NSP32 SDK®
Reference Manual
for Python

ver 1.7

nanoLambda

IMPORTANT NOTICE

nanoLambda Korea and its affiliates (“nanoLambda”) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to nanoLambda’s terms and conditions of sale supplied at the time of order acknowledgment. Customers are responsible for their products and applications using any nanoLambda products. nanoLambda does not warrant or represent that any license, either express or implied, is granted under any nanoLambda patent right, copyright, mask work right, or other nanoLambda intellectual property right relating to any combination, machine, or process in which nanoLambda products or services are used. Information published by nanoLambda regarding third-party products or services does not constitute a license from nanoLambda to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from nanoLambda under the patents or other intellectual property of nanoLambda. Reproduction of nanoLambda information in nanoLambda documents or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. nanoLambda is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions. Resale of nanoLambda products is not allowed without written agreement. Decompiling, disassembling, reverse engineering or attempt to reconstruct, identify or discover any source code, underlying ideas, techniques or algorithms are not allowed by any means. nanoLambda products are not authorized for use in safety-critical applications. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of nanoLambda products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by nanoLambda. Further, buyers must fully indemnify nanoLambda and its representatives against any damages arising out of the use of nanoLambda products in such safety-critical applications.

Table of Contents

Functions Table

S.No	Description	Function Syntax
1	Initialization (DLL/SO)	[ret] = initialize (library_path);
2	Initialize Core API library (DLL/SO)	pSpecCore = initialize_core_api (library_path);
3	Initialize Device API library (DLL/SO)	pSpecDevice = initialize_device_api (library_path);
4	Close Core API Library	[ret]=close_core_api(pSpecCore);
5	Connect spectral sensor	[ret]=connect_device(pSpecDevice);
6	Disconnect spectral sensor	[ret]= disconnect_device(pSpecDevice);
7	Create Core Spectrum object	[ret]=create_core_object(pSpecCore);
8	Get sensor ID from device	[ret,sensor_ID] = get_sensor_id_device (pSpecDevice);
9	Get sensor ID from cal file	[ret,sensor_ID] = get_sensor_id_file (pSpecCore);
10	Get wavelength info. from cal file	[start_wavelength, end_wavelength, interval_wavelength] =get_wavelength_information(pSpecCore);
11	Get resolution	[resolution]=get_resolution(pSpecCore);
12	Get spectrum data size	[length]=get_spectrum_length(pSpecCore);
13	Get filter data	[FilterData]= get_filter_data(pSpecDevice, Frame_Average);
14	Activate a physical device sensor with sensor ID	[ret]= device_ID_activation (pSpecDevice,sensor_ID);
15	Activate a specific sensor with sensor ID from calibration File	[ret]=calibration_ID_activation(pSpecCore,sensor_ID);
16	Activate a device sensor with index	[ret]= index_activation(pSpecDevice, sensor_index);
17	Add one sensor cal data to the sensor cal data list	[ret]=load_sensor_file(pSpecCore,cal_file_path);
18	Get total number of sensors in data list	[ret] = get_capacity_sensor_data_list(pSpecCore);
19	Set background data	[ret]=set_background_data(pSpecCore,background_data);
20	Set sensor parameters to device	[ret]=set_sensor_parameters_to_device(pSpecDevice, adc_gain, adc_range);
21	Get sensor parameters from device	[adc_gain, adc_range] = get_sensor_parameters_from_device(pSpecDevice);
22	Get sensor parameters from cal data	[adc_gain, adc_range] = get_sensor_parameters_from_calibration_data(pSpecCore);
23	Set shutter speed to device	[ret]=set_shutter_speed (pSpecDevice,shutter_speed);
24	Get shutter speed from device	[shutter_speed]=get_shutter_speed (pSpecDevice);

25	Get Optimal shutter speed from device	[optimal_shutter_speed]= get_optimal_shutter_speed(pSpecDevice);
26	Get total num of filters	[total_fitlers] = get_num_of_filters(pSpecDevice);
27	Get Shutter Speed limits	[min_ss, max_ss] = get_shutter_speed_limits(pSpecDevice);
28	Calculate spectrum	[spec_data, wave_data] =calculate_spectrum (pSpecCore, raw_sensor_data, current_shutter_speed);
29	Exposure Time to Shutter Speed	[shutter_speed] = exposure_time_to_ss(pSpecDevice, master_clock, exposure_time_value);
30	Shutter Speed to Exposure Time	[exposure_time] = ss_to_exposure_time(pSpecDevice, master_clock, shutter_speed);
31	Total sensors connected	[sensors_connected]= total_sensors_connected(pSpecDevice);

Function Descriptions

Initialize

Syntax:

[ret] = initialize (library_path);

Parameters:

[0] library_path - the path of the Base library (DLL/SO) file.

