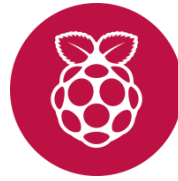


NSP32 SDK

Installation Manual

on **Raspberry Pi**



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. This development kit is only for the purpose of performance evaluation and application development. nanoLambda assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using any components of the development kit. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards. 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 data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. nanoLambda is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions. Resale of nanoLambda development kit and any of its components is not allowed. 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 whatever. nanoLambda products are not authorized for use in safety-critical applications (such as life support) where a failure of the nanoLambda product would reasonably be expected to cause severe personal injury or death. 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.

Application Programming Interface (API) For NSP32 Spectral Sensor

NSP32 application programming interface (API) is a list of all classes that are part of the NSP32 Software Development Kit (SDK). It includes all libraries, classes, and interfaces, along with their methods, fields, and constructors. These prewritten classes provide a tremendous amount of functionality to a programmer. A programmer should be aware of these classes and should know how to use them. If you browse through the list of packages in the API, you will observe that there are packages written for reading data from NSP32 spectral sensor, Connecting single or multiple sensors, managing input and output, getting spectrum data for single or multiple sensors, and many more. Please browse this manual for complete list of available functions and their descriptions to see how they can be used.

Table of Contents

Introduction	4
Setup Raspberry-Pi Device for NSP32 Data Acquisition.....	5
Download and Install OS Image.....	5
Preliminary Settings On Raspberry-Pi	5
Download and Install 3 rd Parties Libraries On Raspberry-Pi	5
Downloading libusb-1.0.9 and libusb-compat-0.1.4	5
Installing libusb-1.0.9	6
Installing libusb-compat-1.0.4	6
Installing gsl-1.13 on Raspberry	6
How To Build and Run Examples	7
Setup for Cross Compile On Ubuntu Host	7
You need to setup your host (Ubuntu) to do cross-compile your example or your application program with pre-built static libraries in NSP32 SDK.....	7
Install the ARM-based gcc/g++ tool chain	7
Install libusb-1.0.9	7
Install libusb-compat-1.0.4	7
Install gsl-1.13.....	8
Extracting the Package	8
Build and Run C/C++ Example.....	11
Run Python Example.....	13
Run on Windows (Win32).....	13
Run on Raspbian-Jessie or Linux(Ubuntu)	14

Introduction

This user manual is formulated to briefly explain how to setup the working environment for collecting the data from the NSP32 spectral sensor.

To use NSP32 spectral sensor for your spectral sensing application, you need to install few pre-requisite libraries for measurement and/or experimentation. The development environment is tested on full desktop image based on Rasbian-Jessie on Raspberry Pi 2 and 3 Model B (v1.1).

There are two ways to work with Raspberry Pi:

1. You can develop all the applications on Raspberry Pi (Device)
2. You can do all the development on Host machine (e.g., Ubuntu OS) and then transfer the application to the device using secure shell.

NSP32 SDK(Crystal libraries) provided by nanoLambda can be used for both ways because nanoLambda provides pre-built static libraries for you. Beside NSP32 SDK(Crystal libraries), you need to install three 3rd party libraries also.

In this manual, we will explain briefly

1. how to preliminary configure your Raspberry-Pi device or your Host machine
2. how you can install these 3rd party libraries on both host and device
3. how to configure your IDE(e.g., NetBeans) for application development
4. how to build your example or application on device or host
5. how to test/run your C/C++ and/or Python example or application on device or host.

Setup Raspberry-Pi Device for NSP32 Data Acquisition

Download and Install OS Image

1. Download the Raspbian-Jessie (any release) image from the below mentioned link
 - <https://www.raspberrypi.org/downloads/raspbian/>
2. Install Raspbian-Jessie image on RaspberryPi device (refer to the link below):
 - <https://www.raspberrypi.org/documentation/installation/installing-images/README.md>

Preliminary Settings On Raspberry-Pi

After installing the operating system image on the Raspberry, turn it on, open the terminal and run few preliminary commands to update and upgrade your Raspbian-Jessie OS:

1. `$ sudo apt-get update`
2. `$ sudo apt-get upgrade`
3. If it will give error that not enough space (Raspberry pi 2)
 - a. In a console, enter a command below:
 - i. `sudo raspi-config`
 - b. The menu, which appears has as first menu item:
Expand Filesystem Ensures that all of the SD card storage is available to the OS
 - c. Use it and you are done.
4. `$ sudo apt-get install build-essential`

Download and Install 3rd Parties Libraries On Raspberry-Pi

Install prerequisites for the libusb. To use NSP32 spectral sensor for measurement and/or experimentation, you have to install USB driver on the RaspberryPi:

1. `$ sudo apt-get install libudev-dev`
2. `$ sudo apt-get install pkg-config`

Downloading libusb-1.0.9 and libusb-compat-0.1.4

Download the .tar.gz for libusb-1.0.9 and libusb-compat-0.1.4 from links below:

- **libusb-1.0.9:**
<http://sourceforge.net/projects/libusb/files/libusb-1.0/libusb-1.0.9/>
- **libusb-compat-0.1.4:**
<http://sourceforge.net/projects/libusb/files/libusb-compat-0.1/libusb-compat-0.1.4/>

And run commands below:

1. `$ tar xvf ~/Downloads/libusb-1.0.9.tar.bz2`
2. `$ tar xvf ~/Downloads/libusb-compat-0.1.4.tar.bz2`

Installing libusb-1.0.9

To install libusb library, run commands below:

1. `$ cd libusb-1.0.9/`
2. `$ sudo ./configure`
3. `$ sudo make install`

Checking whether the install was successful or not

```
$ ls /usr/local/lib | grep libusb
```

Installing libusb-compat-1.0.4

To install libusb-compat library, run commands below:

1. `$ cd libusb-compat-1.0.4/`
2. `$ sudo ./configure`
3. `$ sudo make install`

Checking whether the install was successful or not

```
$ ls /usr/local/lib | grep libusb
```

Installing gsl-1.13 on Raspberry

Download gsl-1.13 from this link

<http://public.p-knowledge.co.jp/gnu-mirror/gsl/>

Run commands below to install gsl(GNU Scientific Library):

1. `cd gsl-1.13`
2. `$ sudo ./configure`
3. `$ sudo make`
4. `$ sudo make install`
5. `$ sudo ldconfig`

How To Build and Run Examples

Setup for Cross Compile On Ubuntu Host

You need to setup your host (Ubuntu) to do cross-compile your example or your application program with pre-built static libraries in NSP32 SDK.

Install the ARM-based gcc/g++ tool chain

Run commands to install tool chain on your host:

1. `$ sudo apt-get install gcc-arm-linux-gnueabi`
2. `$ sudo apt-get install g++-arm-linux-gnueabi`
3. `$ sudo ldconfig`

Test if the installation of tool chain was successful or not:

```
$ arm-linux-gnueabi-gcc
```

You should see executed results in your console window like below because you did try to run a cross-compiler (gcc) without any source code:

```
arm-linux-gnueabi-gcc: fatal error: no input files
compilation terminated.
```

Install libusb-1.0.9

Run commands to install libusb-1.0.9 on your host:

1. `$ cd libusb-1.0.9/`
2. `$ sudo ./configure --build=/path/to/arm-linux-gnueabihf --prefix=/path/to/save`
3. `$ sudo make install`

Check whether the install was successful or not by running a command below:

```
$ ls /usr/local/lib | grep libusb
```

Install libusb-compat-1.0.4

Run commands to install libusb-compat-1.0.4 on your host:

1. `$ cd libusb-compat-1.0.4/`
2. `$ sudo ./configure --build=/path/to/arm-linux-gnueabihf --prefix=/path/to/save`

Note

Sometimes libusb-compat can't find the installed libusb-1.0.

In that case we need to set PKG_CONFIG_PATH .

1. \$ export PKG_CONFIG_PATH=/installed/libusb/pkgconfig
2. echo \$PKG_CONFIG_PATH
3. \$ sudo make install

Checking whether the install was successful or not

```
$ ls /usr/local/lib | grep libusb
```

