Python-Neuroshare Documentation

Release

Christian Kellner

June 05, 2014

Contents

The Neuroshare API is a standardized interface to access electrophysiology data stored in various different file formats. To do so, it uses format- specific shared libraries. Refer to the official website

http://neuroshare.org

for more information.

The aim of this library is to provide a high level interface to the Neuroshare API, i.e. it focuses on API usability more then being a mere python version of the C API. Thus none of the original Neuroshare API calls are directly exposed but the interface consists of python objects that resemble (more or less) the Neuroshare Entities.

CHAPTER 1

Installation

The compile-time requirements for python-neuroshare are the 'setuptools' and the Python development files and a working C compiler (clang or gcc) and NumPy. For Debian based distributions, e.g. Ubuntu, this can easily be done with:

```
$ sudo apt-get install clang python-setuptools \
                       python-dev python-numpy
```

After that, python-neuroshare is installed with the following command:

```
$ sudo python setup.py install
```

Additional runtime dependencies:

- The Neuroshare vendor DLLs for the specific data file(s)! Please refer to the following section for more information.

1.1 Installation of vendor DLLs

Python-neuroshare relies on the vendor specific DLLs to access data files. Therefore the specific DLLs for each type of file must be downloaded and installed into one of the following locations:

```
/usr/local/lib/neuroshare
/usr/lib/neuroshare
~/.neuroshare
```

A (possibly incomplete) list of the vendor specific DLLs can be obtained be obtained from the neuroshare website:

http://neuroshare.sourceforge.net/DLLLinks.shtml

Please note that you need the corresponding DLLs for your platform (e.g. Linux, 64-bit). If you find yourself in the situation that there is no DLL for your specific platform and you are either on a UNIX-like system you can use G-Node's very one nswineproxy component to use the Windows 32 bit DLLs. Please refer to the nswineproxy homepage for more information:

https://github.com/G-Node/nswineproxy

CHAPTER 2

Quickstart

Opening a file:

```
import neuroshare as ns
fd = ns.File ("NeuroshareExample.mcd")
```

Iterate over the entities in the file:

```
for entity in fd.list_entities():
    print entity.label, entity.entity_type
    ... do something else with entity ...
```

Access analog signal data:

```
analog1 = fd.entities[1] #open analog signal entity
data, times, count = analog1.get_data() #load data
```

data will contain the raw data, times the timepoints of each datapoint and count how many datapoints in data are actually continous.

CHAPTER 3

Reporting Bugs & Submitting Patches

Any bugs can and should be filed to the project's issue tracker at github:

https://github.com/G-Node/python-neuroshare/issues

CHAPTER 4

Contact & Support

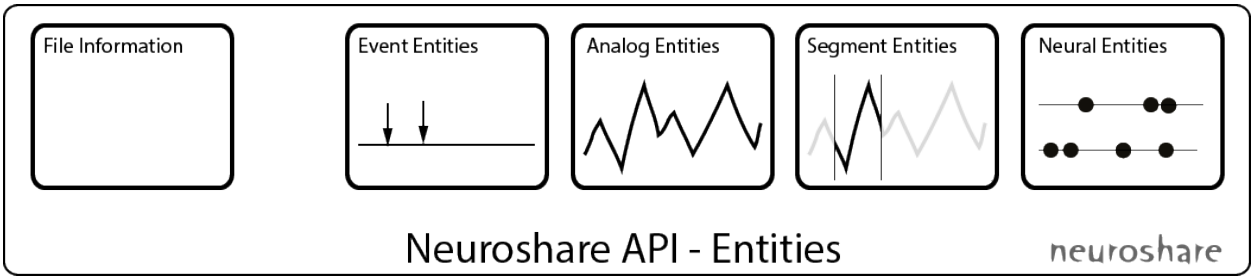
Support and discussion of python-neuroshare related questions happen in the official G-Node IRC channel #gnode on the freenode IRC network (cf. http://freenode.net).

CHAPTER 5

User's Guide

5.1 File structure and Entities

Neuroshare provides access to raw data and metadata (such as the sampling rate and creation date) via so called Entities, which groups data of the same type together. The standard defines 4 different entities: Events, Analog signals, Segments and Neural entities (i.e. spiketrains):



Event entities represent specific timepoints with associated data, e.g. trigger events.