Description:

Load base spectrum library(‘CrystalBase.dll/so’).

Returns:

[ret=1] true on success

[ret=-1] false on failure.

Example:

```
ret = initialize([cd '\CrystalBase.dll/so']);
```

Initialize Core API Library

Syntax:

pSpecCore= initialize_core_api (library_path);

Parameters:

[0] library_path - the path of the API library (DLL/SO) file.

Description:

Load Core Spectrum API library(‘CrystalCore.dll/so’) to calculate spectra and color information with raw sensor data from NSP32 spectral sensor. After load API library, creation for Core Spectrum object is required before using any of the other functions.

Returns:

[ret= pSpecCore - This is the handle you need to pass as an argument to call the functions defined in Core API] on success

[ret=-1] false on failure.

Example:

```
pSpecCore = initialize_core_api([cd '\CrystalCore.dll/so']);
```

Initialize Device API Library

Syntax:

pSpecDevice = initialize_device_api (library_path);

Parameters:

[0] library_path - the path of the API library (DLL/SO) file.

Description:

Load Device API library(‘CrystalPort.dll/so’) to control NSP32 spectral sensor. After load API library, initialization for NSP32 spectral sensor is required before using any of the other functions.

Returns:

[ret= pSpecDevice - This is the handle you need to pass as an argument to call the functions defined in Device API] on success

[ret=-1] false on failure.

Example:

```
pSpecDevice = initialize_device_api([cd '\CrystalPort.dll/so']);
```

Close Core API library

Syntax:

[ret] = close_core_library (pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

Close Core API

Returns:

[ret=1] true on success

[ret=-1] false on failure.

Example:

```
ret = close_core_api( pSpecCore);
```

Connect NSP32 Spectral Sensor

Syntax:

[ret] = connect_device (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

Connect to the physical device and returns the number of sensors connected.

Returns:

[ret=num of sensors connect] on success

[ret=-1] false on failure.

Example:

```
ret = connect_device (pSpecDevice);
```

Disconnect NSP32 Spectral Sensor

Syntax:

[ret] = disconnect_device(pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function close the instance of spectrometer.

Returns:

[ret=1] true on success

[ret=-1] false on failure.

Example:

```
ret = disconnect_device (pSpecDevice);
```

Create Core Spectrum Object

Syntax:

[ret] = create_core _object ();

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

The function will enable the verbos.

Returns:

[ret=1] true on success

[ret=-1] false on failure.

Example:

```
ret = create_core_object (pSpecCore);
```

Get Sensor ID from Device

Syntax:

[ret,sensor_ID] = get_sensor_id_device (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function returns the current spectrometer's ID(e.g. Y8457-2-33-77-0).

Returns:

[tuple=(ret,sensor_ID)] on success

[tuple=(-1, -1)] on failure

Example:

```
(ret,sensor_id)] = get_sensor_ID_device(pSpecDevice);
```

Get Sensor ID from Sensor Calibration Data

Syntax:

[ret, sensor_id] = get_sensor_id_file(pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

This function returns the current spectrometer's ID(e.g. Y8457-2-33-77-0).

Returns:

[tuple=(ret, sensor_ID)] on success

[tuple=(-1, -1)] on failure

Example:

```
(ret,sensor_id) = get_sensor_id_file(pSpecCore);
```

Get Wavelength Informatioin from Cal. Data

Syntax:

[start_wavelength, end_wavelength, interval] = get_wavelength_information (pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

This function returns the starting, ending and interval wavelength for respective spectrometer type.

Returns:

[tuple = (start_wavelength, end_wavelength, interval)] on success

[tuple = (-1,-1,-1)] on failure

Example:

```
(w_start, w_end, w_step) = get_wavelength_info_from_cal_data();
```

Get Resolution

Syntax:

[resolution,] = get_resolution (pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

Get resolution(number of points) for spectrum output.

Returns:

[ret=resolution] on success

[ret=-1] on failure

Example:

```
[resolution] = get_resolution (pSpecCore);
```

Get Spectrum Size (Length)

Syntax:

[length] = get_spectrum_length(pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

This function returns the size of spectrum output.