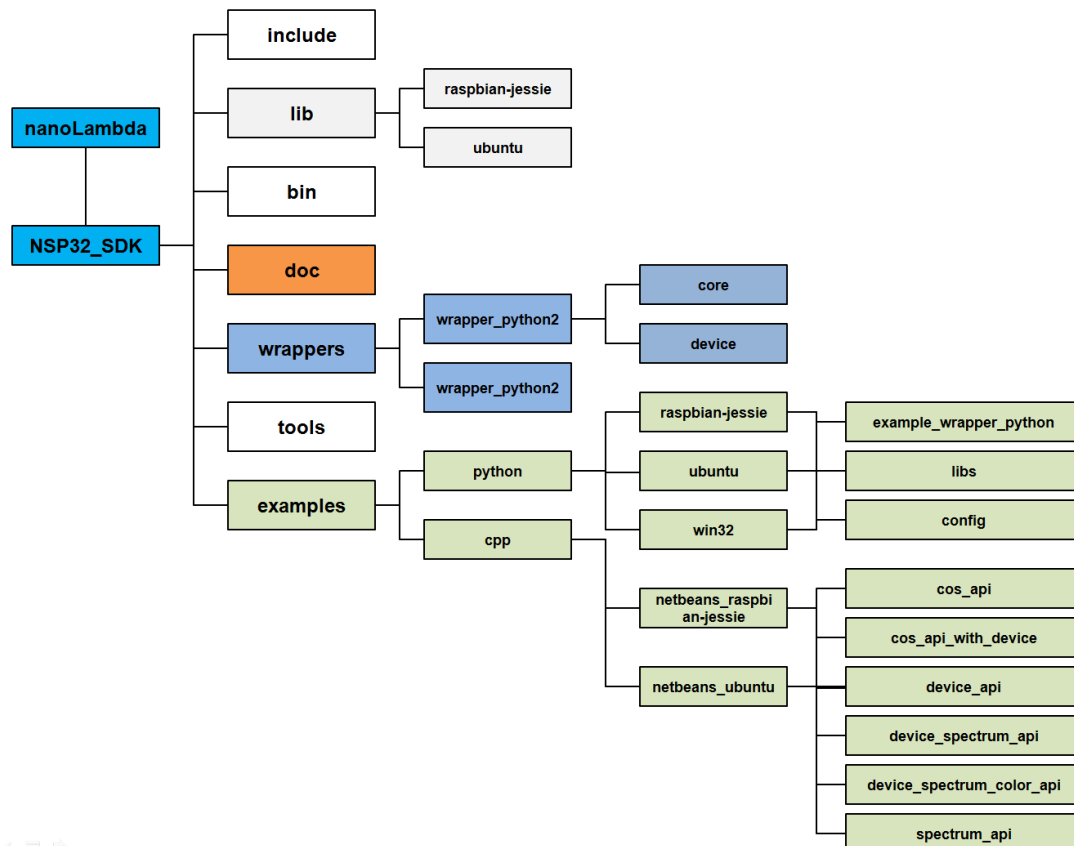
Install gsl-1.13

Download gsl-1.13 from this link : <http://public.p-knowledge.co.jp/gnu-mirror/gsl/>

Run commands to install gsl (GNU Scientific Library) on your host:

1. \$ cd gsl-1.13
2. \$ sudo ./configure --build=/path/to/arm-linux-gnueabi --prefix=/path/to/save
3. \$ sudo make
4. \$ sudo make install
5. \$ sudo ldconfig

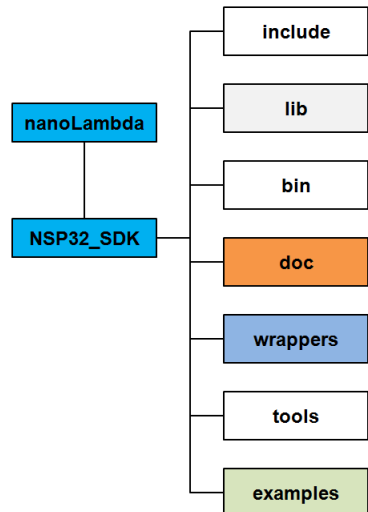
Install NSP32 SDK



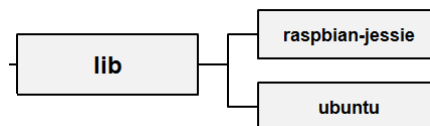
The Package is compressed in .tar.gz format. Double click the 'NSP32_SDK_1_7_installer_ubuntu.tar.gz' file or you can uncompress it by using the command

```
- $ tar xvf NSP32_SDK_1_7_package.tar.gz
```

NSP32_SDK folder contains sub-directories.

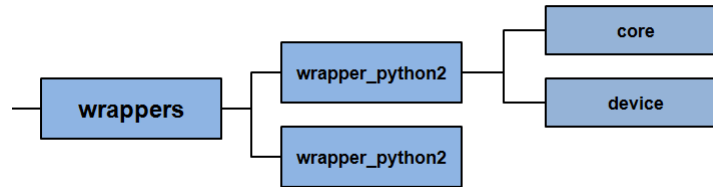


- "include" folder contains all the header files, you need to run the example code.
- "lib" folder contains all the static libraries. "lib" folder has sub-directories which in-turn has all libraries for specific platforms like Raspbian-jessie and Ubuntu OSs.

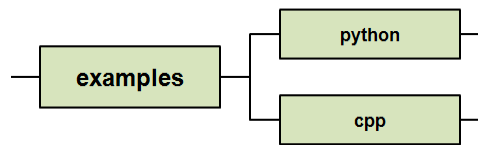


- "doc" folder contains all the documents. Right now there are 3 manuals (PDF formats). Two are manuals for installation and running example codes for RaspberryPi and Ubuntu, respectively. The other one is Python API reference manual.
 1. NSP32 SDK Installation Manual for Ubuntu-v1.7.pdf
 2. NSP32 SDK Installation Manual for RaspberryPi-v1.7.pdf
 3. NSP32 SDK Reference Manual for Python-v1.7.pdf
- "wrapper" folder contains wrapper for python. nanoLambda NSP32 SDK supports Python language with Python wrappers for both version of Python 2.xx and 3.xx.
- Under 'NSP32_SDK/wrappers/Python' folder, you can find two separate wrapper folders, "wrapper_Python2" and "wrapper_Python3". In each folder, there are 2 sub-directories,