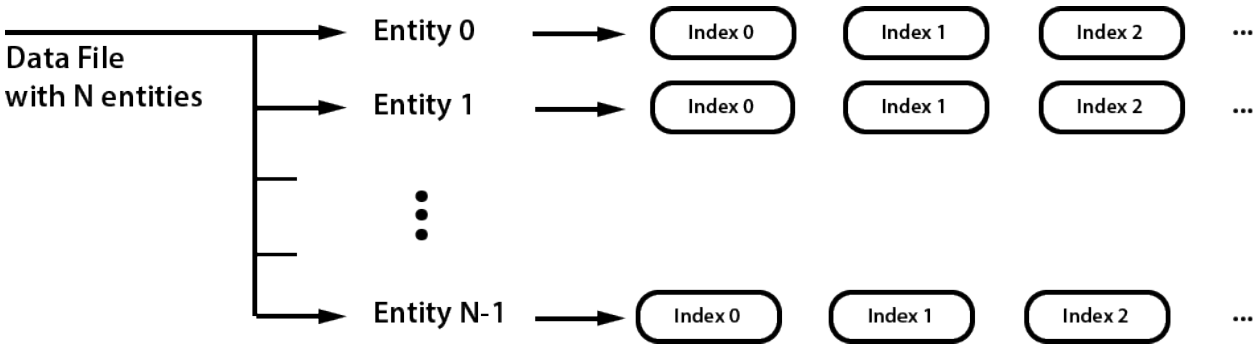
Analog signal entities represents continuously sampled, i.e. digitized, analog data. Examples are waveforms recorded via an electrode (microelectrodes, EKG, EEG).

Segment entities contain cutouts of continuously sampled analog signals from one or more sources that are usually short in time. Most prominent example are waveforms of action potentials from one ore more electrodes.

Neural entities are arrays of timestamps when action potentials happened, i.e. arrays of spike times.

5.2 Data access

All entities in the file are accessed by their entity index. Each individual entity can have one or more data entries attached to it; these are indetified by a sequential index.



5.3 API

The basic desgin of the API closely follows the Neuroshare entity model. For all 4 entities there is a class that represents that entity:

- `EventEntity` for Events
- `AnalogEntity` for Analog signal entities
- `SegmentEntity` for Segements
- `NeuralEntity` for Neural entities

All entity classes derive from a common `Entity` class that provides metadata common to all entites such as the label (`Entity.label()`) and how many data entries are contained in the entity (`Entity.item_count()` or just `len(entity)`).

Opening a file is simply done by creating a `neuroshare.File` object with the path to the datafile as constructor argument: `fd = neuroshare.File('data.mcd')`. Individual enitities can be accessed via the `File.get_entity()` function or via indexing through the `File.entities()` property (e.g. `File.entities[idx]`).

Data is accessed via the `get_data()` function that all 4 entities provide. Consult the documentation of the individual functions for details.

5.4 Code Examples

5.4.1 List entities in a file

How to list all entities in a file called *data.mcd*:

```
import neuroshare as ns
fd = ns.File('data.mcd')
for i, entity in enumerate(fd.entities):
    print('%04d: "%s" type: %d' % (i, entity.label, entity.entity_type))
```

This will produces the following output:

```
0000: "trig0001 0000 0000 trig0001" type: 1
0001: "elec0001 0000 0000       01" type: 2
0002: "elec0001 0001 0001       02" type: 2
0003: "elec0001 0002 0002       03" type: 2
[...]
```

5.4.2 Access the raw data inside an analog signal entity

To access the data and timestamps of an analog entity the `AnalogEntity.get_data()` is used:

```
analog1 = fd.entities[1]  #access the entity 1

#now load all the raw data
data, timestamps, cont_count = analog1.get_data()
```

The `data` value is a `3-tuple` which contains the raw data and the timestamps for each datapoint. It is also possible to retrieve a subset of the available data:

```
data = analog1.get_data(20, 10) #fetch 10 elements starting at index 20
print("%d" % data[0].shape)
# -> 10
print (data[0])
# ->
# [  8.50000000e-05   7.00000000e-05   2.16666667e-05   3.16666667e-05
#   3.66666667e-05   0.00000000e+00  -5.50000000e-05  -9.33333333e-05
#  -6.66666667e-05   3.33333333e-06]
```

5.4.3 Metadata

Metadata is exposed as python properties of the individual entities:

```
print(analog1.units)
# -> 'V'
print(analog1.sample_rate)
# -> 25000.0
```

CHAPTER 6

API Reference

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*