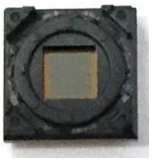
NSP32 SDK®
Reference Manual
for MATLAB



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.

Functions Table

Description	Function Syntax
Load Core API library (DLL)	[ret]=load_core_api_library (library_path);
Load Device API library (DLL)	[ret]=load_device_api_library (library_path);
Unload API library (DLL)	[ret]=unload_api_library(library_name);
Connect spectral sensor	[ret]=connect_device();
Disconnect spectral sensor	[ret]= disconnect_device();
Create Core Spectrum object	[ret]=create_core_spectrum_object();
Destroy Core Spectrum object	[ret]=destroy_core_spectrum_object();
Get sensor ID from device	[ret,spectrometer_id, sensor_count]=get_sensor_ID_from_device ();
Get sensor ID from cal data	[ret,spectrometer_id,sensor_count]=get_sensor_ID_from_cal_data ();
Get wavelength info. from cal data	[ret,start_wavelength, end_wavelength, interval_wavelength] =get_wavelength_info_from_cal_data();
Get resolution	[ret,resolution]=get_resolution();
Get spectrum data size	[ret,length]=get_spectrum_size();
Activate a specific sensor with sensor ID	[ret]=activate_sensor_with_ID(sensor_ID);
Add one sensor cal data to the sensor cal data list	[ret]=add_to_sensor_data_list(cal_file_path);
Acquire raw sensor data	[ret,filter_data]=acquire_sensor_data(num_of_averages);
Set background data	[ret]=set_background_data(background_data);
Set sensor parameters to device	[ret]=set_sensor_parameters_to_device(adc_gain, adc_range);
Get sensor parameters from device	[ret,adc_gain, adc_range]=get_sensor_parameters_from_device();
Get sensor parameters from cal data	[ret,adc_gain, adc_range]=get_sensor_parameters_from_cal_data();
Set shutter speed to device	[ret]=set_shutter_speed_to_device(shutter_speed);
Get shutter speed from device	[ret,shutter_speed]=get_shutter_speed_from_device();
Find optimal shutter speed with AE	[ret,shutter_speed]=find_optimal_shutter_speed ();
Calculate spectrum	[ret,spec_data, wave_data] =calculate_spectrum(raw_sensor_data, current_shutter_speed);
Calculate color	[ret,X,Y,Z,R,G,B,x,y,z,cct]=calculate_color(spec_data,wave_data);

Function Descriptions

Load Core API Library

Syntax:

[ret] = load_core_api_library (library_path);

Parameters:

library_path – The path of the API library (DLL) file.

Description:

Load Core Spectrum API library('CrystalCore.dll') 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=1] true on success
[ret=0] API library didn't be loaded
[ret=-1] false on failure.

Example:

```
ret = load_core_api_library([cd '\CrystalCore.dll']);
```

Load Device API Library

Syntax:

[ret] = load_device_api_library (library_path);

Parameters:

library_path – The path of the API library (DLL) file.

Description:

Load Device API library('CrystalPort.dll') 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=1] true on success
[ret=0] API library didn't be loaded
[ret=-1] false on failure.

Example:

```
ret = load_device_api_library([cd '\CrystalPort.dll']);
```

Unload API Library

Syntax:

[ret] = unload_api_library (library_name);

Parameters:

library_name – The name of the API library (DLL) file.

Description:

Unload API libraries('CrystalCore.dll' and 'CrystalPort.dll').

Returns:

[ret=1] true on success

[ret=0] API library didn't be loaded

[ret=-1] false on failure.

Example:

```
[ret] = unload_api_library('CrystalCore');
[ret] = unload_api_library('CrystalPort');
```

Connect NSP32 Spectral Sensor

Syntax:

[ret] = connect_device ();

Parameters:

void

Description:

Before you start control your spectral sensor, you must connect one or more spectral sensors to your system via USB connection. Currently, USB connection is only a way to build a connection to your spectral sensor.

Returns:

[ret=1] true on success

[ret=0] API library didn't be loaded

[ret=-1] false on failure.

Example:

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

Disconnect NSP32 Spectral Sensor

Syntax:

[ret] = disconnect_device();

Parameters:

void

Description:

You can disconnect your spectral sensor with `duDisconnect()` function which releases all connections to sensors, destroys the sensor interface object, and free allocated system

resources internally.

Returns:

[ret=1] true on success

[ret=-1] false on failure.

Example:

```
[ret] = disconnect_device();
```

Create Core Spectrum Object

Syntax:

[ret] = create_core_spectrum_object ();

Parameters:

void.