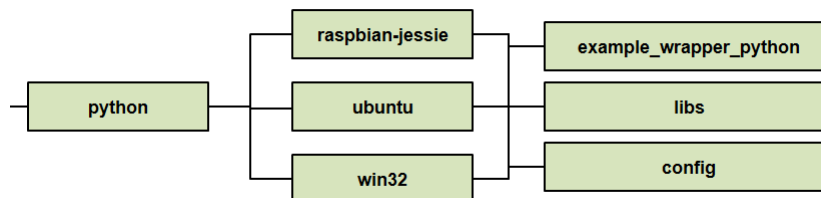
"core" and "device". "core" folder includes all the functions related to the core spectrum of the spectral sensor and "device" folder includes all the functions related to NSP32 device interface. All the codes are written in C/C++ and they are just wrapped with an additional layer to provide Python interface.



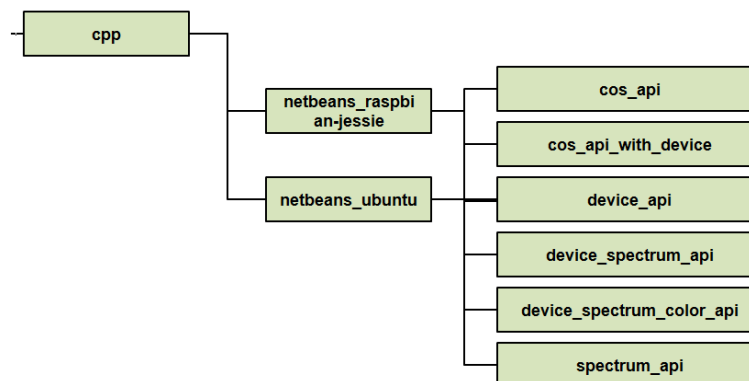
- "examples" folder contains examples for CPP and Python languages.



- Python folder contains 3 sub-directories, platform specific. "raspbian-jessie" for Raspberry pi, "ubuntu" for Ubuntu and "win32" for Windows 32-bit version. Each folder also has 3 sub-folders. "example_wrapper_python" folder contains an example code which tells the usage of almost all the wrapper functions. "libs" folder contains the library for platform specific. For getting the accurate spectral data from the NSP32 spectral sensor, user need to put sensor specific sensor data file in "config" folder.

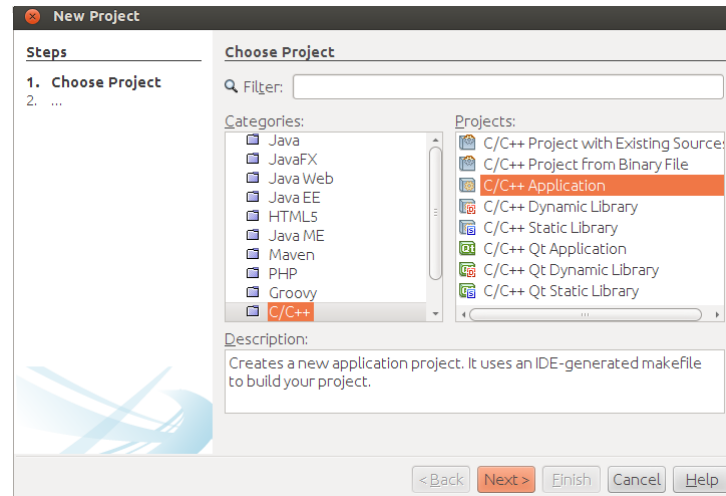


- "cpp" folder contains two folders, one for Raspbian-jessie and one for Ubuntu. Each folder has 6 more directories to show how user can get the data and other information from NSP32 spectral sensor.