Returns:

[ret=spectrum length] on success

[ret=-1] on failure

Example:

```
[length] = get_spectrum_size (pSpecCore);
```

Get Filter Data

Syntax:

[filter_data] = get_filter_data(pSpecDevice, Frame_Average);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

[1] Frame Averages. (default is 20)

Description:

This function returns the filter data

Returns:

[ret=filter_data] on success

[ret=-1] on failure

Example:

```
[length] = get_spectrum_size (pSpecCore);
```

Activate Sensor with ID

Syntax:

[ret] = device_ID_activation (pSpecDevice, sensor_ID);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.
[1] Enter the spectrometer ID you want to use for current spectrum output.

Description:

This function is helpful when multiple sensors are connected and you want to use a specific sensor. Enter the sensor ID of the sensor and then that sensor will be used.

Returns:

[ret = 1] true on success
[ret = -1] false on failure.

Example:

```
ret = device_ID_activation(pSpecDevice, sensor_ID );
```

Activate Sensor with Index

Syntax:

[ret] = index_activation(pSpecDevice, sensor_index);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.
[1] Enter the index of the spectrometer index you want to use for current spectrum output. Default is 0.

Description:

This function is helpful when multiple sensors are connected and you want to use a specific sensor. Enter the index of the sensor and then that sensor will be used.

Returns:

[ret = 1] true on success
[ret = -1] false on failure.

Example:

```
ret = index_activation(pSpecDevice, sensor_index);
```

Activate Sensor with ID Using Cal File

Syntax:

[ret] = calibration_ID_activation (pSpecCore, sensor_ID);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

[1] Enter the spectrometer ID you want to use for current spectrum output.

Description:

This function is helpful when multiple sensors are connected and you want to use a specific sensor. Enter the sensor ID of the sensor and then that sensor will be used.

Returns:

[ret = 1] true on success

[ret = -1] false on failure.

Example:

```
ret = calibration_ID_activatoin(pSpecCore, sensor_ID );
```

Add One Sensor Cal Data To Data List

Syntax:

[ret] = load_sensor_file(pSpecCore, calibration_ file_path);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

[1] path to the calibration file.

Description:

This function loads the spectrometer's calibration data file.

Returns:

[ret = 1] true on success

[ret = -1] false on failure.

Example:

```
file_name = 'sensor_Y8457-2-33-77-0.dat';
ret = load_sensor_file (pSpecCore,file_name);
```

Get Total Num of Sensors in Data List

Syntax:

[ret] = get_capacity_sensor_data_list(pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

This function returns the total number of sensors present in sensor data list.

Returns:

[ret = total number of sensors in data list] on success

[ret = -1] false on failure.

Example:

```
ret = get_capacity_sensor_data (pSpecCore);
```

Set Background Data

Syntax:

[ret] = set_background_data (pSpecCore, background_sensor_data);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

[1] Filter data acquired at Shutter speed = 1.

Description:

This function set background data.

Returns:

[ret = 1] true on success

[ret = -1] false on failure.

Example:

```
ret = set_background_data (pSpecCore, background_sensor_data);
```

Set Sensor Parameters To Device

Syntax:

[ret] = set_sensor_parameters_to_device (pSpecDevice, adc_gain, adc_range);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

[1] ADC Gain

[2] ADC Range

Description:

This function sets the ADC Range and ADC Gain to the physical device.

Returns:

[ret = 1] true on success

[ret = -1] false on failure

Example:

```
adc_gain = 0;
adc_range = 132;
 ret = set_sensor_parameters (pSpecDevice, adc_gain,adc_range);
```

Get Sensor Parameters from Device

Syntax:

[adc_gain, adc_range] = get_sensor_parameters_from_device (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function returns the current settings of sensor registers from physical device (ADC gain and ADC range).

Returns:

[tuple = (adc_gain,adc_range)] on success

[tuple = (-1,-1)] on failure

Example:

```
(adc_gain,adc_range) = get_sensor_parameters_from_device(pSpecDevice);
```

Get Sensor Parameters from Sensor Calibration Data

Syntax:

[adc_gain, adc_range] = get_sensor_parameters_from_calibration_data (pSpecCore);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.

Description:

This function returns the current settings of sensor registers from calibration file (ADC gain and ADC range).