Description:

For spectra calculations, spectral characteristics of a particular spectral sensor, and information on sensor parameters (ADC gain and ADC range), you must first create a Core Spectrum object. The core goal of the Core Spectrum object is to calculate the spectra from the raw sensor data obtained from the NSP32 spectral sensor. The calibration data specific to each spectral sensor is involved in spectra's calculations.

Returns:

[ret=1] true on success

[ret=0] API library didn’t be loaded

[ret=-1] false on failure.

Example:

```
ret = create_core_spectrum_object();
```

Destroy Core Spectrum Object

Syntax:

[ret] = destroy_core_spectrum_object ();

Parameters:

void.

Description:

When your MATLAB application is ready to terminate, call this function to destroy Core Spectrum object.

Returns:

[ret=1] true on success

[ret=0] API library didn’t be loaded

[ret=-1] false on failure.

Example:

```
[ret] = destroy_core_spectrum_object();
```

Get Sensor ID from Device

Syntax:

[ret,sensor_id, sensor_count] = get_sensor_ID_from_device ( );

Parameters:

void

Description:

If you wish to identify which spectral sensor is currently activated and you are communicating with, there is a wrapper function that helps identify the spectral sensor by returning sensor's unique ID. You can get a sensor ID from physical spectral sensor with this function (e.g. Y8585-1-85-85-0).

Returns:

[sensor_id] sensor ID of currently activated sensor (string)
[sensor_count] total number of sensors in your system
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure

Example:

```
[ret, sensor_id] = get_sensor_ID_from_device();
```

Get Sensor ID from Sensor Calibration Data

Syntax:

[ret,sensor_id, sensor_count] = get_sensor_ID_from_cal_data( );

Parameters:

void

Description:

If you wish to identify which spectral sensor is currently activated and you are communicating with, there is a wrapper function that helps identify the spectral sensor by returning sensor's unique ID. You can get a sensor ID from spectral sensor's calibration file with this function. You must always keep in mind that the physical sensor ID and the ID from the calibration file must be matched before start spectrum calculations (e.g. Y8585-1-85-85-0).

Returns:

[sensor_id] sensor ID of currently activated sensor (string)
[sensor_count] total number of sensors in the sensor data list
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure

Example:

```
[ret,sensor_id] = get_sensor_ID_from_cal_data();
```

Get Wavelength Informatioin from Cal. Data

Syntax:

[ret, start_wavelength, end_wavelength, interval] = get_wavelength_info_from_cal_data ();

Parameters:

void

Description:

Four different types of NSP32 spectral sensors (VIS, NIR1, NIR2 and FULL) of the nanoLambda are shipped to the user with their own calibration data file, each calibrated for a different target wavelength range. Therefore, you can use this function to retrieve information about the target wavelength range and resolution of your spectral sensor.

Returns:

[start_wavelength] start wavelength
[end_wavelength] end wavelength
[interval] wavelength interval
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure

Example:

```
[ret,w_start,w_end,w_step] = get_wavelength_info_from_cal_data();
```

Get Resolution from Device

Syntax:

[ret, resolution] = get_resolution ();

Parameters:

void

Description:

Get resolution information of a NSP32 spectral sensor.

Returns:

[resolution] resolution of spectral sensor
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure

Example:

```
[ret,resolution] = get_resolution ();
```

Get Spectrum Size (Length)

Syntax:

[ret,length] =get_spectrum_size();

Parameters:

void

Description:

This function returns the size of spectrum output. You need to get this information before you try to calculate the spectra because you must allocate memory to store spectra.

Returns:

[length] spectrum length (size)
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure

Example:

```
[ret,length] = get_spectrum_size();
```

Activate Sensor with ID

Syntax:

[ret] = activate_sensor_with_ID(sensor_ID);

Parameters:

sensor_ID – Sensor ID what you want to activate it to acquire sensor data and calculate spectrum and wavelength data.

Description:

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

Returns:

[ret=1] true on success
[ret=0] API library didn't be loaded
[ret=-1] false on failure.

Example:

```
sensor_ID = 'Y8585-1-85-85-1';
[ret] = activate_sensor_with_ID( sensor_ID );
```

Add One Sensor Cal Data To Data List

Syntax:

[ret] = add_to_sensor_data_list(calibration_ file_path);

Parameters:

calibration_ file_path – File path for the sensor calibration file.

Description:

Once Core Spectrum object is created successfully, next, you must add one sensor calibration data to sensor data list by providing a sensor calibration data file path as an argument to this function. If you have a return value larger than 0, then this addition is succeeded and from now on, you can use this sensor for your MATLAB application by activating it with

`activate_sensor_with_ID()` function.