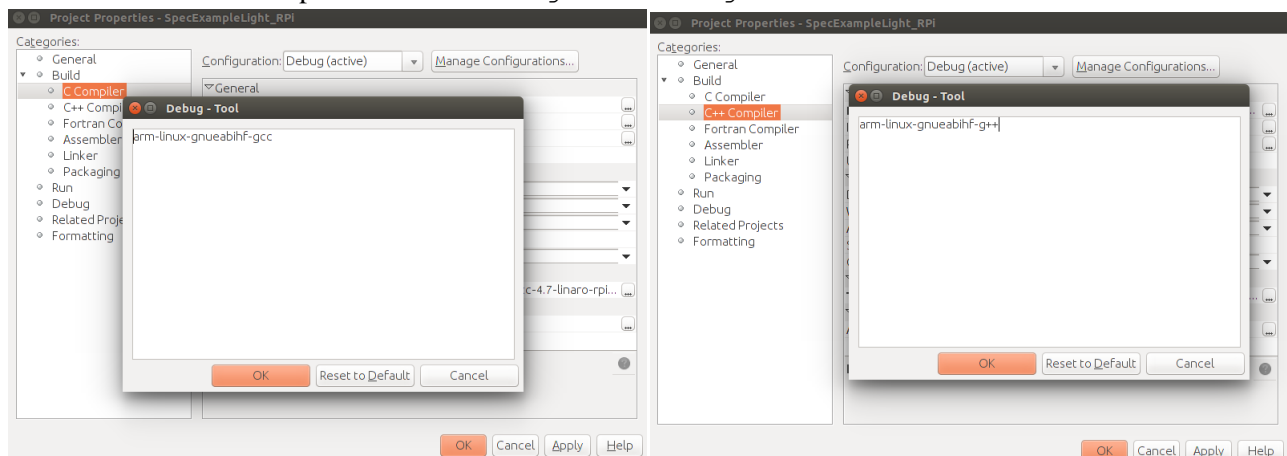


Build and Run C/C++ Example

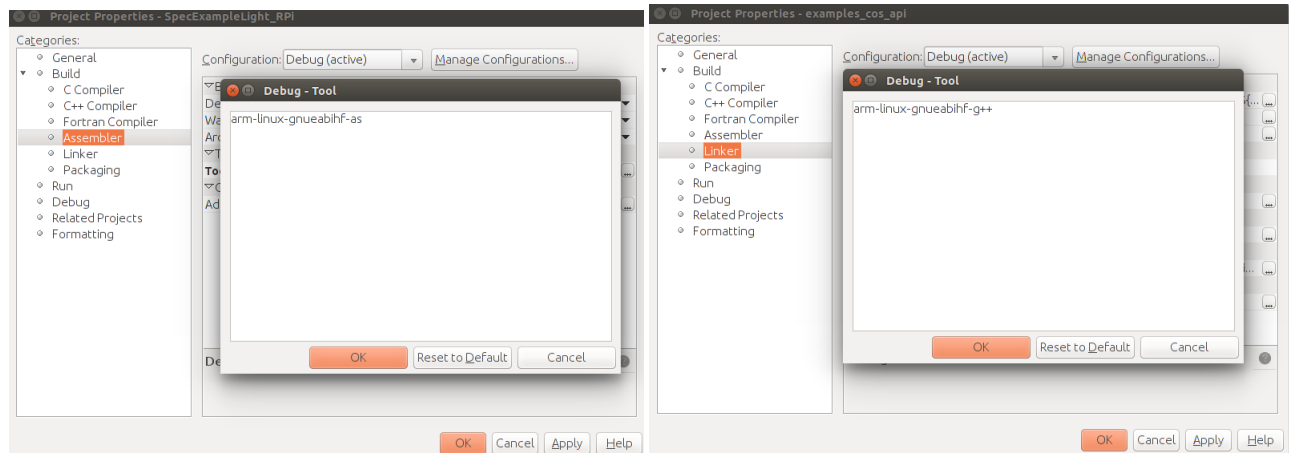
Now, you are ready for build examples from nanoLambda or your own code. For building and running console based example, we will again use the same NetBeans IDE and this time we will select “C/C++ Application” project type from the “New Project” dialog window:



1. On the next screen followed by “Next” button click, enters the project name and project location and click “Finish”.
2. After this, go to the project on NetBeans IDE and right click and select “Properties” on pop-up menu.
3. Change the C Compiler, C++ Compiler, and Assembler and Linker under “Build” property:
 - a. C Compiler: arm-linux-gnueabi-hf-gcc
 - b. C++ Compiler: arm-linux-gnueabi-hf-g++