Returns:

[tuple = (adc_gain,adc_range)] on success

[tuple = (-1,-1)] on failure

Example:

```
adc_gain,adc_range) = get_sensor_parameters_from_calibration_data(pSpecCore);
```

Set Shutter Speed To Device

Syntax:

[ret] = set_shutter_speed (pSpecDevice, shutter_speed);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

[1] Shutter Speed value.

Description:

This function will set the current shutter speed for data acquisition.

Returns:

[ret=1] on success

[ret=-1] on failure.

Example:

```
[ret] = set_shutter_speed(pSpecDevice,shutter_speed);
```

Get Shutter Speed From Device

Syntax:

[shutter_speed] = get_shutter_speed (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function will return already set shutter speed.

Returns:

[ret=shutter speed] on success

[ret=-1] on failure.

Example:

```
[shutter_speed] = get_shutter_speed(pSpecDevice);
```

Get Optimal Shutter Speed From Device

Syntax:

[optimal_shutter_speed] = get_optimal_shutter_speed (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function will return optimal shutter speed from a specific illumination enviornment.

Returns:

[ret=optimal shutter speed] on success

[ret=-1] on failure.

Example:

```
[optimal_shutter_speed] = get_optimal_shutter_speed(pSpecDevice);
```

Get Total Number of Filters

Syntax:

[total_filters] = get_num_of_filters (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function will return total num of filters.

Returns:

[ret=total fitlers] on success

[ret=-1] on failure.

Example:

```
[total_fitlers] = get_num_of_fitlers(pSpecDevice);
```

Get Shutter Speed Limits

Syntax:

[min_shutter_speed, max_shutter_speed] = get_shutter_speed_limits (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function will return min and max shutter speed.

Returns:

[tuple = (min_shutter_speed, max_shutter_speed] on success

[tuple = (-1,-1)] on failure.

Example:

```
(min_shutter_speed, max_shutter_speed)= get_shutter_speed_limits(pSpecDevice);
```

Calculate Spectrum

Syntax:

[spec_data, wave_data] = calculate_spectrum_data(pSpecCore, raw_sensor_data, current_shutter_speed);

Parameters:

[0] pSpecCore - return of "Initialize Core API Library" function.
[1] raw_sensor_data – raw sensor data (size=1024)
[2] current_shutter_speed – current shutter speed for spectral sensor

Description:

This function will gives the values of the xyz.

Returns:

[tuple = (spec_data, wavelength_data)] on success
[tuple = (-1,-1)] on failure.

Example:

```
(spec_Data, wavelength_data) = calculate_spectrum_data(pSpecCore,
raw_sensor_data, current_shutter_speed);
```

Shutter Speed to Exposure Time

Syntax:

[exposure_time] = ss_to_exposure_time (pSpecDevice,master_clock, shutter_speed);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.
[1] master clock: the clock value of MCU
[2] Shutter speed: Current Shutter Speed

Description:

This function will convert shutter speed value of milliseconds.

Returns:

[ret = exposure_time in ms] on success
[ret = -1] on failure.

Example:

```
[ exposure_time ] = ss_to_exposure_time (pSpecDevice,master_clock, shutter_speed);
```

Exposure Time To Shutter Speed

Syntax:

[shutter_speed] = exposure_time_to_ss (pSpecDevice,master_clock, exposure_time);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

[1] master clock: the clock value of MCU

[2] Exposure_time: time in ms

Description:

This function will convert milliseconds time in shutter speed value.

Returns:

[ret = shutter_speed] on success

[ret = -1] on failure.

Example:

```
[ shutter_speed ] = exposure_time_to_ss (pSpecDevice,master_clock,exposure_time);
```

Total Sensors Connected

Syntax:

[total_sensors] = total_sensors_connected (pSpecDevice);

Parameters:

[0] pSpecDevice - return of "Initialize Device API Library" function.

Description:

This function will return the total number of sensors connected to the system.

Returns:

[ret = total sensors connected] on success

[ret = -1] on failure.