Returns:

[ret=1] true on success
[ret=0] API library didn't be loaded
[ret=-1] false on failure.

Example:

```
cal_file_name = 'sensor_Y8457-2-33-77-0.dat';
[ret] = add_to_sensor_data_list(cal_file_name);
```

Acquire Raw Sensor Data

Syntax:

[ret,filter_data] = acquire_sensor_data (num_of_averages);

Parameters:

num_of_averages – the number of frames for averaging.

Description:

This function returns a raw sensor data under acquisition conditions (shutter speed, frame count for averaging, and so on) what you are specified a priori.

Returns:

[ret=1] true on success
[ret=0] API library didn't be loaded
[ret=-1] false on failure.

Example:

```
set_shutter_speed(1);
num_of_averages = 50;
[ret,filter_data]=acquire_sensor_data(num_of_averages);
```

Set Background Data

Syntax:

[ret] = set_background_data (background_sensor_data);

Parameters:

background_sensor_data – one raw sensor data will be used as background data.

Description:

You can use this function to set background data before you try to calculate spectra of interest. To do this, you need to change current shutter speed to '1' and acquire a raw sensor data. The purpose of background data is to compensate sensor's dark current and ambient noise.

Returns:

[ret=1] true on success
[ret=0] API library didn't be loaded
[ret=-1] false on failure.

Example:

```
set_shutter_speed(1);
```

```
num_of_averages = 50;
[ret,filter_data]=acquire_sensor_data(num_of_averages);
ret = set_background_data(filter_data);
```

Set Sensor Parameters To Device

Syntax:

[ret] = set_sensor_parameters_to_device (adc_gain, adc_range);

Parameters:

adc_ gain – ADC gain value (0 or 1)

adc_range – ADC range value(0<=range<=255).

Description:

This function sets the sensor parameters (ADC gain and range values) for acquiring raw sensor data and calculating spectra. Sensor parameters of a NSP32 spectral sensor must be matched with sensor parameters in sensor calibration data.

Returns:

[ret=1] true on success

[ret=0] API library didn't be loaded

[ret=-1] false on failure (invalid argument range or failure on function call).

Example:

```
adc_gain = 1;     % 1=1x, 0=4x
adc_range = 132;  % 0 <= ADC range <= 255
[ret] = set_sensor_parameters_to_device(adc_gain, adc_range);
```

Get Sensor Parameters from Device

Syntax:

[ret,adc_gain, adc_range] = get_sensor_parameters_from_device ();

Parameters:

void

Description:

This function is used to get the sensor parameters from your NSP32 spectral sensor.

Returns:

[adc_gain] ADC gain (0 or 1)

[adc_range] ADC range (0<=range<=255)

[ret=0] API library didn't be loaded

[ret=-1] Power too high. Spectrometer is saturated.

[ret=-2]Power too low, spectrometer response is not correct.

Example:

```
[ret,adc_gain,adc_range] = get_sensor_parameters_from_device();
```

Get Sensor Parameters from Sensor Calibration Data

Syntax:

[ret,adc_gain, adc_range] = get_sensor_parameters_from_cal_data ();

Parameters:

void.

Description:

This function is used to get the sensor parameters from sensor calibration data

Returns:

[adc_gain] ADC gain (0 or 1)

[adc_range] ADC range (0<=range<=255)

[ret=0] API library didn't be loaded

[ret=-1] Power too high. Spectrometer is saturated.

[ret=-2]Power too low, spectrometer response is not correct.

Example:

```
[ret,adc_gain,adc_range] = get_sensor_parameters_from_cal_data();
```

Set Shutter Speed To Device

Syntax:

[ret] = set_shutter_speed_to_device (shutter_speed);

Parameters:

shutter_speed – shutter speed(exposure time) for currently activated spectral sensor.

Description:

This function will set new shutter speed to your NSP32 spectral sensor.

Returns:

[r, g, b] CIE RGB tristimulus color values

[ret=1] on success

[ret=0] API library didn't be loaded

[ret=-1] on failure.

Example:

```
shutter_speed = 50;
[ret] = set_shutter_speed_to_device(shutter_speed);
```

Get Shutter Speed From Device

Syntax:

[ret, shutter_speed] = get_shutter_speed_from_device ();

Parameters:

void.

Description:

This function will return a current shutter speed from NSP32 spectral sensor.

Returns:

[shutter_speed] current shutter speed for spectral sensor.

[ret=1] on success

[ret=0] API library didn't be loaded
[ret=-1] on failure.