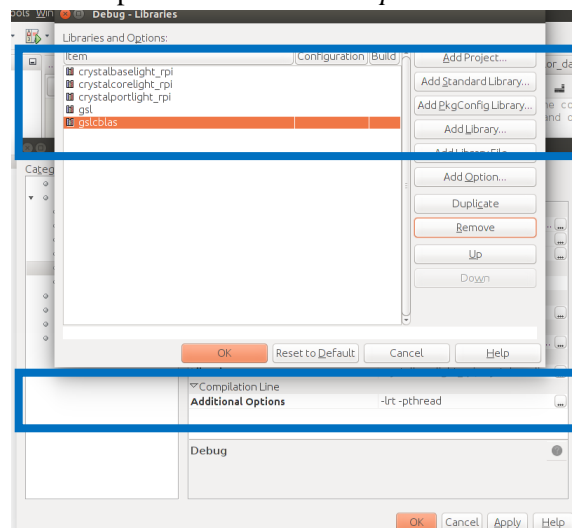


- c. Assembler: arm-linux-gnueabi-hf-as
- d. Linker : arm-linux-gnueabi-hf-g++



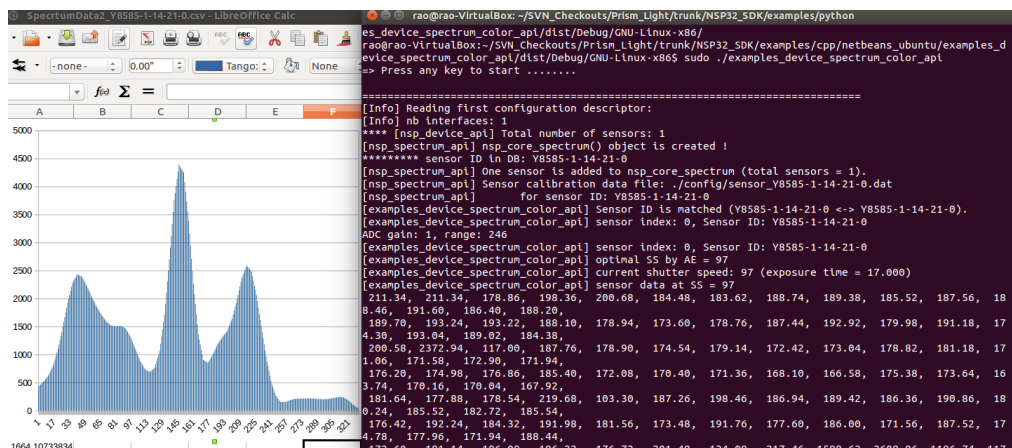
4. In the last, you must add two more things to “*Linker*”:

- a. Libraries: `crystalbase_rpi`, `crystalcore_rpi`, and `crystalport_rpi`
 - You can add 3 libraries included in NSP32 SDK to “*Libraries*” field by using “*Add Library...*” on the “*Libraries*” dialog window.
- b. One additional options : `-lrt -pthread`
 - You can add this option to “*Additional Options*” field as below.



5. Right click the project and build it. After that, copy the executables along with sensor calibration file to Raspberry pi.
6. Run your example with this command
 - a. `$sudo ./examp_name`

For your reference, you can see the results of spectra calculation by one example project (`examples_device_spectrum_color_api`) on the host’s console window below.



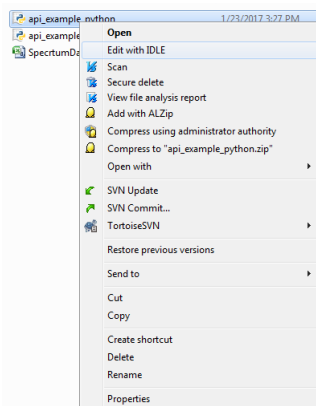
Run Python Example

nanoLambda NSP32 SDK supports Python language with Python wrappers for both version of Python 2.xx and 3.xx. Under 'NSP32_SDK/wrappers/Python' folder, you can find two separate wrapper folders, 'wrapper_Python2' and 'wrapper_Python3'. In each folder, there are 2 sub-directories, 'core' and 'device'. 'core' folder includes all the functions related to the core spectrum of the spectral sensor and 'device' folder includes all the functions related to NSP32 device interface. All the codes are written in C/C++ and they are just wrapped with an additional layer to provide Python interface.

Run on Windows (Win32)

For running the example on Windows machine (32/64-bit):

1. Run 'IDLE'(Python Integrated Development Environment) to edit/execute a Python example file('example_wrapper_python.py')



Note

To run your Python code, you must install Python 2.7.12 or Python 3.4.2 (32-bit) on your computer. You can find both versions of Python (32-bit) from here:

- <https://www.Python.org/downloads/release/Python-2712/>
- <https://www.Python.org/downloads/release/Python-342/>

2. You will get Python example code in IDLE as below. Run Python example by clicking 'Run→Run Module F5' menu on IDLE:

```

api_example_python.py - D:\SVN\Prism_Light\trunk\api\examples\Python\win32\api_example_pytho...
File Edit Format Run Options Window Help

print (["Python-3"] Python Version : ", sys.version_info.major, ".")
print (["Python-3"] Python Version : ", sys.version_info.major, ".")

#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")

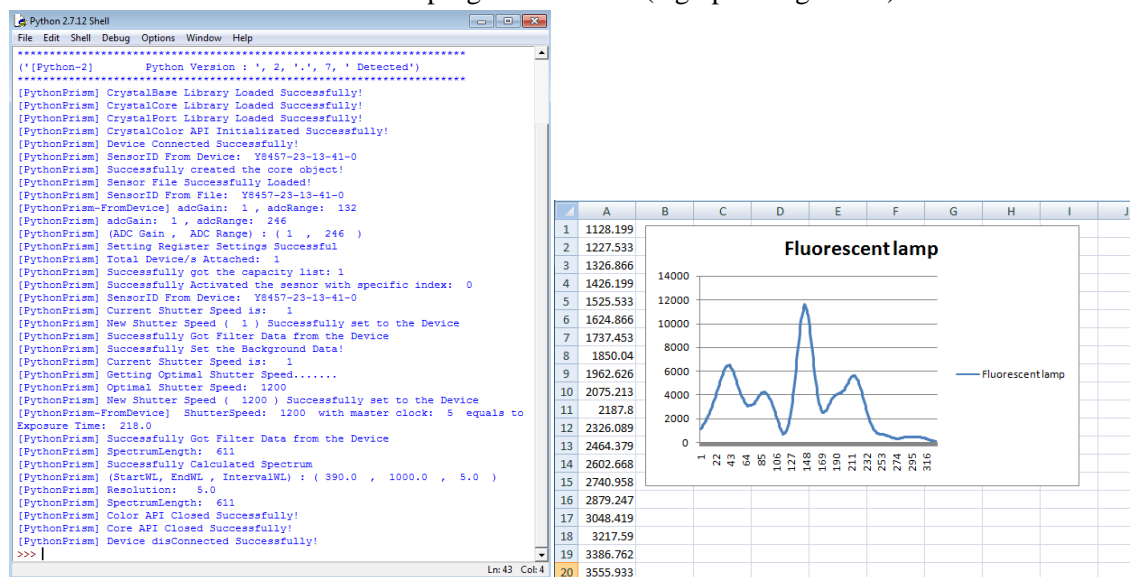
initialize_color_api(pSpecCore)

connectReturn = connect_device(pSpecDevice) # return total num of devices co

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_" + senso
    else:
        csInit_Return = load_sensor_file(pSpecCore, b"..\\config\\sensor_" + senso
    (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_sensor_data_list(pSpecCore)

