory, in which the license files should be located.
3. Call other function to control the device.

## 6. VB.net 2010 Example

To use VB.net develop the client application. Following steps is for the reference.

1. Add the bas file(S500API_Neo.bas) to the project
2. Copy the DLL file ("C++API_Neo.dll") to the client application location.

Following steps showing the function sequences for calling:

1. Call function openMonitor to initiate the device
2. Call function setLicenseFilePath to specify the license file directory, in which the license files should be located.
3. Call other function to control the device
4. Remember to call closeMonitor to close all devices that have been opened by function openMonitor before exit the program for the relative resources release.