Example:

```
[ret,shutter_speed] = get_shutter_speed_from_device();
```

Find Optimal Shutter Speed with AE (Auto-Exposure)

Syntax:

[ret, shutter_speed] = find_optimal_shutter_speed ();

Parameters:

void.

Description:

This function will return a optimal shutter speed found by AE (Auto-Exposure) function.

Returns:

[shutter_speed] optimal shutter.
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure.

Example:

```
[ret,shutter_speed] = find_optimal_shutter_speed ();
```

Calculate Spectrum

Syntax:

[ret, spec_data, wave_data] = calculate_spectrum(raw_sensor_data, current_shutter_speed);

Parameters:

raw_sensor_data – raw sensor data (size=1024)
current_shutter_speed – current shutter speed for currently activated spectral sensor

Description:

This function will calculate the spectra with a raw sensor data and current shutter speed information, and returns spectra and wavelength data.

Returns:

[spec_data] spectrum data
[wave_data] wavelength data
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure.

Example:

```
shutter_speed = 100;
frame_averages = 50;
[ret] = set_shutter_speed_to_device(shutter_speed);
[ret,sensor_data]=acquire_sensor_data(frame_averages);
[ret,spec_data,wave_data] = calculate_spectrum(sensor_data,shutter_speed);
```

Calculate Color

Syntax:

[ret, X,Y,Z,R,G,B,x,y,z,cct] = calculate_color(spectrum_data, wavelength_data);

Parameters:

spectrum_data – spectrum data to calculate color

wavelength_data –wavelength data to calculate color(length is same with 'spectrum_data')

Description:

This function will calculate several color information with input spectra and wavelength data and returns to you.

Returns:

[X] X component in CIE XYZ color space
[Y] Y component in CIE XYZ color space
[Z] Z component in CIE XYZ color space
[R] R component in RGB color space (sRGB, D65)
[G] G component in RGB color space (sRGB, D65)
[B] B component in RGB color space (sRGB, D65)
[x] x component calculated from CIE XYZ color space
[y] y component calculated from CIE XYZ color space
[z] z component calculated from CIE XYZ color space
[cct] Correlated Color Temperature (CCT)
[ret=1] on success
[ret=0] API library didn't be loaded
[ret=-1] on failure.

Example:

```
shutter_speed = 100;
frame_averages = 50;
[ret] = set_shutter_speed_to_device(shutter_speed);
[ret,sensor_data]=acquire_sensor_data(frame_averages);
[ret,spec_data,wave_data] = calculate_spectrum(sensor_data,shutter_speed);
[X,Y,Z,R,G,B,x,y,z,cct] = calculate_color(spec_data, wave_data);
```

