

StellarNet

Python SDK Stellarnet_driver3

Documentation v2.6



14390 Carlson Circle
Tampa, FL 33626
+1 (813) 855-8687



CONTENTS

Introduction.....	4
Compiled Driver Support.....	5
StellarNet Python Driver Installation.....	6
➤ Windows	6
a. Python 3 Installation & Setup	6
b. Install Packages.....	10
c. Switching from Windows DLL to Python driver	11
i. Method 1: Run InstallDriver.exe	11
ii. Method 2: Install Libusb Filter	11
➤ Linux 64 bit (Ubuntu) / Mac	15
a. Python 3 Installation & Setup	15
b. Install Libusb Filter	15
c. Install Packages.....	15
Getting Started	16
➤ Python StellarNet Driver Demo	16
➤ MATLAB StellarNet Driver Demo	20
➤ C# / .NET StellarNet Driver Demo.....	23
➤ LabVIEW 2018+ StellarNet Driver Demo	27
Python API Reference.....	31
• array_get_spec(chan)	31
• array_get_spec_only (chan)	32
• array_spectrum(spectrometer, wav).....	33
• ext_trig(spectrometer, trigger_val)	34
• getSpectrum_X(spectrometer)	35
• getSpectrum_Y(spectrometer)	36
• getDeviceId(spectrometer).....	37
• getDeviceParam(spectrometer).....	38
• setParam(spectrometer, inttime, scansavg, smooth, xtiming, clear =True)	39
• setTempComp(spectrometer, temp_comp).....	42
• getBurstFifo_Y(spectrometer)	43



• allowBurst(spectrometer).....	44
• total_device_count()	44
• getFullDeviceID(spectrometer)	45
• deviceConnectionCheck(spectrometer)	46
• reset(spectrometer).....	47
• version()	47
• installDeviceDriver ()	48
• uninstallDeviceDriver()	49
• Class StellarNet(object)	50
○ __init__(self, device)	50
○ __del__(self)	50
○ extrig (self, trigger)	50
○ set_config(self, **kwargs)	51
○ get_config(self)	52
○ get_device_id(self).....	53
○ read_spectrum(self).....	53
○ compute_lambda(self, pixel).....	54
○ print_info(self) 55	
Version Update	56
➤ Version 2.6.....	56
➤ Version 2.5 56	
License	56



INTRODUCTION

The StellarNet Python driver enables seamless integration with StellarNet spectrometers, providing essential functionalities for changing device settings, acquiring data, and developing custom applications. This capability is crucial for users who need precise control and real-time data collection from their spectrometers, as it allows for tailored experimental setups and automated processes. The driver also supports interoperability with languages and environments such as MATLAB, C# / .NET, and LabVIEW 2018+, enhancing its versatility and making it easier for researchers and engineers to incorporate spectrometer data into their diverse workflows. This flexibility is key for optimizing experimental outcomes, enhancing data precision, and improving efficiency in data analysis.

The provided directory contains:

- Demo script file(s).
- stellarnet_driverLibs directory:
 - The stellarnet_driverLibs folder contains compiled Python drivers for different Python versions, bitness, and operating systems.
 - Windows_only
 - Note: StellarNet provides two different drivers for spectrometer USB communication on Windows, and they cannot be run simultaneously: the *StellarNet Python USB Driver* and the *USBDEV>StellarNet Spectrometer driver* for StellarNet spectrometers. The *StellarNet Python USB Driver* is required to run this Python driver. Please review the [Switching Between Drivers Knowledge Base](#) for more information.
 - InstallDriver.exe → Used by the function [installDeviceDriver\(\)](#) to install the Python Driver, or you can run it manually to install the driver.
 - StellarNet_Python_USB_Driver → Used by the function [uninstallDeviceDriver\(\)](#) to remove the Python Driver.



COMPILED DRIVER SUPPORT

The table shows a list of the available operating systems, bit size, and the Python version for the StellarNet Driver. All of the compiled driver files will be in the “Stellarnet_driverLibs” folder.

Operating Systems	File Format	Python Support
Window (64 bits)	stellarnet_driver3.cp**.-win_amd64.pyd	2.7, 3.5 to 3.12
Window (32 bits)	stellarnet_driver3.cp**.-win32.pyd	2.7, 3.5 to 3.10
Linux	stellarnet_driver3.cpython-3*m-x86_64-linux-gnu.so	3.5 to 3.12
Mac	stellarnet_driver3.cpython-3*-darwin.so	3.7 to 3.12
Raspberry pi (32 bits)	stellarnet_driver3.cpython-3*-arm-linux-gnueabi.so	3.7 to 3.8 3.10 to 3.11
Raspberry pi (64 bits)	stellarnet_driver3.cpython-3*-aarch64-linux-gnu.so	3.7 to 3.12

NOTE:

- For Python 2.7, please remove “.cp27-win_amd64” or “.cp27-win32” from the file name. Do NOT change any other file names or file extensions unless instructed to do so.
- You may delete any compiled driver file(s) which are not suitable for your operating system/bit size/Python version to save space. For example, if you are only using the driver for 64-bit Windows OS, then other compiled drivers with the extension of “.so” can be removed as well as “stellarnet_driver3.cp**.-win64.pyd”.

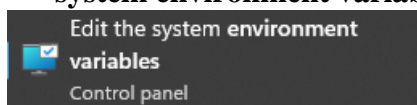
STELLARNET PYTHON DRIVER INSTALLATION

Note: If you are using Python Driver with other programming languages or platforms, ensure that the Python bitness matches that of the other languages or platforms. For example, if MATLAB is 64-bit, then Python must also be 64-bit. The specific Python version does not matter, as long as you are using one of the compiled Python versions listed in the [Compiled Driver Support](#) section. Please review the [Getting Started](#) section for your specific languages or platforms **before** following the steps below to install the Python executable.

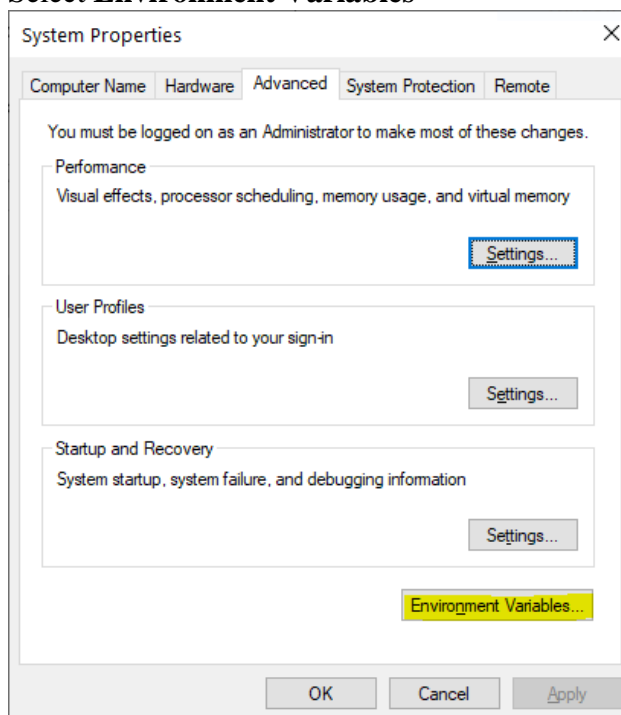
➤ Windows

a. Python 3 Installation & Setup

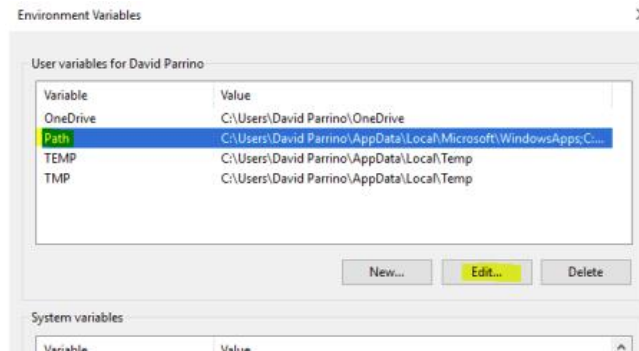
- i. Install Python by navigating to the Python.org [Downloads page](#). A newer version of Python should work, but the driver is being tested with Python 3.9 for this example.
- ii. Add Python to PATH variable
 - A. In the search box on the taskbar at the bottom left hand of your desktop, type “environment variables”, and then select **Edit the system environment variables**