```

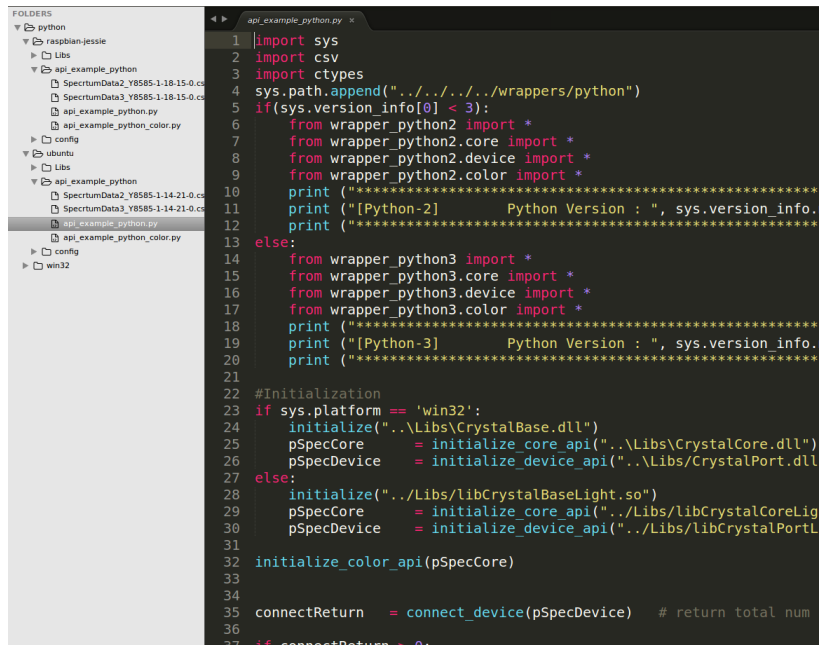
- b. You will get some text outputs from Python Shell and one CSV format file having the acquired and calculated spectra from NSP32 spectral sensor. You can check the spectrum data in CSV format file with Microsoft Excel program as below (a graph at right side).



Run on Raspbian-Jessie or Linux(Ubuntu)

For running the example on either Raspbian-Jessie or Linux(Ubuntu),

- copy the your sensor calibration file (e.g., 'sensor_Y8585-1-85-85-0.dat') to a sub-folder named with 'config'.
- \$cd example_wrapper_python
- Open your Python example code with SubLime text editor. A picture at below is a snapshot of this Python example code.