Example:

```
[ total_sensors ] = total_sensors_connected (pSpecDevice);
```

API Library Examples

'api_python_example.py'

```
% api_python_example.py
%
% Python example code for API
%
%   Copyright 2015- nanoLambda, Inc.
%   $Revision: 1.0.0.0 $  $Date: 2015/12/08 $
%
import sys
import csv
import ctypes
sys.path.append("..")
if(sys.version_info[0] < 3):
    from api_python2 import *
    from api_python2.core import *
    from api_python2.device import *
    from api_python2.color import *
    print ("*********************************************************************")
    print ("[Python-2]        Python Version : ", sys.version_info.major,
"." ,sys.version_info.minor , " Detected")
    print ("*********************************************************************")
else:
    from api_python3 import *
    from api_python3.core import *
    from api_python3.device import *
    from api_python3.color import *
    print ("*********************************************************************")
    print ("[Python-3]        Python Version : ", sys.version_info.major,
"." ,sys.version_info.minor , " Detected")
    print ("*********************************************************************")

#Initialization
if sys.platform == 'win32':
    initialize("..\Libs\CrystalBase.dll")
    pSpecCore      = initialize_core_api("..\Libs\CrystalCore.dll")
    pSpecDevice    = initialize_device_api("..\Libs/CrystalPort.dll")
else:
    initialize("../Libs/libCrystalBaseLight.so")
    pSpecCore      = initialize_core_api("../Libs/libCrystalCoreLight.so")
    pSpecDevice    = initialize_device_api("../Libs/libCrystalPortLight.so")

initialize_color_api(pSpecCore)


connectReturn   = connect_device(pSpecDevice)   # return total num of devices connected with
system

if connectReturn > 0:

    (ret, sensorID) = get_sensor_id_device(pSpecDevice)

    create_core_object(pSpecCore)
```

```
    if sys.platform == 'win32':
        csInit_Return = load_sensor_file(pSpecCore, b"..\config\sensor_" + sensorID + b".dat")
    else:
        csInit_Return = load_sensor_file(pSpecCore, b"../config/sensor_" + sensorID + b".dat")

    (ret, sensorID) = get_sensor_id_file(pSpecCore)

    get_sensor_parameters_from_device(pSpecDevice)

    (adcGain,adcRange) = get_sensor_parameters_from_calibration_file(pSpecCore)

    settingReturn = set_sensor_parameters_to_device(pSpecDevice,adcGain,adcRange)

    total_num_of_sensors = total_sensors_connected(pSpecDevice)

    get_capacity_sensor_data_list(pSpecCore)

    for index in range(total_num_of_sensors):

        #activate a specific device(sensor)
        activatingReturn = index_activation(pSpecDevice,index)

        #get sensor id of currently activated device(sensor)
        (ret, sensorID) = get_sensor_id_device(pSpecDevice)

        #get and set shutter speed of device(sensor)
        get_shutter_speed(pSpecDevice)
        set_shutter_speed(pSpecDevice,1)

        #get one filter output (sensor data)
        filterData = get_filter_data(pSpecDevice,20)

        #set background data
        set_background_data(pSpecCore,filterData)

        #get and set shutter speed of device(sensor)
        get_shutter_speed(pSpecDevice)

        #Get shutter speed with AE
        newSS = get_optimal_shutter_speed(pSpecDevice)
        set_shutter_speed(pSpecDevice,newSS)

        #convert shutter speed to exposure time (ms) for your reference
        ss_to_exposure_time(pSpecDevice,5,newSS)

        filterData = get_filter_data(pSpecDevice,20)

        specSize = get_spectrum_length(pSpecCore)
        (ret, specData,wavelengthdata) = calculate_spectrum_data(pSpecCore,filterData,newSS)

        (Start_Wavelength, End_Wavelength, Interval_Wavelength) =
get_wavelength_information(pSpecCore)

        get_resolution(pSpecCore)

        if sys.version_info[0] < 3:
            fileName = (r"SpecrtumData2_" + sensorID + ".csv");
            data = []
            for i in range(get_spectrum_length(pSpecCore)):
                data.append(str(specData[i]).split(","))
```

```
            with open(fileName, "wb") as csv_file:
                writer = csv.writer(csv_file, delimiter=',')
                for line in data:
                    writer.writerow(line)
            csv_file.close()
        else:
            fileName = (b"SpecrtumData3_" + sensorID + b".csv");
            data = []
            for i in range(get_spectrum_length(pSpecCore)):
                data.append(str(specData[i]).split(","))

            with open(fileName, 'w', newline='') as csvfile:
                filewriter = csv.writer(csvfile, delimiter=',',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
                for line in data:
                    filewriter.writerow(line)
            csvfile.close()
else:
    print ("**********************************************************************")
    print ("[PrismError]Device Not Connected. Please connect Device and try again.")
    print ("**********************************************************************")
close_color_api(pSpecCore)
close_core_object(pSpecCore)
disconnect_device(pSpecDevice)
```