B. Select Environment Variables

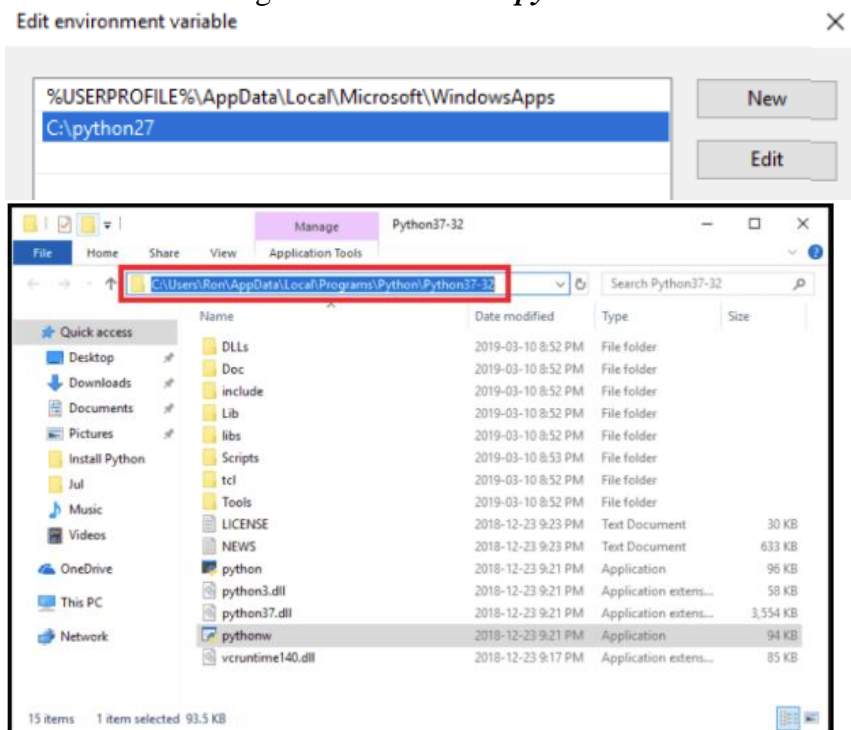


C. Under **user variable for user**, click **Path** and then click **Edit**



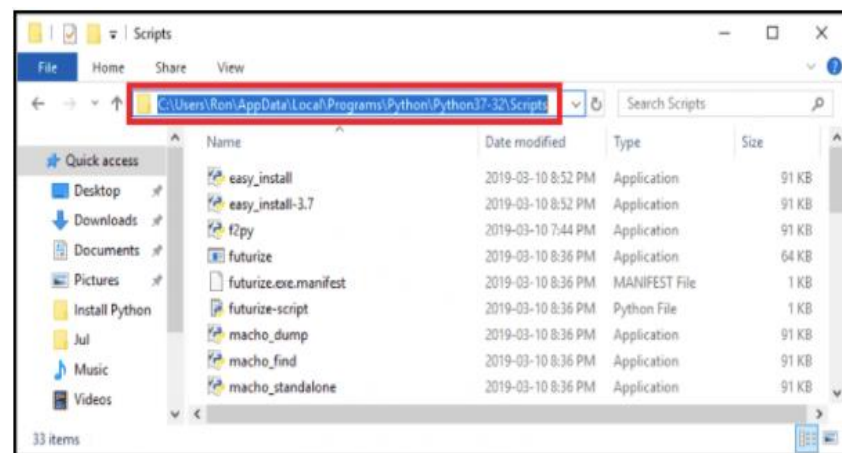
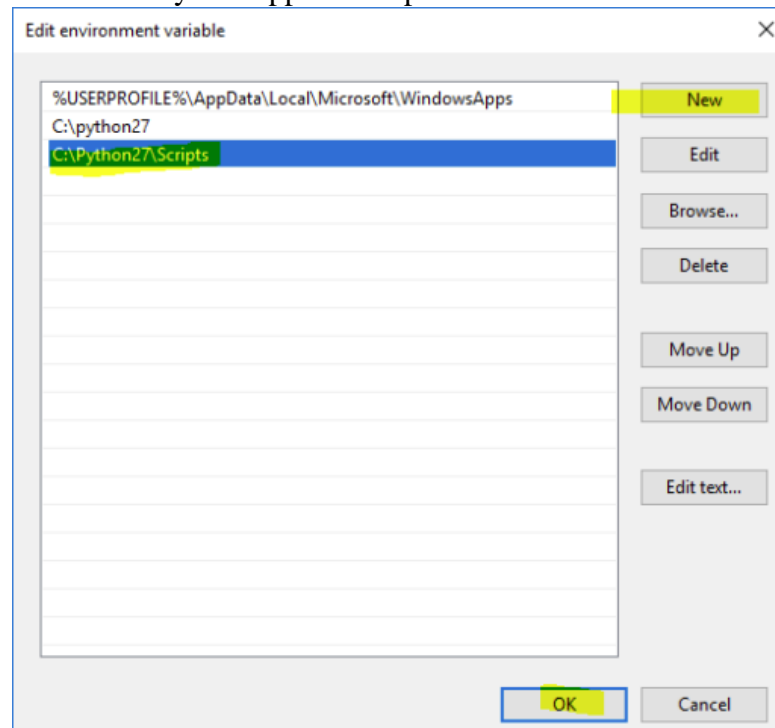
D. In the new pop-up window, click **New** and type in the Python application path.

- The Python application path is the folder where you originally installed Python. This path can be found by searching for the location of *python.exe*



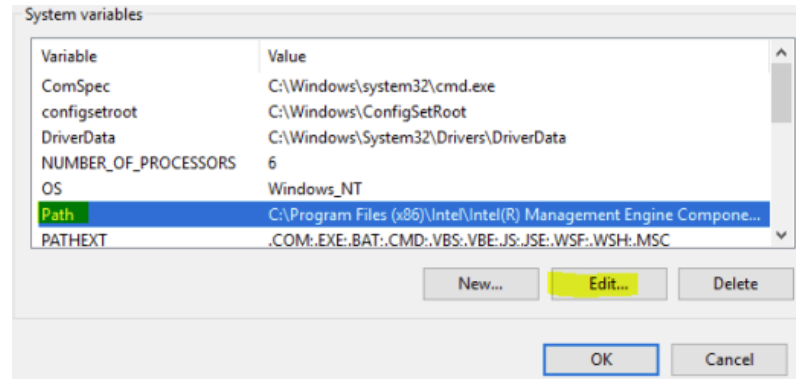
(An example of the Python application path)

- E. Again, click **New**, and type in the Python Scripts path and click **OK** to close and save the window.
- The Python Scripts folder should be located within the Python application path.

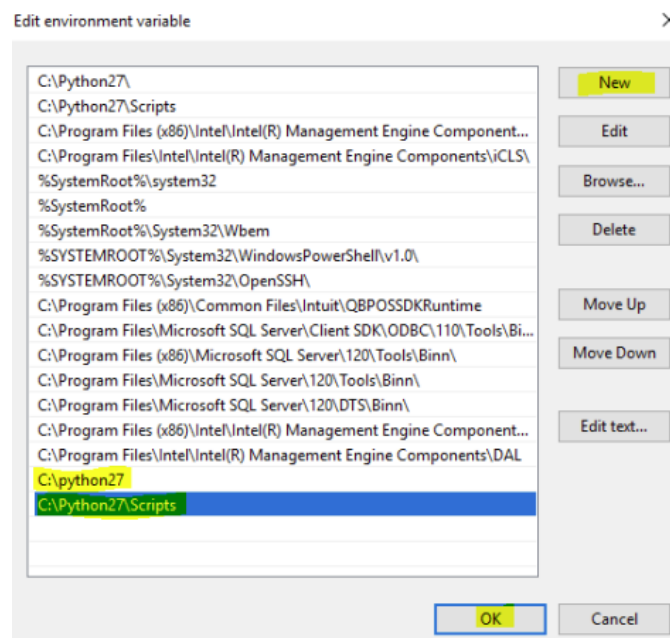


(An example of Python Scripts path)

F. Under **System variables** in the Environment Variables window, click **Path** and then **Edit**



G. In the new pop-up window, click **New**, add the SAME two paths, i.e. the Python application path and Python Scripts path, and click **OK** to close and save the window.

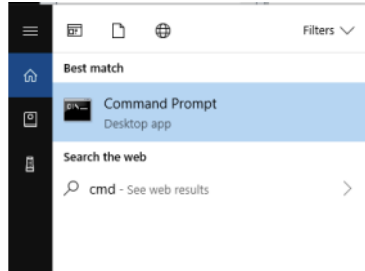


H. Python now can be used directly from the Command Prompt without having to write its location.

b. Install Packages

Python drivers require some packages and libraries. Follow the steps below to install all the dependencies.

i. In the Start Menu, search **Command Prompt**



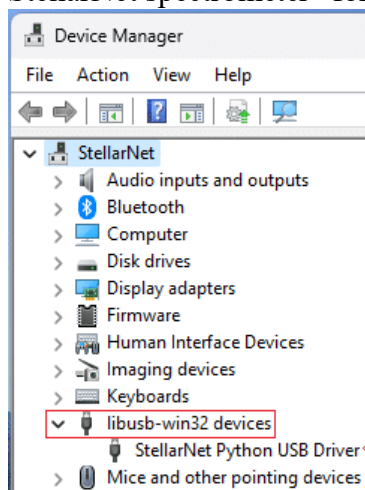
- iii. In the terminal, type “*pip install numpy*” to install numpy package
- iv. In the terminal, type “*pip install pyusb*” to install pyusb package



c. Switching from Windows DLL to Python driver

i. Method 1: Run InstallDriver.exe

- A. Find "InstallDriver.exe" located in the "stellarnet_driverLibs/windows_only" directory.
- B. Right-click on "InstallDriver.exe" and select "Run as Administrator" to run the driver. A popup window "Device Driver Installation Wizard" will appear.
- C. In the popup window, click "Next>" and then you should see "Ready to use" in the Status column. Click "Finish" to close this window.
- D. Unplug and re-plug the spectrometer.
- E. Go to "Device Manager" (search for "Device Manager" in the Start menu).
- F. In Device Manager, you should now see "libusb-win32 devices > StellarNet Python USB Driver" listed instead of "USBDEV > StellarNet spectrometer" for the StellarPro driver.




- G. More information on how to switching between the StellarNet Windows DLL and Python Driver can be found at the [StellarNet Knowledge Base](#).

ii. Method 2: Install Libusb Filter

Libusb Filter is needed to redirect the Windows DLL based StellarNet driver from communicating with the spectrometer so that the Python driver will be able to take over. Note, if you've never installed the windows driver, you might be able to skip this step. It is often much easier to identify the StellarNet spectrometer if the Windows Driver has already been installed.

A. Install [*libusb-win32-devel-filter-1.2.6.0*](#)

B. Double click on the executable *libusb-win32-devel-filter-1.2.6.0*

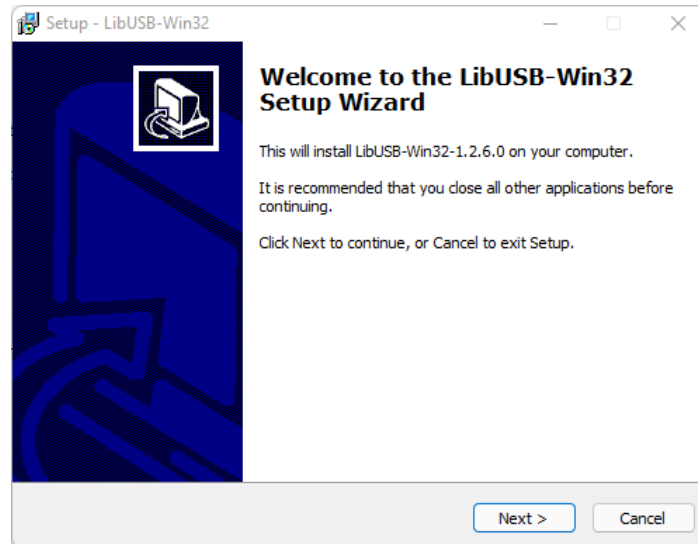
 libusb-win32-devel-filter-1.2.6.0

12/27/2021 11:02 AM

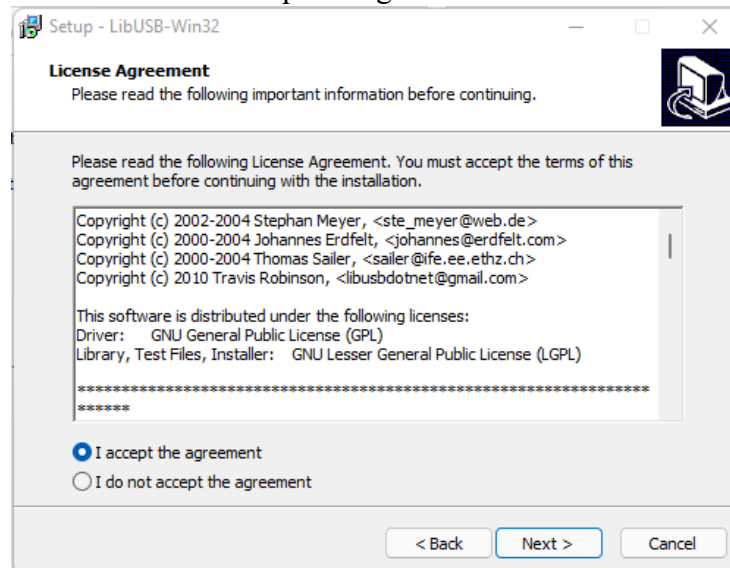
Application

627 KB

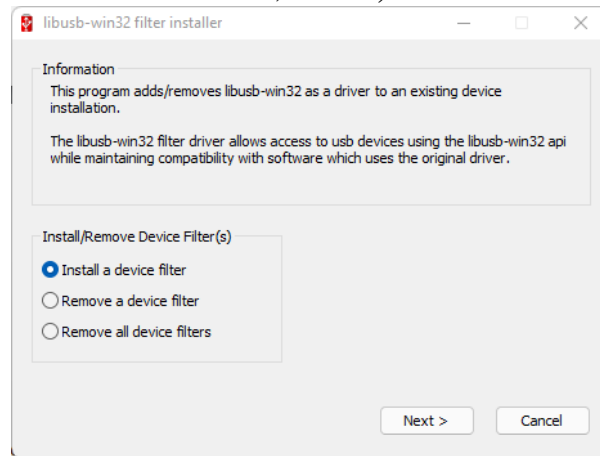
C. Click **Next >**



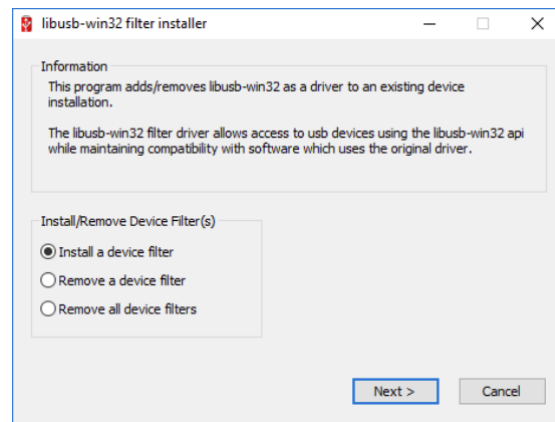
D. Select “I accept the agreement” and click **Next >**



E. Click **Next >**, **Install**, and **Finish** until you reach the page below

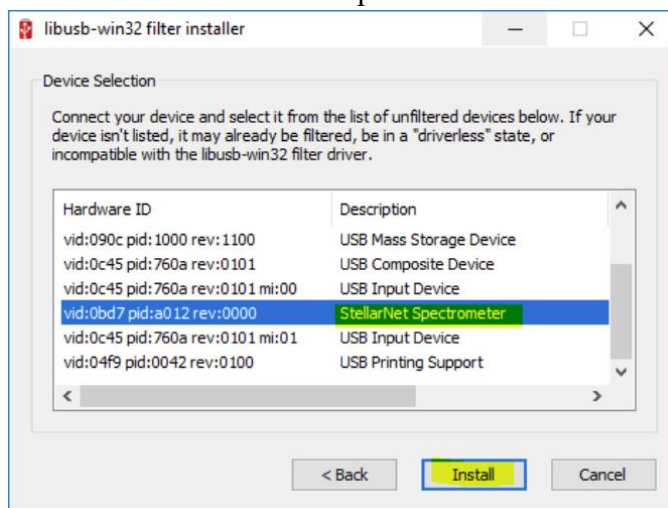


F. Select “Install a device filter” and click **Next >**



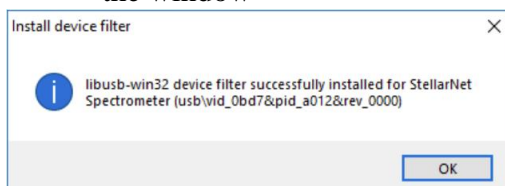


G. Select “StellarNet Spectrometer” and click **Install**



*Note: If the Windows DLL has not been installed, the Description will not appear as StellarNet Spectrometer.

H. The filter should be successfully installed. Click OK to close out the window





➤ **Linux 64 bit (Ubuntu) / Mac**

a. Python 3 Installation & Setup

By default, Linux / Mac comes with Python already installed. More information on how to install the Python can be found at python.org.

b. Install Libusb Filter

Linux: `sudo apt-get install libusb-1.0-0-dev`

Mac: `brew install libusb`

c. Install Packages

Python drivers require some packages and libraries. Follow the steps below to install all the dependencies.

- i. In the terminal, type `"pip install numpy"` to install numpy package
- ii. In the terminal, type `"pip install pyusb"` to install pyusb package

Note, all dependencies and libraries installation instructions are provided, but for any reason if you get an import error, try to install the unsuccessful library through PIP, or if you have any questions, you can email Support@StellarNet.us with screenshots to illustrate the situation.



GETTING STARTED

➤ Python StellarNet Driver Demo

- Make sure Python is installed as described in [StellarNet Python Driver Installation](#).
- Place this folder in the same folder as the stellarnet_demo.py and at the beginning of the demo script should include the following statement to link the code file with the driver library
 - `from stellarnet_driverLibs import stellarnet_driver3 as sn`
- To run the script, use the following commands based on your operating system:
 - For windows/Mac: **python stellarnet_demo.py**
 - For Linux/Raspi: **sudo python stellarnet_demo.py**
 - Sudo command is needed for libusb to communicate with the spectrometer, allowing for reading from and writing to the spectrometer.
 - Note about sudo with anaconda environment: sudo employs a different PATH than the typical environment. The Python version in the sudo environment might not be the same as the anaconda environment that you are running on (you can run 'sudo python --version' to see the Python version in the sudo environment). Ensure that you specify the use of Anaconda's Python interpreter rather than the system Python.

Check which Anaconda environment is being run with the following command:

which python

Then, run the sudo command pointing to Anaconda's Python interpreter. Specify the full path to the correct interpreter:

sudo stellarnet_demo.py

Note: Replace with the path shown in 'which python', i.e., the path to Anaconda's Python interpreter.



stellarnet_demo.py

```
from stellarnet_driverLibs import stellarnet_driver3 as sn

# For Windows ONLY: Must be run in administrator mode
# Only need to run it one time after switch back from the SpectraWiz. Or manually run In-
installDriver.exe located in stellarnet_driverLis/windows_only
# sn.installDeviceDriver()

# This return a Version number of compilation date of driver
version = sn.version()
print(version)

# Device parameters to set
inttime = 50
scansavg = 1
smooth = 0
xtiming = 3

#init Spectrometer - Get BOTH spectrometer and wavelength
spectrometer, wav = sn.array_get_spec(0) # 0 for first channel and 1 for second channel , up to
127 spectrometers
"""

# Equivalent to get spectrometer and wav separately:
spectrometer = sn.array_get_spec_only(0)
wav = sn.getSpectrum_X(spectrometer)
"""
```



```
print(spectrometer)

sn.ext_trig(spectrometer, True)

# Get device ID
deviceID = sn.getDeviceId(spectrometer)
print('\nMy device ID: ', deviceID)

# Get current device parameter
currentParam = sn.getDeviceParam(spectrometer)

# Call to Enable or Disable External Trigger to by default is Disbale=False -> with timeout
# Enable or Disable Ext Trigger by Passing True or False, If pass True than Timeout function
will be disable, so user can also use this function as timeout enable/disbale
sn.ext_trig(spectrometer,True)

# Only call this function on first call to get spectrum or when you want to change device set-
ting.
# -- Set last parameter to 'True' throw away the first spectrum data because the data may not be
true for its inttime after the update.
# -- Set to 'False' if you want to throw away the first data, however your next spectrum data
might not be valid.
sn.setParam(spectrometer, inttime, scansavg, smooth, xtiming, True)

# Get spectrometer data - Get BOTH X and Y in single return
first_data = sn.array_spectrum(spectrometer, wav) # get specturm for the first time
"""

# Get Y value ONLY :
first_data = sn.getSpectrum_Y(spectrometer)
```



```
""  
  
print('First data:', first_data )  
  
#=====   
# Burst FIFO mode: Not recommended with high integration time.  
# burst_data_2 = sn.getBurstFifo_Y(spectrometer)  
#=====   
  
# Release the spectrometer before ends the program  
sn.reset(spectrometer)  
  
# For Windows ONLY: Must be run in administrator mode  
# sn.uninstallDeviceDriver()
```



➤ MATLAB StellarNet Driver Demo

1. Ensure that the bitness of the installed Python matches the bitness of MATLAB; for example, if MATLAB is 64-bit, then Python must also be 64-bit. Python should be installed according to the instructions in the [StellarNet Python Driver Installation](#).
2. Ensure that the `stellarnet_driverLibs` directory is located in the same directory as the file that contains the script below.

```
% MATLAB demo using StellarNet Python Driver

try
    %Use the command below with correct python exe path to load the python version
    pyenv('Version', 'C:\Python38.exe'); % UPDATE 'C:\Python38.exe' with your own python
    path
catch
end

% calling spectrometer device, 0 represents first channel, 1 represents second channel, etc
channel = int64(0);
spectrometer=py.stellarnet_driverLibs.stellarnet_driver3.array_get_spec_only(channel);

% setting parameters
inttime = int64(100);
xtiming = int64(3);
scansavg = int64(1);
smoothing = int64(4);

param=py.stellarnet_driverLibs.stellarnet_driver3.setParam(spectrometer,inttime,scan-
savg,smoothing, xtiming);% calling lib to set parameter

% Get Spectrometer wavelength (X)
```



```
wav = py.stellarnet_driverLibs.stellarnet_driver3.getSpectrum_X(spectrometer);% getting
wavelengths

% Get Spectrum from the spectrometer
spectrum = py.stellarnet_driverLibs.stellarnet_driver3.getSpectrum_Y(spectrometer);% get-
ting spectrum

% plotting data
plot(wav,spectrum)
xlabel('Wavelength in nm')
ylabel('Counts')
```



MATLAB Demo Code Explanation:

1. **Link Python environment in MATLAB.** Change 'C:\Python38.exe' to your own Python EXE path. For Windows:
 - Search for *python.exe* in the Start menu.
 - Right-click on *python.exe* and select **Open file location**. This will open the directory containing the python executable under the Start menu.
 - Copy the path from the address bar in Windows File Explorer.

```
pyenv('Version', 'C:\Python38.exe');
```

2. **Get spectrometer device.** Channel can be adjusted by passing the channel parameter into the function `array_get_spec`. For example, pass in 0 for channel 1, 1 for channel 2, etc.

```
channel = int64(0);
```

```
spectrometer=py.stellarnet_driverLibs.stellarnet_driver3.array_get_spec_only(channel);
```

3. **Configure spectrometer setting.**

```
py.stellarnet_driverLibs.stellarnet_driver3.setparam(spectrometer,inttime,xtiming,scansavg,smoothing);
```

4. **Get spectrometer wavelength (X) as array data type.**

```
py.stellarnet_driverLibs.stellarnet_driver3.getSpectrum_X(spectrometer)
```

5. **Get spectrometer data (Y) as array data type.**

```
py.stellarnet_driverLibs.stellarnet_driver3.getSpectrum_Y(spectrometer)
```

6. See [Python API Reference](#) for additional functions.



➤ C# / .NET StellarNet Driver Demo

1. Ensure that the bitness of the installed Python matches the bitness of .NET; for example, if .NET is 64-bit, then Python must also be 64-bit. Python should be installed according to the instructions in the [StellarNet Python Driver Installation](#).
2. Add the Python Runtime Library (Required packages [pythonnet](#)):
 - a. In the terminal, run "dotnet add package pythonnet --version 3.0.3" to install the pythonnet library. It is a required library to call python code in .NET program.
 - b. [Python.Net Documentation](#)
3. Copy and paste the provided `stellarnet_driverLibs` folder into the same directory as `Python.Runtime.dll` (e.g., `bin\Debug\net8.0`). The folder contains several Stellarnet Python drivers. You can delete the drivers that do not match your Python version, or you can leave them as they are. Your program will automatically find the matching driver if it exists.
4. In the terminal, type "dotnet run" to run the program.

```
using Python.Runtime;

class StellarNet_CSharp_demo
{
    static void Main()
    {
        // NOTE: Replace this with you python version dll path. Make sure the python path is set
        // in PATH environment variable as well.

        Runtime.PythonDLL = @"C:\Python38\python38.dll";

        // Initialize the python runtime engine
        PythonEngine.Initialize();

        // If you plan to use Python objects from multiple threads, also call
        PythonEngine.BeginAllowThreads();
    }
}
```



```
using (Py.GIL())
{
    // Import python driver
    var pythonScript = Py.Import("stellarnet_driverLibs.stellarnet_driver3");

    // Device parameters to set
    dynamic inttime = new PyInt(50);
    dynamic scansavg = new PyInt(1);
    dynamic smooth = new PyInt(0);
    dynamic xtiming = new PyInt(3);

    // Initialize Spectrometer - Get spectrometer object and wavelength.
    dynamic spec_channel = new PyInt(0); // spectrometer channel
    dynamic spectrometer = pythonScript.InvokeMethod("array_get_spec_only",
spec_channel);
    dynamic specWave = pythonScript.InvokeMethod("getSpectrum_X", spectrome-
ter).tolist();

    System.Console.WriteLine("Spectrometer Wavelength: "+specWave);

    dynamic deviceID = pythonScript.InvokeMethod("getDeviceId", spectrometer);
    System.Console.WriteLine("Device ID: "+deviceID);

    // Print current parameters
    System.Console.WriteLine("Current Device Parameters: "+pythonScript.In-
vokeMethod("getDeviceParam", spectrometer));

    // Change device parameters
```




```
dynamic specParams = pythonScript.InvokeMethod("setParam", spectrometer, inttime,
scansavg, smooth, xtiming, new PyInt(1));

// Print new parameters

System.Console.WriteLine("New Device Parameters: "+pythonScript.In-
vokeMethod("getDeviceParam", spectrometer));

dynamic rawData;

// Capture and print raw Y data 5 times
for (int i = 0; i < 5; i++)
{
    rawData = pythonScript.InvokeMethod("getSpectrum_Y", spectrometer).tolist();
    System.Console.WriteLine(rawData);
}
pythonScript.InvokeMethod("reset", spectrometer);
}
}
}
```



C# / .NET Demo Code Explanation:

1. Link Python environment in C#. Change 'C:\Python38\python38.dll' to your own Python DLL path. For Windows:
 - Search for *python.exe* in the Start menu.
 - Right-click on *python.exe* and select **Open file location**. This will open the directory containing the Python executable under the Start menu.
 - In the same directory, you will find the Python DLL.
 - Copy the path from the address bar in Windows File Explorer.

2. Initialize the python runtime engine

```
PythonEngine.Initialize();
```

3. Import python driver

```
var pythonScript = Py.Import("stellarnet_driverLibs.stellarnet_driver3");
```

4. **Get spectrometer device.** Channel can be adjusted by passing the channel parameter into the function `array_get_spec`. For example, pass in 0 for channel 1, 1 for channel 2, etc.

```
dynamic spec_channel = new PyInt(0); // spectrometer channel
dynamic spectrometer = pythonScript.InvokeMethod("array_get_spec_only",
spec_channel);
```

5. Configure spectrometer setting.

```
pythonScript.InvokeMethod("setParam", spectrometer, inttime, scansavg, smooth,
xtiming, new PyInt(1));
```

6. Get spectrometer wavelength (X) as array data type.

```
dynamic specWave = pythonScript.InvokeMethod("getSpectrum_X", spectrometer);
```

7. Get spectrometer data (Y) as array data type.

```
dynamic rawData = pythonScript.InvokeMethod("getSpectrum_Y", spectrometer);
```


8. See [Python API Reference](#) for additional functions.



➤ LabVIEW 2018+ StellarNet Driver Demo

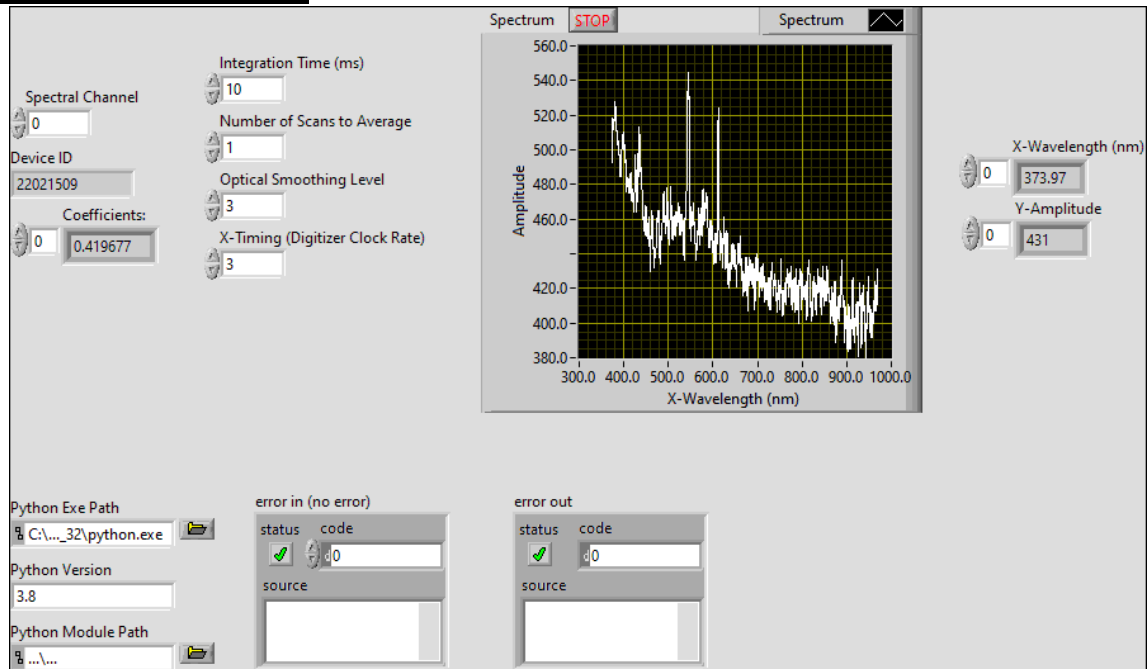
1. LabVIEW and Python Compatibility. LabVIEW Python Node allows you to call Python scripts directly from a LabVIEW Block Diagram, enabling seamless interoperability between the two languages. This functionality was introduced in LabVIEW 2028 and is not available in previous versions. Although unsupported versions might work with the LabVIEW Python functions, NI recommends using supported version of Python only.

LabVIEW Version	Python Version					
	3.10	3.9	3.8	3.7	3.6	2.7
2024 Q3	Compatible	Compatible	Compatible	Compatible	Compatible	
2024 Q1	Compatible	Compatible	Compatible	Compatible	Compatible	
2023 Q3	Compatible	Compatible	Compatible	Compatible	Compatible	
2023 Q1	Compatible	Compatible	Compatible	Compatible	Compatible	
2022 Q3		Compatible	Compatible	Compatible	Compatible	
2021 SP1		Compatible	Compatible	Compatible	Compatible	
2021		Compatible	Compatible	Compatible	Compatible	
2020 SP1					Compatible	Compatible
2020					Compatible	Compatible
2019 SP1					Compatible	Compatible
2019					Compatible	Compatible
2018 SP1					Compatible	Compatible
2018					Compatible	Compatible

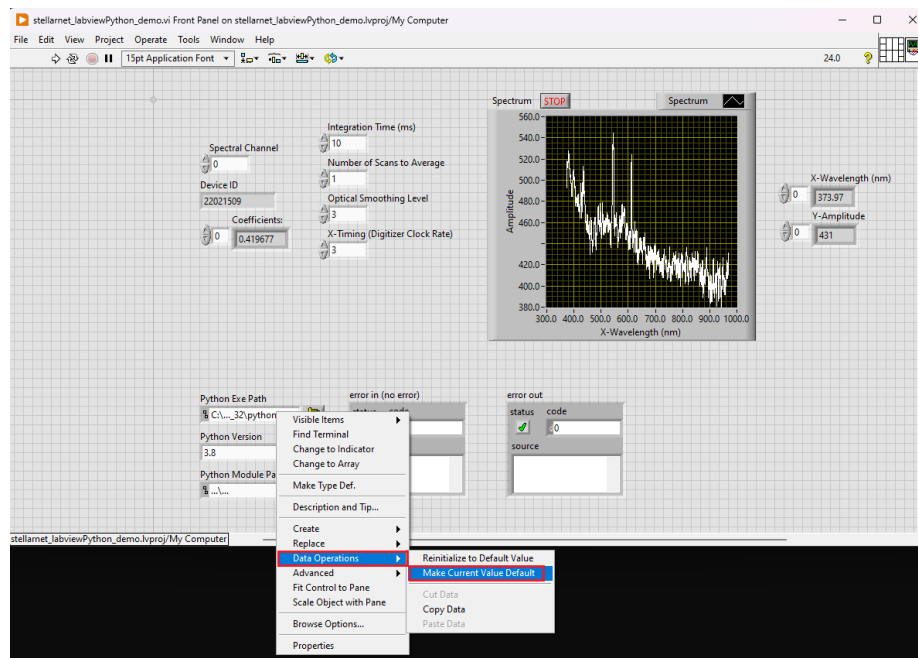
 Compatible

2. Ensure that the bitness of the installed Python matches the bitness of MATLAB; for example, if MATLAB is 64-bit, then Python must also be 64-bit. Python should be installed according to the instructions in the [StellarNet Python Driver Installation](#).
 - Tested on the 64 bits Windows with LabVIEW 32 bits, Python 32-bit, Python version 3.8, and stellarnet_driver3.cp38-win_amd64.
3. See [Python API Reference](#) for additional functions.

LabVIEW Front Panel

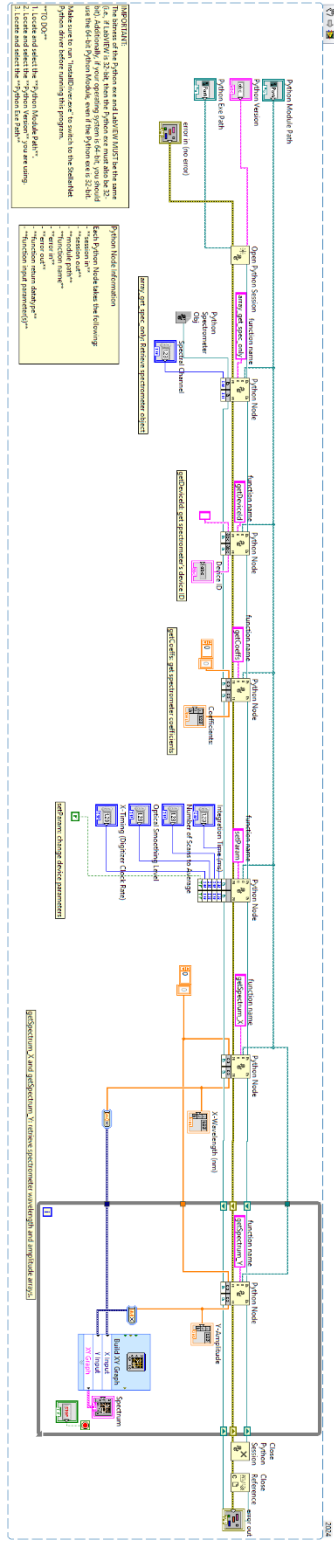


1. **Set up the Python executable path:** At the bottom left of the panel, click on the file icon for the Python executable path. Locate and select python.exe. To find the Python executable on Windows:
 - a. Search for *python.exe* in the Start menu.
 - b. Right-click on *python.exe* and select **Open file location**. This will open the directory containing the Python executable under the Start menu.
2. **Enter the installed Python version:** In the Python Version text field, enter the installed Python version (e.g., if you installed Python version 3.8, enter 3.8).
3. **Set up the Python Module Path:** At the bottom left of the panel, click on the file icon for the Python Module Path. Locate and select the provided compiled StellarNet Python driver that is compatible with the installed Python version, operating system, and its bitness. After selecting the driver file, remove the .pyd extension from the file name.
 - a. Tested on the 64 bits Windows with LabVIEW 32 bits, Python 32-bit, Python version 3.8, and stellarnet_driver3.cp38-win_amd64.
4. **Save default Python version and paths:** To ensure Python version and default paths are saved for future use, right-click on each field, expand "Data Operations," and select "Make Current Value Default" so it will be automatically loaded next time.



5. **Run the program:** Click on the Run button at the top left corner. The program will use the current settings to configure the spectrometers and continuously capture data in a loop. Click the STOP button above the spectrum graph to stop the loop.

LabVIEW Block Diagram





PYTHON API REFERENCE

- `array_get_spec(chan)`

Initiate Spectrometer device and setup parameter for acquisition. Return the spectrometer object and the spectrometer wavelength.

Parameters:

Name	Data Type	Required/Optional	Description
chan	Integer	Required	Spectrometer channel index. For example, index 0 represents the first spectrometer channel, and index 1 represents the second spectrometer channel. Indices go up to 127 for a total of 128 spectrometers.

Return:

- Spectrometer as *dict* with data type *str* as the key:
 - **Key:** 'device', **Value type:** *StellarNet* object
 - **Key:** 'config_id', **Value type:** *int*.
- *numpy.ndarray* for the wavelength data.

Example:

```
# 0 for first channel and 1 for second channel , up to 127 spectrometers
spectrometer, wav = sn.array_get_spec(0)
```



- `array_get_spec_only (chan)`

Initiate Spectrometer device and setup parameter for acquisition. Return the spectrometer object only. Exclude spectrum wavelength – called ‘getSpectrum_X’ to get the spectrum wavelength only.

Parameters:

Name	Data Type	Required/Optional	Description
chan	Integer	Required	Spectrometer channel index. For example, index 0 represents the first spectrometer channel, and index 1 represents the second spectrometer channel. Indices go up to 127 for a total of 128 spectrometers.

Return:

- Spectrometer as *dict* with data type *str* as the key:
 - **Key:** ‘device’, **Value type:** *StellarNet* object
 - **Key:** ‘config_id’, **Value type:** *int*.

Example:

```
spectrometer = sn.array_get_spec_only(0) #get channel 0 spectrometer object
wav = sn.getSpectrum_X(spectrometer)
```




- `array_spectrum(spectrometer, wav)`

Get spectrum data from the corresponding spectrometer. Return a 2D array of Calibrated Wavelength (X) and the spectrum Counts (Y).

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.
wav	numpy.ndarray	Required	Wavelength data

Return: A *numpy.ndarray* of spectrum array.

Example:

```
print(sn.array_spectrum(spectrometer, wav))
```

```
>>> [[ 339.13    1439.    ]
      [ 339.47636572 1412.    ]
      [ 339.8227929 1410.    ]
      ...
      [1175.87779313 1410.    ]
      [1176.3498241 1408.    ]
      [1176.82191653 1414.    ]]
```



- `ext_trig(spectrometer, triggerval)`

Enable or disable the external trigger for the selected spectrometer device. The timeout function will be disabled if the variable `triggerval` is set to `True`. When `triggerval` is `True`, the timeout will be ignored, which may result in infinite waiting if the spectrometer gets disconnected or if data does not return for some reason. When `triggerval` is `False` (the default), a default timer will be set based on the integration time and the number of scans to average. If the data does not return after the timer completes, an error will be thrown to inform the user.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.
triggerval	Boolean	Required	Status for the Ext Trigger to be set. <i>True</i> of no timeout and <i>False</i> for with timeout.

Return: NONE

Example:

```
sn.ext_trig(spectrometer, False)
```



- `getSpectrum_X(spectrometer)`

Get the wavelength of the spectrometer. Exclude Y axis of the spectrum data – called ‘getSpectrum_Y’ to get the spectrum data.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: ‘device’, Value type: <i>StellarNet</i> object. Key: ‘config_id’, Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *list* of the X (wavelength) axis of spectrum data

Example:

```
spectrometer = sn.array_get_spec_only(0)
wav = sn.getSpectrum_X(spectrometer)
```



- `getSpectrum_Y(spectrometer)`

Get spectrum data from the spectrometer. Exclude spectrum wavelength – called ‘getSpectrum_X’ to get the spectrum wavelength.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: ‘device’, Value type: <i>StellarNet</i> object. Key: ‘config_id’, Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *list* of the Y axis of spectrum data

Example:

```
spectrometer = sn.array_get_spec_only(0)
spec_y = sn.getSpectrum_Y(spectrometer)
```



- `getDeviceId(spectrometer)`
Returns the spectrometer device ID.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *str* of device ID.

Example:

```
print(sn.getDeviceParam(spectrometer))  
  
>>>22072014
```



- `getDeviceParam(spectrometer)`

Returns the current spectrometer device parameters.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *dictionary* of spectrometer device parameters.

Example:

```
print(sn.getDeviceParam(spectrometer))

>>>{'int_time': 50, 'x_timing': 3, 'x_smooth': 0,
'scans_to_avg': 1, 'temp_comp': False, 'coeffs': [0.69267,
0.0001229, 339.13, 0.0], 'det_type': 1, 'model': 'BW ZAP',
'device_id': '22072014'}
```



- `setParam(spectrometer, inttime, scansavg, smooth, xtiming, clear =True)`
Configure the spectrometer device parameters, such as integration time, scans to average, smoothing, and timing. It is recommended to set the `clear` parameter to `True` to ensure that the next scan of the spectrum data is valid. When `clear = True`, the function will perform a scan with the new parameters, and this scan will be ignored as it might contain invalid data due to the updated parameters. Setting `clear = True` is equivalent to setting `clear = False` and then calling `array_spectrum` or `getSpectrum_Y` to capture the data.

Parameters:

Name	Data Type	Required/ Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.
inttime	Integer	Required	Integration time
scansavg	Integer	Required	Number of scans to average
smooth	Integer	Required	Degree of smoothing
xtiming	Integer	Required	X timing
clear	Boolean	Required	Throw away for spectrum data after the device update. *It is recommended to set it to <i>True</i> if integration time and/or Xtiming is changed.

Return: None

Example:

```
# Integration time = 100, Scans to average = 4, smooth = 2,
Xtiming = 3
sn.setParam(spectrometer, 100,4,2,1)
```



Parameters Info

- **Detector integration time:**

This should be adjusted for each experiment to maximize the detector output and signal to noise ratio. The integration time is reported in the status bar message at the bottom of the graph (as Time: xx milliseconds). The toolbar slide bar can be used to dynamically set the appropriate level without saturating at the graph top.

- **Number of scans to average: (1...)**

Sets the number of spectra to signal average. Please note that the real-time display is updated AFTER these numbers of spectra have been acquired. This option provides a smoothing in the Y-axis, effectively increasing the system signal to noise by the square root of the number of scans being averaged. Set the averaging to the highest number

- **Smoothing level 0, 1, 2, 3, or 4**

This performs data smoothing by applying a moving average of adjacent pixels to the data arrays. For example, a Pixel Boxcar setting of 1 would average 5 total pixels: 2 pixels on the left, 2 pixels on the right, and one in the center. 0 performs NO data smoothing.

Smoothing Setting	Total pixels averaged
0	0
1	5
2	9
3	17
4	33

- **Temperature compensation: True=ON**

The first 15 "optical black" pixels that are read from the detector, except Hamamatsu and InGas, provide a useful level of the sensor dark current even when the detector is illuminated. If this option is on, the system periodically samples this area and makes an adjustment to the normal readout. When the level raises or lowers with temperature, the scan is adjusted accordingly.



- **XTiming resolution control: 1/2/3**

This feature provides increased optical resolutions. Selection 1 is the lowest optical resolution and is synchronized with the selected detector integration. In general, if your requirements for optical resolution are greater than 1nm, then selection 1 is ok. Selection 2 or 3 slows the digitizer & detector clock by a factor of 2 and 4 respectively. With XT levels 2 & 3 you will be able to observe increasingly higher resolutions. The detectors signal amplifier improves with slower throughput. When selecting XT level 2/3 the detector integration time must be increased to 30ms or longer to avoid sync delays.



- `setTempComp(spectrometer, temp_comp)`

Set temperature compensation to reflect on the returned on the spectrum data. There are 15 “optically black” pixels on StellarNet’s SONY ILX-511b CMOS detectors which are not hit by light during an acquisition. They provide a continuous measurement of the average dark spectrum and can be used to adjust for baseline drift during an experiment. In other words, this feature compensates for changes in the baseline due to temperature.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: ‘device’, Value type: <i>StellarNet</i> object. Key: ‘config_id’, Value type: <i>int</i> .	Required	Spectrometer device and configuration information.
temp_comp	Boolean	Required	Turn on or off the temperature compensation calculation

Return: None

Example:

```
sn.setTempComp(spectrometer, True) # Enable temperature compensation  
  
sn.setTempComp(spectrometer, False) # Disable temperature compensation
```

Note: This feature is not useful for Hamamatsu CMOS detectors as these do not have dark pixels.



- `getBurstFifo_Y(spectrometer)`

Burst Mode requires a zAP1 board upgrade with a large FIFO. When operating in Burst Mode, 128 consecutive spectra are collected into the large FIFO at the configured integration time and then sent as one packet over USB. This feature is recommended for integration times shorter than 30-50ms as real time acquisition is compromised by the data transfer polling rate of USB hardware. The polling rate is approximately 30Hz but can vary depending on hardware and operating system conditions. Burst mode is NOT recommended for longer integration times. Scans to average will not be applied for any Burst Mode capture.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *list of tuples* spectrum data. The list will contain a total of 128 tuples and each tuple stored the retrieved spectrum data.

Example:

```
Burst_data = sn.getBurstFifo_Y(spectrometer)
```

Note:

Burst Mode is only available in zAP1 boards.

Burst mode is not available in standard board configurations.



- `allowBurst(spectrometer)`

Retrieve whether burst mode is allowed for the spectrometer.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *boolean* value indicating whether burst mode is allowed for the spectrometer.

Example:

```
sn.allowBurst(spectrometer)
```

- `total_device_count()`

Retrieve the total number of connected StellarNet spectrometers.

Parameters: None

Return: An *integer* representing the total count of the connected StellarNet spectrometers.

Example:

```
sn.total_device_count()
```



- `getFullDeviceID(spectrometer)`
Retrieve full spectrometer device ID.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *string* representing the full spectrometer device ID.

Example:

```
sn.getFullDeviceID(spectrometer)
```



- `deviceConnectionCheck(spectrometer)`

Retrieve whether the connected spectrometer is still connected.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: A *boolean* indicating whether the connected spectrometer is still connected.

Example:

```
sn.deviceConnectionCheck(spectrometer)
```



- **reset(spectrometer)**

Release the spectrometer object. It is recommended to release the spectrometer object at the end of the program to release the USB resource. You only need to call this function when you are done with the spectrometer.

Parameters:

Name	Data Type	Required/Optional	Description
spectrometer	Dictionary: Key: 'device', Value type: <i>StellarNet</i> object. Key: 'config_id', Value type: <i>int</i> .	Required	Spectrometer device and configuration information.

Return: None

Example:

```
sn.reset(spectrometer)
```

- **version()**

Keep track of the version for the StellarNet Inc USB Driver as well as the latest compiled date.

Parameters: NONE

Return: A *str* that contains the version and latest compiled date information.

Example:

```
print(sn.version)  
  
>>>'StellarNet Inc USB Driver - Version 2.5 Compiled on 9/9/2024'
```



- **installDeviceDriver ()**

This function only applies to the Windows Operating System to avoid a conflict with the SpectraWiz software. This function only needs to be run ONCE when you run the script for the first time. When you are ready to switch back to the SpectraWiz, make sure you run [uninstallDeviceDriver\(\)](#) to uninstall the device driver.

- After this function is executed, a popup window titled "Device Driver Installation Wizard" will show up. Click "Next" to install it, and then click "Finish".

Note:

- To utilize this function, the script MUST be running in Administrator mode.
- This function must be called at the beginning of the script to enable the device driver.
- If your computer has some type of anti-virus software, this function might be blocked, either allow the access from the anti-virus software or a manual install of the device driver (or Filter Wizard) will be needed.

Parameters: NONE

Return: None

Example:

```
sn.installDeviceDriver()
```




- **uninstallDeviceDriver()**

This function is for the Windows Operating System with the installed SpectraWiz software only. This function will avoid the conflict of the USB utility for the programming script with the SpectraWiz, so uninstalling the device driver using this function will allow the use of the SpectraWiz Software. This function should be called whenever you are ready to use the SpectraWiz software after the programming.

Note:

- To utilize this function, the script MUST be running in the Administrator mode.

Parameters: NONE

Return: NONE

Example:

```
sn.uninstallDeviceDriver()  
>>>Microsoft PnP Utility  
  
Driver package uninstalled.  
Driver package deleted successfully.  
done
```



- **Class StellarNet(object)**

StellarNet spectrometer class: Most of the methods above can also be performed directly by calling the class member methods in the StellarNet instance.

- **__init__(self, device)**
Class constructor. Prepares spectrometer device for use.
- **__del__(self)**
Class destructor. Releases spectrometer device resources.
- **extrig (self, trigger)**
Set External Trigger status for the spectrometer device.
Parameters:

Name	Data Type	Required/Optional	Description
trigger	Boolean	Required	Ext Trigger status. <i>True</i> for no timeout and <i>False</i> for with timeout.

Return: NONE

Example:

```
spectrometer['device'].extrig(True)           //Disable Timer  
spectrometer['device'].extrig(False)          //Enable Timer
```



- **set_config(self, **kwargs)**

Set and configure the the spectrometer device, includes the integration time, smoothing, scans to average, X timing and temperature compensation.

Parameters:

Name	Data Type	Required/Optional	Description
int_time	Integer	Optional	The integration time in milliseconds.
x_timing	Integer	Optional	The XTiming rate.
x_smooth	Integer	Optional	The boxcar smoothing window size.
scans_to_avg	Integer	Optional	The number of scans to be averaged together.
temp_comp	Integer	Optional	Temperature compensation.

Return: NONE

Example:

```
spectrometer['device'].set_config(int_time = 100, scans_to_avg = 1)
```



- **get_config(self)**
Gets the current spectrometer device configuration, includes the integration time, smoothing, scans to average, X timing and temperature compensation.

