StellarNet Python Spectrometry Driver and webserver

This set of Python scripts allows StellarNet USB spectrometers to be accessed by applications running on Linux-based and other computers, including embedded machines. The code is compact, has minimal dependencies and is readily callable by applications written in Python and other languages. Up to 127 spectrometers can be connected to a single Linux computer.

Also included is a microserver implementation which allows fast control and access to JSON-formatted data by applications on the same machine or across the LAN or Internet. A RESTful API allows multiple users to access multiple spectrometers in parallel using simple HTML formats.

Though the driver and microserver have been tested on several Linux distributions (and are documented here for Mint Linux, a popular fork of Ubuntu which is in turn a fork of Debian Linux), the project should be transportable to other platforms as well.

Among the features of these implementations are a numeric configuration ID that increments whenever the parameters of a spectrometer are changed (essential for multiuser or multi-process applications) and a timestamp in msec for each spectrum returned (helpful for process control techniques such as PID). The driver and micro server operate in non-privileged userspace without administrative permissions.

Included are some executable GUI examples written in LabVIEW for use as a local application on the same machine on which the server is running or as remote applications over the LAN or Internet.

Installation

Installation on the Linux computer requires a few elements that can be downloaded from your distribution's repositories using apt-get, Synaptic or other package manager.

This section is applicable both where applications to be run on the same machine to which the spectrometers are connected *or* where applications are to be run remotely, accessing the spectrometers via a network.

We recommend proceeding through the following installation check-list sequentially. If any element is unfamiliar to you, explore its help or man pages in a terminal (e.g., issue `screen -h` or `man screen`).

Install Python

You need Python 2.7.xx. (Python 3 may not be compatible.) To check the version installed: in terminal, issue

```
python --version
```

Install a newer version from your distribution's repositories if necessary.

Install python-pip

This is an installer for Python packages. Install from the repos.

Install curl

curl is a command-line utility for dealing with URL interactions. Install from the repos.

Install screen

screen is a useful utility for logging and backgrounding process activity. Install from the repos.

Install pyusb

Install the correct version via the following terminal command:

```
sudo pip install pyusb==1.0.0a3
```

Install StellarNet files

Create a directory in your userspace. For the purposes of this documentation we'll call this directory "spectrometer" but the name is unimportant. Copy the following StellarNet files as indicated below:

File	Location	Comment
stellarnet.py	spectrometer	
stellarnet.hex	spectrometer	
teststellarnet.py	spectrometer	
spectroweb.py	spectrometer	Issue `sudo chmod -x spectroweb.py`
99-local.rules	/etc/udev/rules.d/	You will need to sudo to copy to this directory

Then restart the machine:

```
reboot
```

Once restarted:

Install python-flask

python-flask is a highly optimized web framework which provides the web API. Install from the repos.

Install & Configure python-virtualenv

python-virtualenv is a virtual instance manager for Python. Install from the repos.

Once python-virtualenv is installed, configure it with the following terminal commands:

```
cd spectrometer

virtualenv venv

. venv/bin/activate

pip install Flask

pip install pyusb==1.0.0a3
```

Testing the Python driver

You can interact with the Python driver directly from the command line. This is useful for validating the installation.

Plug in one or more spectrometers to the computer's USB ports. (If not bus powered, power them up.)

From a terminal, issue

```
python stellarnet.py info
```

This should return something like

```
Multiple devices, select one with -d option:
  10042624, auto_id0, auto_id1
```

…this indicates the serial number of connected devices with parameters stored in flash RAM, and the automatically-assigned identifiers for spectrometers lacking flash RAM.

Pick one of these devices and run some inquiries:

```
python stellarnet.py -d 10042624 info
```

```
--- Device Information
idVendor:      0BD7
idProduct:     A012
iManufacturer: 'StellarNet'
iProduct:      'USB2EPP'
--- Stored Strings:
00 '??
      ??????????????????????????????'
20 'BLUE-Wave VIS-14 #10042624      '
40 '                                '
60 '????????????????????????????????'
80 '0.691290000000                 1'
A0 '0.000129400000                 0'
C0 '315.660                         '
E0 '0.000000000000                 4'
```

Each command-line option has its own help screen which may be called with the `-h` option, for example:

```
python stellarnet.py -h
```

```
usage: stellarnet.py [-h] [-d DEVICE] {info,plot,perf} ...

Exercise StellarNet spectrometer.

positional arguments:
  {info,plot,perf}
    info                print device information (default)
    plot                plot spectrum
    perf                run performance test

optional arguments:
  -h, --help            show this help message and exit
  -d DEVICE, --device DEVICE
                        select device (default: None)
```

```
python stellarnet.py info -h
```

```
usage: stellarnet.py info [-h] [-l]

optional arguments:
  -h, --help  show this help message and exit
  -l, --list  list available devices
```

Starting the Web server

In everyday usage, you can start the web server by issuing

```
cd stellarnet

. venv/bin/activate
```

Then, if logging is desired (mind your disk space!), issue

```
screen -L ./spectroweb.py
```

If no logging is desired, issue

```
screen ./spectroweb.py
```

…The use of screen allows the microserver process to operate in the background and the terminal window to be closed without killing the process.

Alternatively, you can issue `./spectroweb.py&` to invoke the microserver as a background process, and then issue `jobs` to identify the backgrounded job number; issuing `disown %jobnumber` will then disassociate the job from the terminal window, allowing the window to be closed.

To test, in a second terminal window, issue

```
curl -i http://localhost:5000/spectrometers
```

Stopping the Microserver

It is necessary to stop and restart the microserver if a spectrometer is plugged or unplugged.

If you used screen and the terminal window in which you activated the server has remained open, ctrl-a d should allow you to close the window; ctrl-c will kill the webserver.

Or, in a terminal, list the currently active processes by issuing

```
ps ax
```

Identify the spectroweb.py process number in the list, and kill it by issuing

```
kill processnumber
```

Webserver REST API

The web server accepts JSON-formatted requests and generates JSON-formatted responses to allow access to connected spectrometers via an IP address. The default port is 5000. Supported methods are illustrated in the following table. Commands are sent as HTTP 1.1 formatted strings.

Purpose	Command	Example
Enumerate connected spectrometers	/spectrometers	GET /spectrometers
JSON response:	…/<device_ID>/…	
Get/set coefficients & parameters for a spectrometer	/spectrometers/*ID*/config	GET /spectrometers/1234/config
JSON response:	Specified device's coefficients, detector type, device ID, model, and current operating parameters: integration time, averaging, temperature compensation, boxcar smoothing, Xtiming	
		PUT /spectrometers/10042624/config HTTP/1.1 Content-Length: 61 {"int_time":102,"x_timing":3,"x_smooth":1,"scans_to_avg":3}
JSON response:	<configuration_ID>	
Get spectrum from a spectrometer	/spectrometers/*ID*/spectrum	GET /spectrometers/1234/spectrum
JSON response:	JSON formatted spectral data with current configuration ID and timestamp (msec, Unix time http://en.wikipedia.org/wiki/Unix_time)	

Note: timestamp accuracy depends on your system clock and its setting. For Internet-connected systems, installing and enabling ntp is recommended:

Examples:

Send

Receive

```
GET /spectrometers/10042624/config HTTP/1.1
Host: 192.168.0.8
```

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 247
Server: Werkzeug/0.9.4 Python/2.7.4
Date: Mon, 11 Nov 2013 20:58:38 GMT

{
  "coeffs": [
    0.69129,
    0.0001294,
    315.66,
    0.0
  ],
  "det_type": 1,
  "device_id": "10042624",
  "int_time": 102,
  "model": "BLUE-Wave VIS-14",
  "scans_to_avg": 1,
  "temp_comp": 0,
  "x_smooth": 0,
  "x_timing": 3
}
```

```
PUT /spectrometers/10042624/config HTTP/1.1
Content-Length: 61

{“int_time":102,"x_timing":3,"x_smooth":1,"scans_to_avg":3}
```

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 20
Server: Werkzeug/0.9.4 Python/2.7.4
```

```
Date: Mon, 11 Nov 2013 21:05:40 GMT

{
  "config_id": 6
}
```

```
GET /spectrometers/10042624/spectrum HTTP/1.1
Host: 192.168.0.8
```

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 22595
Server: Werkzeug/0.9.4 Python/2.7.4
Date: Mon, 11 Nov 2013 21:11:21 GMT

{
  "config_id": 6,
  "data": [
    2547,
    2553,
    2551,
    […]
    2635,
    2605
  ],
  "timestamp": 1384204281901
}
```

Calculating the wavelength for a pixel

The pixel-number to wavelength calculation is straightforward and utilizes the spectrometer coefficients:

$$\text{lambda} = (c4 * i^3/8) + (c2 * i^2/4) + (c1 * i/2) + c3;$$

where i = pixel number = 0..2047