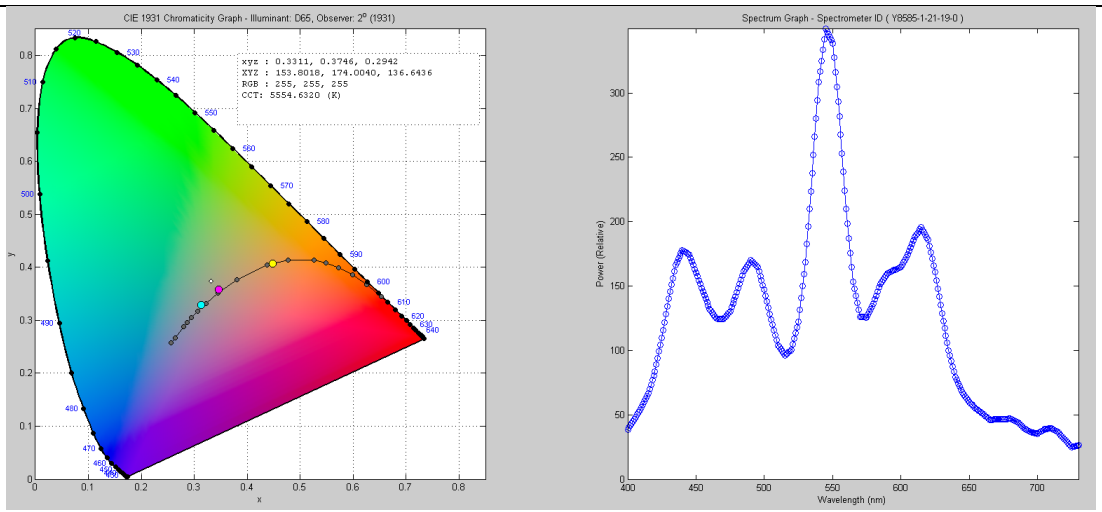
API Library Examples

'spectrometer_example.m'

```
% spectrometer_example.m
%
% MATLAB example code for APIs
%
%   Copyright 2016- nanoLambda, Inc.
%   $Revision: 1.7.0.0 $  $Date: 2016/12/30 $
%
close all;
clear all;
clc;

LIBRARY_PATH = '\lib_win32\';
CORE_LIBRARY_NAME = 'CrystalCore.dll';
DEVICE_LIBRARY_NAME = 'CrystalPort.dll';

path(path,[cd , '/api']);
path(path,[cd , LIBRARY_PATH]);

% load API DLLs
[ret_device] = load_device_api_library([cd LIBRARY_PATH DEVICE_LIBRARY_NAME]);
[ret_core] = load_core_api_library([cd LIBRARY_PATH CORE_LIBRARY_NAME]);

if( ret_core ~= 1 || ret_device ~= 1 )
    disp('[Error] Fail to load API libraries.');    return;
end
%%
% initialize Device object.
[ret] = connect_device();
if( ret < 0 )
    disp('[Error] No sensor in your system.');    return;
end

% get sensor ID of physical spectral sensor
[SENSOR_ID, total_count] = get_sensor_ID_from_device();
disp(['sensor ID from device: ' SENSOR_ID]);

% create Core Spectrum object.
[ret] = create_core_spectrum_object();

% add one calibration file path to Core Spectrum object
sensor_cal_file_path = [cd '\sensor_' SENSOR_ID '.dat'];
[ret] = add_to_sensor_data_list(sensor_cal_file_path);
if ret == -1
    disp('Fail to initialize API.');    return;
end

% get spectral sensor ID from calibration data
[SENSOR_ID2, total_count2] = get_sensor_ID_from_cal_data();
disp(['sensor ID from cal data: ' SENSOR_ID2]);

% get sensor parameters (ADC gain and range) from calibration data
[adc_gain, adc_range] = get_sensor_parameters_from_cal_data();

% get wavelength info.
[START_WAVELENGTH, END_WAVELENGTH, WAVELENGTH_STEP, ret] = get_wavelength_info_from_cal_data();
WAVELENGT_RANGE = (START_WAVELENGTH:1:END_WAVELENGTH);
WAVELENGTH_STEP = 1;
```

```
% activate a specific sensor with sensor ID
activate_sensor_with_ID(SENSOR_ID2);

% set sensor parameters to physical spectral sensor.
[ret] = set_sensor_parameters_to_device(adc_gain, adc_range);

FRAME_AVERAGE = 10;
% get background data at ss=1
set_shutter_speed_to_device(1);
[ret,background_data] = acquire_sensor_data(FRAME_AVERAGE);

% set background to Core Spectrum object.
[ret] = set_background_data(background_data);

SHUTTER_SPEED = 50;
set_shutter_speed_to_device(SHUTTER_SPEED);

% set acquisition conditions to acquire spectrum data.
if ret ~= -1
    figure;
    for frame_i=1:10
        % acquire raw sensor data from spectral sensor
        [ret,filter_data] = acquire_sensor_data(FRAME_AVERAGE);
         if( isempty(filter_data) == 0 )
             % calculate spectrum by Core Spectrum API
            [spec_data, wavelength_data, ret] = calculate_spectrum(filter_data, SHUTTER_SPEED);
             % just check
            if length(spec_data) == 0
                continue;
            end
             % calculate color with spectrum and wavelength data
            [X, Y, Z, R, G, B, x, y, z, cct] = calculate_color(spec_data, wavelength_data);
             % plot it
            subplot(1,2,1);
            plot(filter_data);
            xlabel('Filter Index');
            ylabel('Power (Relative)');
            title('Raw Sensor Data Graph');
            axis([-Inf Inf 0 4096]);
            grid on;

            subplot(1,2,2);
            plot(wavelength_data, spec_data);
            xlabel('Wavelength (nm)');
            ylabel('Power (Relative)');
            title(sprintf('Spectrum
Graph\nXYZ=(%.3f,%.3f,%.3f\nRGB=(%d,%d,%d),CCT=%.3f',X,Y,Z,R,G,B,cct));
            axis([-Inf Inf 0 Inf]);
            grid on;
            drawnow;
        end

        disp(sprintf('%d-th run for spectrum calculation.', frame_i));
    end
end

% disconnect device API
[ret] = disconnect_device();

% destroy APIs (core spectrum and color application)
[ret] = destroy_core_spectrum_object();
[ret] = unload_api_library('CrystalCore');
```

(2) 'spectrometer_example_color.m'



(3) 'spectrometer_example_gui.m'