Parameters: NONE

Return:

- A *dict* of the device configuration information.

Example:

```
config_info = spectrometer['device'].get_config()
print(config_info)

>>> {'int_time': 480, 'x_timing': 1, 'x_smooth': 0, 'scans_to_avg': 1,
'temp_comp': 0, 'coeffs': [0.69267, 0.0001229, 339.13, 0.0], 'det_type': 1,
'model': 'VIS 50', 'device_id': '18011639'}
```



```
#Get calebration coefficients info only
coeffs_info = spectrometer['device'].get_config()['coeffs']
print(coeffs_info)

>>> [0.69267, 0.0001229, 339.13, 0.0]
```



- **get_device_id(self)**
Gets the spectrometer device ID in string.

Parameters: NONE

Return: A *str* that represents the device id.

Example:

```
device_id = spectrometer['device'].get_device_id()
print(device_id)
>>> 18011639
```

- **read_spectrum(self)**
Reads and returns a spectrum from the spectrometer.

Parameters: NONE

Return: A *tuple* of short integers

Example:

```
spectrum_info = spectrometer['device'].read_spectrum()
print(spectrum_info)
>>> (1345, 1314, 1312, 1328, 1344, 1331, 1324, 1310, 1320, 1312, 1312,
1309, 1303, 1312, 1312, ..., 1347, 1345, 1347, 1344, 1343, 1340, 1330,
1320, 1331, 1325, 1316, 1317, 1333)
```

Note: Class [StellarNet.set_config\(\)](#) for a description of the parameters that control the operation of the spectrometer or the post-processing of the spectrum.



- **compute_lambda(self, pixel)**
Compute lambda from the pixel index.

Parameters:

Name	Data Type	Required/Optional	Description
pixel	Integer	Required	The pixel index on which to perform the computation.

Return: A *float* represents the pixel's wavelength.

Example:

```
lambda_value = spectrometer['device'].compute_lambda(0)
print(lambda_value)
>>>339.13
```



- **print_info(self)**
Print spectrometer device information, i.e. idVendor, idProduct, iManufacturer, and Stored Strings, etc.

Parameters: NONE

Return: NONE

Example:

```
spectrometer['device'].print_info()

>>> --- Device Information
idVendor:    OBD7
idProduct:   A012
iManufacturer: 'StellarNet'
iProduct:    'USB2EPP'

--- Stored Strings:
00 '°x
01 '
02 '°x
03 '
04 '
05 '
06 '°x
07 '
08 '°x
09 '
0A '°x
0B '°x
0C '°x
0D '°x
0E '°x
0F '°x
10 '°x
11 '°x
12 '°x
13 '°x
14 '°x
15 '°x
16 '°x
17 '°x
18 '°x
19 '°x
1A '°x
1B '°x
1C '°x
1D '°x
1E '°x
1F '°x
20 'VIS 50 #18011639f2g1
21 '
22 '
23 '
24 '
25 '
26 '
27 '
28 '
29 '
2A '
2B '
2C '
2D '
2E '
2F '
30 '°x
31 '°x
32 '°x
33 '°x
34 '°x
35 '°x
36 '°x
37 '°x
38 '°x
39 '°x
3A '°x
3B '°x
3C '°x
3D '°x
3E '°x
3F '°x
40 '
41 '
42 '
43 '
44 '
45 '
46 '
47 '
48 '
49 '
4A '
4B '
4C '
4D '
4E '
4F '
50 '°x
51 '°x
52 '°x
53 '°x
54 '°x
55 '°x
56 '°x
57 '°x
58 '°x
59 '°x
5A '°x
5B '°x
5C '°x
5D '°x
5E '°x
5F '°x
60 '°x
61 '°x
62 '°x
63 '°x
64 '°x
65 '°x
66 '°x
67 '°x
68 '°x
69 '°x
6A '°x
6B '°x
6C '°x
6D '°x
6E '°x
6F '°x
60 '°x
61 '°x
62 '°x
63 '°x
64 '°x
65 '°x
66 '°x
67 '°x
68 '°x
69 '°x
6A '°x
6B '°x
6C '°x
6D '°x
6E '°x
6F '°x
70 '°x
71 '°x
72 '°x
73 '°x
74 '°x
75 '°x
76 '°x
77 '°x
78 '°x
79 '°x
7A '°x
7B '°x
7C '°x
7D '°x
7E '°x
7F '°x
80 '0.6926700000000000 1'
81 '
82 '
83 '
84 '
85 '
86 '
87 '
88 '
89 '
8A '
8B '
8C '
8D '
8E '
8F '
80 '0.6926700000000000 1'
81 '
82 '
83 '
84 '
85 '
86 '
87 '
88 '
89 '
8A '
8B '
8C '
8D '
8E '
8F '
90 '0.000122900000 0'
91 '
92 '
93 '
94 '
95 '
96 '
97 '
98 '
99 '
9A '
9B '
9C '
9D '
9E '
9F '
90 '0.000122900000 0'
91 '
92 '
93 '
94 '
95 '
96 '
97 '
98 '
99 '
9A '
9B '
9C '
9D '
9E '
9F '
A0 '0.000122900000 0'
A1 '
A2 '
A3 '
A4 '
A5 '
A6 '
A7 '
A8 '
A9 '
AA '
AB '
AC '
AD '
AE '
AF '
A0 '0.000122900000 0'
A1 '
A2 '
A3 '
A4 '
A5 '
A6 '
A7 '
A8 '
A9 '
AA '
AB '
AC '
AD '
AE '
AF '
B0 '0.000122900000 0'
B1 '
B2 '
B3 '
B4 '
B5 '
B6 '
B7 '
B8 '
B9 '
BA '
BB '
BC '
BD '
BE '
BF '
B0 '0.000122900000 0'
B1 '
B2 '
B3 '
B4 '
B5 '
B6 '
B7 '
B8 '
B9 '
BA '
BB '
BC '
BD '
BE '
BF '
C0 '339.130
C1 '
C2 '
C3 '
C4 '
C5 '
C6 '
C7 '
C8 '
C9 '
CA '
CB '
CC '
CD '
CE '
CF '
C0 '339.130
C1 '
C2 '
C3 '
C4 '
C5 '
C6 '
C7 '
C8 '
C9 '
CA '
CB '
CC '
CD '
CE '
CF '
D0 '339.130
D1 '
D2 '
D3 '
D4 '
D5 '
D6 '
D7 '
D8 '
D9 '
DA '
DB '
DC '
DD '
DE '
DF '
D0 '339.130
D1 '
D2 '
D3 '
D4 '
D5 '
D6 '
D7 '
D8 '
D9 '
DA '
DB '
DC '
DD '
DE '
DF '
E0 '0.0000000000000000 4'
E1 '
E2 '
E3 '
E4 '
E5 '
E6 '
E7 '
E8 '
E9 '
EA '
EB '
EC '
ED '
EE '
EF '
E0 '0.0000000000000000 4'
E1 '
E2 '
E3 '
E4 '
E5 '
E6 '
E7 '
E8 '
E9 '
EA '
EB '
EC '
ED '
EE '
EF '
F0 '0.0000000000000000 4'
F1 '
F2 '
F3 '
F4 '
F5 '
F6 '
F7 '
F8 '
F9 '
FA '
FB '
FC '
FD '
FE '
FF '
F0 '0.0000000000000000 4'
F1 '
F2 '
F3 '
F4 '
F5 '
F6 '
F7 '
F8 '
F9 '
FA '
FB '
FC '
FD '
FE '
FF '

```



VERSION UPDATE

➤ Version 2.6

- ✓ T3 detector support update
- ✓ Additional methods:
 - allowBurst()
 - total_device_count()
 - getFullDeviceID()
 - deviceConnectionCheck()
- ✓ Raspberry pi (64 bits):
 - Supports Python versions 3.7 to 3.12
- ✓ Raspberry pi (32 bits):
 - Supports Python versions 3.7 to 3.8 and 3.10 to 3.11
 - Python versions 3.4 to 3.6 have been removed
- ✓ Mac
 - Supports Python versions 3.7 to 3.12

➤ Version 2.5

- ✓ Hot pixel mapping
- ✓ Easy access to device Count, device ID, device parameters
- ✓ Get wavelength (X) and Spectrum (Y) separately as a single list
- ✓ Set and retrieve device parameters
- ✓ Burst mode for CCD and CMOS
- ✓ Auto-driver install and uninstall on Windows
- ✓ Temperature compensation
- ✓ Support for more Operating Systems, bitness, and Python Versions

LICENSE

Copyright 2024 StellarNet, Inc.

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.