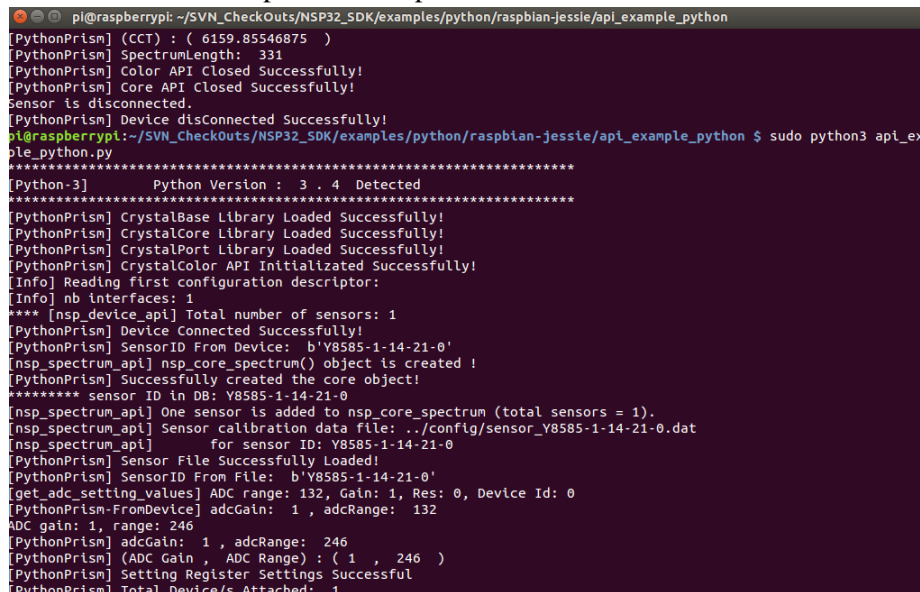


```

1 import sys
2 import csv
3 import ctypes
4 sys.path.append(".././../wrappers/python")
5 if(sys.version_info[0] < 3):
6     from wrapper_python2 import *
7     from wrapper_python2.core import *
8     from wrapper_python2.device import *
9     from wrapper_python2.color import *
10    print ("*****")
11    print ("[Python-2]      Python Version : ", sys.version_info.maj
12    print ("*****")
13 else:
14     from wrapper_python3 import *
15     from wrapper_python3.core import *
16     from wrapper_python3.device import *
17     from wrapper_python3.color import *
18    print ("*****")
19    print ("[Python-3]      Python Version : ", sys.version_info.maj
20    print ("*****")
21
22 #Initialization
23 if sys.platform == 'win32':
24     initialize("../Libs\CrystalBase.dll")
25     pSpecCore = initialize_core_api("../Libs\CrystalCore.dll")
26     pSpecDevice = initialize_device_api("../Libs\CrystalPort.dll")
27 else:
28     initialize("../Libs/libCrystalBaseLight.so")
29     pSpecCore = initialize_core_api("../Libs/libCrystalCoreLight
30     pSpecDevice = initialize_device_api("../Libs/libCrystalPortLi
31
32 initialize_color_api(pSpecCore)
33
34
35 connectReturn = connect_device(pSpecDevice) # return total num o
36
37 if connectReturn > 0:

```

6. Run example for different versions of Python in the terminal window:
 - a. For Python 2.xx: `$sudo python example_wrapper_pyhon.py`
 - b. For Python 3.xx: `$sudo python3 example_wrapper_pyhon.py`
7. You'll see the terminal outputs of example as below:



```

pi@raspberrypi: ~/SVN_CheckOuts/NSP32_SDK/examples/python/raspbian-jessie/api_example_python
[PythonPrism] (CCT) : ( 6159.85546875 )
[PythonPrism] SpectrumLength: 331
[PythonPrism] Color API Closed Successfully!
[PythonPrism] Core API Closed Successfully!
Sensor is disconnected.
[PythonPrism] Device dlsConnected Successfully!
pi@raspberrypi:~/SVN_CheckOuts/NSP32_SDK/examples/python/raspbian-jessie/api_example_python $ sudo python3 api_ex
le_python.py
*****
[Python-3]      Python Version : 3 . 4 Detected
*****
[PythonPrism] CrystalBase Library Loaded Successfully!
[PythonPrism] CrystalCore Library Loaded Successfully!
[PythonPrism] CrystalPort Library Loaded Successfully!
[PythonPrism] CrystalColor API Initialized Successfully!
[Info] Reading first configuration descriptor:
[Info] nb interfaces: 1
*** [nsp_device_api] Total number of sensors: 1
[PythonPrism] Device Connected Successfully!
[PythonPrism] SensorID From Device:  b'Y8585-1-14-21-0'
[nsp_spectrum_api] nsp_core_spectrum() object is created !
[PythonPrism] Successfully created the core object!
***** sensor ID in DB: Y8585-1-14-21-0
[nsp_spectrum_api] One sensor is added to nsp_core_spectrum (total sensors = 1).
[nsp_spectrum_api] Sensor calibration data file: ../config/sensor_Y8585-1-14-21-0.dat
[nsp_spectrum_api] for sensor ID: Y8585-1-14-21-0
[PythonPrism] Sensor File Successfully Loaded!
[PythonPrism] SensorID From File:  b'Y8585-1-14-21-0'
[get_adc_setting_values] ADC range: 132, Gain: 1, Res: 0, Device Id: 0
[PythonPrism-FromDevice] adcGain: 1 , adcRange: 132
ADC gain: 1, range: 246
[PythonPrism] adcGain: 1 , adcRange: 246
[PythonPrism] (ADC Gain , ADC Range) : ( 1 , 246 )
[PythonPrism] Setting Register Settings Successful
[PythonPrism] Total Device/s Attached: 1

```

8. You can also plot the acquired spectra with 'LibreOffice' program as below:

