snowmicropyn

Release 1.0.1

Aug 06, 2018

Contents

A warmly welcome to the documentation of *snowmicropyn*, a python package to read, export and post process data (*.pnt files) recorded by SnowMicroPen, a snow penetration probe for scientifc applications developed at SLF.

For feedback and suggestions, please write to snowmicropen@slf.ch or use our Issue Tracker.

CHAPTER 1

What is where?

- Our Source Code Repository is on *GitHub*.
- This Documentation can be studied on *Read the Docs*.
- Our Releases are placed on *PyPI* (Use **`pip`** to install)

CHAPTER 2

Table of Content

2.1 Overview

2.1.1 What's inside?

The *snowmicropyn* package contains two entities:

- An *API* to automate reading, exporting and post processing pnt files using the python language. You'll need some basic programming skills to use it.
- *pyngui*, a desktop application to read, export and post process pnt files. **`pyngui`** uses the API itself too.

2.1.2 How do I get it?

Installing *snowmicropyn* is a trivial task in case you're experienced with Python:

```
pip install snowmicropyn
```

No clue what we're talking about? You find a more detailed description in section *Installation*!

2.1.3 *snowmicropyn*'s API

The following snippet is a simple example how to read a pnt file, read some of it's meta information and export its samples (measured distance & force) into a CSV file.

```
from snowmicropyn import Profile

p = Profile.load('S31M0067.pnt')

print(p.timestamp)  # Timestamp of recording
print(p.smp_serial)  # Serial number of SnowMicroPen used
```

(continues on next page)

(continued from previous page)

```
print(p.coordinates)  # WGS 84 (latitude, longitude)

# Export samples into CSV format
# (By default, filename will be :file:`S31M0067_samples.csv)
p.export_samples()
```

You find detailed information about the API in the *API User's Guide*. For more information about the API's elements, checkout the *API Reference*.

2.1.4 Launch `pyngui`

After installing *snowmicropyn* , open a Terminal Window and type `pyngui` and hit `return` to start the **`pyngui`**. Happy examining!

If you want to launch the **`pyngui`** manually, type:

```
python -m snowmicropyn.pyngui.app
```

2.2 Installation

2.2.1 Prerequisite: Python 3

snowmicropyn is written in the Python programming language. You need to have Python 3.4 (or later) installed on your machine to run *snowmicropyn*.

To check if your computer has Python installed, open a Terminal Window and type

```
python --version
```

Note: When both, Python 2 and Python 3 is installed on your computer, usually you run Python 2 by the command **`python`** and Python 3 by the command **`python3`**.

If you have Python installed, you'll get a version string returned. In case you get a response like "`command not found`" or a version smaller than 3.4.x, you have to install or update Python.

How to install Python? Download an official package or follow the instructions from the Python Guide.

Make sure you install the latest Python 3 release.

2.2.2 Installing *snowmicropyn*

So you managed to install Python 3 on your computer. Well done! Now, by using the **`pip`** command (which is part of Python), the installation of *snowmicropyn* is a peace of cake. Open a terminal window and type:

```
pip install snowmicropyn
```

Note: When both, Python 2 and Python 3 is installed on your computer, you may need to type **`pip3`** instead of **`pip`**.

This will install the latest version of *snowmicropyn* available on PyPI and its dependencies. In case you want to install a specific version of snowmicropyn, append it to the package name as in this example:

```
pip install snowmicropyn==0.1.3
```

That's about it. We hope you managed to get *snowmicropyn* on your machine.

Hint: You may consider using a virtual environment to separate your *snowmicropyn* installation from other projects. But that's already an more advanced topic.

Tip: A good place to start getting into Python is the Python Guide.

2.3 Upgrading

In case you installed *snowmicropyn* before and like to upgrade to the latest version available, execute the following command:

```
pip install snowmicropyn --upgrade --no-cache-dir
```

2.4 Uninstalling

Get rid of *snowmicropyn* is simple too:

```
pip uninstall snowmicropyn
```

2.5 API User's Guide

2.5.1 Data Files

When performing a measurement with SnowMicroPen, the device writes the data onto its SD card in a binary file with a `pnt` extension. (Example: `S13M0067.pnt`). For each measurment process, a new pnt file is written. Each pnt file consists of a header with meta information followed by the actual data, the force samples.

Note: The *snowmicropyn* package never ever writes into a pnt file. Good to know your precious raw data is always save.

Corresponding ini files

However, when using functionality of this package, an additional storage to save other data is required. This storage is an `ini` file, named like the pnt file (Example from section before: `S13M0067.ini`).

2.5.2 First steps

The core class of the API is the *snowmicropyn.Profile* class. It represents a profile loaded from a pnt file. By using its static load method, you can load a profile:

```
import snowmicropyn
p = snowmicropyn.Profile.load('./S13M0067.pnt')
```

In the load call, there's also a check for a corresponding ini file, in this case for the `S13M0067.ini`.

2.5.3 Logging *snowmicropyn*'s Version and Git Hash

As a scientist, you may interested to keep a log so you can reproduce what you calculated with what version of *snowmicropyn*. The package contains a version string and a git hash identifier.

To access the packages version string, you do:

```
import snowmicropyn
v = snowmicropyn.__version__
```

To access the git hash string of this release, you do:

```
import snowmicropyn
gh = snowmicropyn.githash()
```

When exporting data using this module, the created CSV files also will contain a comment as first line with version string and git hash to identify which version of *snowmicropyn* was used to create the file.

Warning: However, this is no mechanism to protect a file from later alternation. It's just some basic information which maybe will be useful to you.

2.5.4 Examples

Some examples will help you to get an overview of *snowmicropyn*'s features.

Hint: To get the code mentioned in this guide, Download the source code of *snowmicropyn*. You'll find the examples in the subfolder `examples` and even some pnt files to play around with in the folder `examples/profiles`.

Explore properties

In our first example, we load a profile and explore its properties. We set some markers and finally call the *snowmicropyn.Profile.save()* so the markers get save in a ini file so we don't loose them.

```
import logging
import sys

import snowmicropyn

# Enable logging to stdout to see what's going on under the hood
logging.basicConfig(level=logging.DEBUG, stream=sys.stdout)
```

(continues on next page)

(continued from previous page)

```
print(snowmicropyn.__version__)
print(snowmicropyn.githash())

p = snowmicropyn.Profile.load('profiles/S37M0876.pnt')

print('Timestamp: {}'.format(p.timestamp))
print('SMP Serial Number: {}'.format(p.smp_serial))
print('Coordinates: {}'.format(p.coordinates))

p.set_marker('surface', 100)
p.set_marker('ground', 400)
print('Markers: {}'.format(p.markers))

# We don't want to loose our markers. Call save to write it to an ini
# file named like the pnt file.
p.save()
```

Batch exporting

You're just home from backcountry where you recorded a series of profiles with your SnowMicroPen and now want to read this data with your tool of choise which supports reading CSV files? Then this example is for you!

```
import glob

from snowmicropyn import Profile

match = 'profiles/*.pnt'

for f in glob.glob(match):
    print('Processing file ' + f)
    p = Profile.load(f)
    p.export_samples()
    p.export_meta(include_pnt_header=True)
    p.export_derivatives()
```

After you executed this example, there will be a `..._samples.csv` and a `..._meta.csv` for each pnt file in the directory.

Plotting

In this example, we use the delicious matplotlib to explore the penetration signal of a profile.

```
from matplotlib import pyplot as plt

from snowmicropyn import Profile

p = Profile.load('profiles/S37M0876.pnt')

# Plot distance on x and samples on y axis
plt.plot(p.samples.distance, p.samples.force)

# Prettify our plot a bit
plt.title(p.name)
```

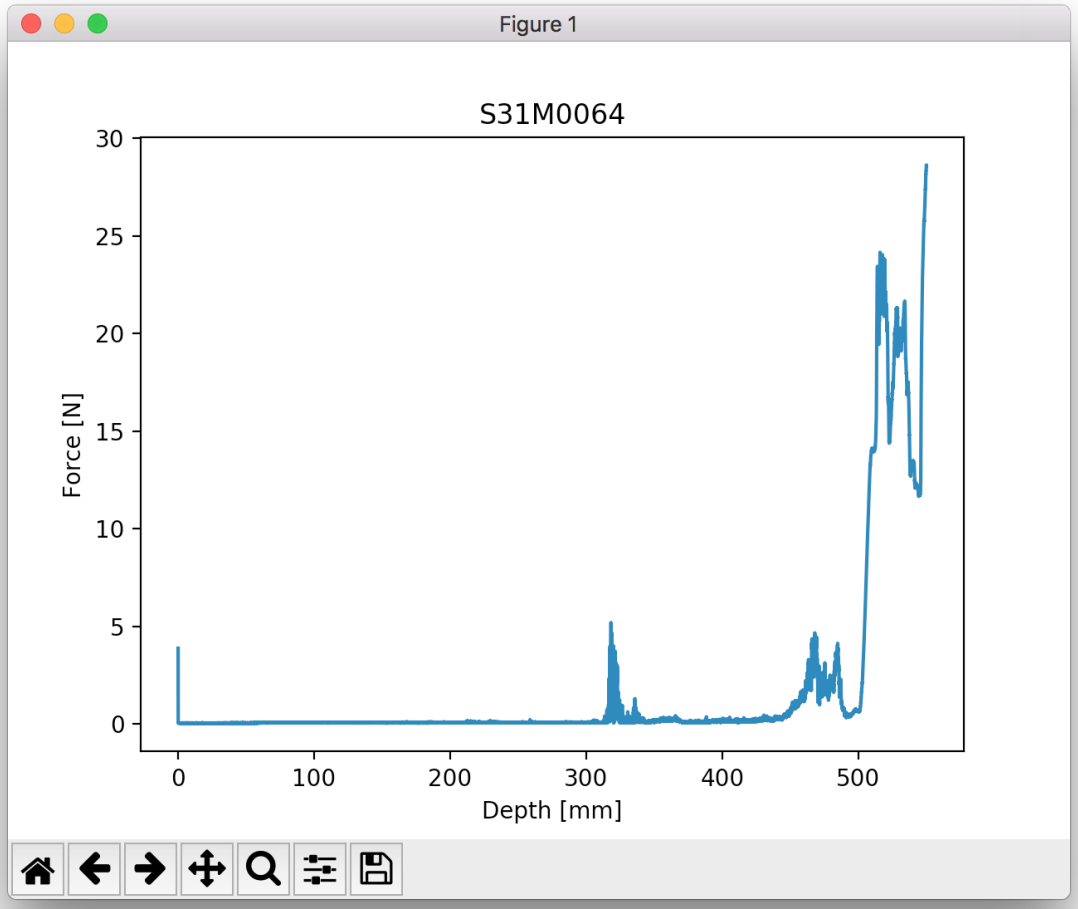
(continues on next page)

(continued from previous page)

```
plt.ylabel('Force [N]')
plt.xlabel('Depth [mm]')

# Show interactive plot with zoom, export and other features
plt.show()
```

When this code is executed, a window like to following should open:



Expore using the tool buttons below the plot! You can even modify the axes and export the plot into an image file.

A Touch of Science

Alright, let's do some science. In this example, we examine a profile recorded at our Testsite Weissfluhjoch. There's a crust and a depth hoar layer in this profile. By using command:*pyngui*, we already identified the layers for you by setting markers. Let's calculate the mean SSA within the crust and the weight that lies on the depth hoar layer.

```
from snowmicropyn import Profile
from snowmicropyn import proksch2015

p = Profile.load('profiles/S37M0876.pnt')
p2015 = proksch2015.calc(p.samples)

crust_start = p.marker('crust_start')
crust_end = p.marker('crust_end')

crust = p2015[p2015.distance.between(crust_start, crust_end)]

# Calculate mean SSA within crust
print('Mean SSA within crust: {:.1f} m^2/m^3'.format(crust.P2015_ssa.mean()))

# How much weight lies above the hoar layer?
surface = p.marker('surface')
hoar_start = p.marker('depthhoar_start')
above_hoar = p2015[p2015.distance.between(surface, hoar_start)]
weight_above_hoar = above_hoar.P2015_density.mean() * (hoar_start - surface) / 1000
print('Weight above hoar layer: {:.0f} kg/m^2'.format(weight_above_hoar))
```

This will print something like:

```
Mean SSA within crust: 5.5 m^2/m^3
Weight above hoar layer: 98 kg/m^2
```

2.6 API Reference

2.6.1 Receive Version Number and Git Hash

To receive the version of *snowmicropyn* and the git hash of *snowmicropyn* you're using, do the following:

```
import snowmicropyn
version = snowmicropyn.__version__  #  e.g. '0.1.2'
hash = snowmicropyn.githash()  # e.g. '55623b2d71e7cb7...'
```

Receiving and logging this is useful for tracking purposes. The version and the git hash are loggend also when you import the package (using python's standard logging facility). In case this information is crucial to you, it's important to do logging setup before importing the package, otherwise you miss it. The other option is to do the logging yourself using the function *snowmicropyn.githash()* and accessing `snowmicropyn.__version__`.

`snowmicropyn.`**`githash`**`()`
Get the git hash of this release of *snowmicropyn*.

The Hash is a string. It can be `None`, which means you're using a non official release of *snowmicropyn*.

2.6.2 Its Core: The `Profile` Class

`class` `snowmicropyn.`**`Profile`**(*pnt_file*, *name=None*)
Represents a loaded pnt file.

SnowMicroPen stores a recorded profile in a proprietary and binary format with a `pnt` file extension. A pnt file consists of a header with meta information and the recorded force measurement values. When a pnt file is loaded using this class, it reads this data. Meta information then can be accessed by many properties like *timestamp*

or *overload*. The measurement data is called "samples". Its accessed using the property *samples* or methods prefix with `samples_`.

The class supports the settings of "markers". They identified by name and mark a certain distance value on the profile. You can set markers, read marker values, and remove markers. The two *well known* markers called "surface" and "ground" are used to determine the snowpack. Markers are not stored in a pnt file. As a fact, the pnt file is always just read and never written by the *snowmicropyn* package. To store marker values, this class writes ini files (`*.ini`) named same as the pnt file (but with its ini file extension, of course). Use the method *save()* to save your markers.

When a profile is loaded, the class tries to find a ini file named as the pnt file. In case one is found, it's read automatically and your prior set markers are available again.

To improve readability of your code, your encouraged to load a profile using its static method *load()*. Here's an example:

```
import snowmicropyn
p = snowmicropyn.Profile.load('./S13M0013.pnt')
```

After this call you can access the profile's meta properties:

```
p.name
p.timestamp  # Timezone aware :)
p.coordinates  # WGS 84 latitude and longitude
p.spatial_resolution  # [mm]
p.overload
```

... and plenty more (not complete list).

To get the measurement values, you use the *samples()* property:

```
s = p.samples  # It's a pandas dataframe
print(s)
```

Export of data can be achieved using the methods *export_meta()* and *export_samples()*. Each method writes a file in CSV format:

```
p.export_meta()
p.export_samples()
```

amplifier_range
Returns the amplifier's range of the SnowMicroPen used to record this profile.

amplifier_serial
Returns the amplifier's serial number of the SnowMicroPen used to record this profile.

coordinates
Returns WGS 84 coordinates (latitude, longitude) of this profile in decimal format as a tuple (`(float, float)`) or `None` when coordinates are not available.

The coordinates are constructed by header fields of the pnt file. In case these header fields are empty or contain garbage, `None` is returned. You always can read the header fields yourself using the *pnt_header_value()* of this class for investigating what's present in the pnt header fields.

detect_ground()
Convenience method to detect the ground. This also sets the marker called "surface".

detect_surface()
Convenience method to detect the surface. This also sets the marker called "surface".

export_meta (*file=None*, *include_pnt_header=False*)
Export meta information of this profile into a CSV file.

When parameter `file` is not provided, the default name is used which is same as the pnt file from which the profile was loaded with a suffix *_meta* and the *csv* extension.

Parameters

- **file** – A **'Path-like object<https://docs.python.org/3/glossary.html#term-path-like-object>'_**.
- **include_pnt_header** – When `True`, raw pnt header fields are included too.

export_samples (*file=None*, *precision=4*, *snowpack_only=False*)
Export the samples of this profile into a CSV file.

When parameter `file` is not provided, the default name is used which is same as the pnt file from which the profile was loaded with a suffix *_samples* and the *csv* extension.

Parameters

- **file** – A path-like object.
- **precision** – Precision (number of digits after comma) of the values. Default value is 4.
- **snowpack_only** – In case set to true, only samples within the markers surface and ground are exported.

gps_numsats
Returns the number of satellites available when location was determined using GPS. Acts as an indicator of location's quality.

gps_pdop
Returns positional DOP (dilution of precision) value when location was determined using GPS. Acts as an indicator of location's quality.

ground
Convenience property to access value of 'ground' marker.

ini_file
`pathlib.Path` instance of the ini file in which markers are saved.

This file may does not exist.

static load (*pnt_file*, *name=None*)
Loads a profile from a pnt file.

This static method loads a pnt file and also its ini file in case its available. You can pass a name for the profile if you like. When omitted (passing `None`), the content of the pnt header field (*`Pnt.Header.FILENAME`*) is used.

Parameters

- **pnt_file** – A path-like object.
- **name** – Name of the profile.

marker (*label*, *fallback=<object object>*)
Returns the value of a marker as a `float`. In case a fallback value is provided and no marker is present, the fallback value is returned. It's recommended to pass a `float` fallback value. `None` is a valid fallback value.

Parameters

- **label** – Name of the marker requested.

- **fallback** – Fallback value returned in case no marker exists for the provided name.

markers
Returns all markers on the profile (a dictionary).

The dictionary keys are of type string, the values are floats. When no markers are set, the returned dictionary is empty.

max_force()
Get maximum force value of this profile.

name
Name of this profile. Can be specified when profile is loaded or, by default, "filename" header entry of the pnt file is used.

overload
Returns the overload value configured when this profile was recorded.

The unit of this value is N (Newton).

pnt_file
`pathlib.Path` instance of the pnt file this data was loaded from.

pnt_header_value(*pnt_header_id*)
Return the value of the pnt header by its ID.

For a list of available IDs, see *snowmicropyn.Pnt.Header*.

remove_marker(*label*)
Remove a marker.

Equivalent to `set_marker(label, None)`.

samples
Returns the samples. This is a pandas dataframe.

samples_within_distance(*begin=None*, *end=None*, *relativize=False*)
Get samples within a certain distance, specified by parameters `begin` and `end`

Default value for both is `None` and results to returns values from beginning or to the end of the profile.

Use parameter `relativize` in case you want to have the returned samples with distance values beginning from zero.

Parameters

- **begin** – Start of distance of interest. Default is `None`.
- **end** – End of distance of interest. Default is `None`.
- **relativize** – When set to `True`, the distance in the samples returned starts with 0.

samples_within_snowpack(*relativize=True*)
Returns samples within the snowpack, meaning between the values of marker "surface" and "ground".

save()
Save markers of this profile to a ini file.

> **Warning:** An already existing ini file is overwritten with no warning.

When no markers are set on the profile, the resulting file will be empty.

sensor_sensitivity
Returns the sensitivity of SnowMicroPen's force sensor. The unit of this value is µC/N.

sensor_serial
Returns the serial number of the force sensor of the SnowMicroPen used.

set_marker(*label*, *value*)
Sets a marker.

When passing `None``as value, the marker is removed. Otherwise, the provided value is converted into a ``float. The method raises `ValueError` in case this fails.

Parameters

- **label** – Name of the marker.
- **value** – Value for the marker. Passing a `float` is recommended.

smp_firmware
Returns the firmware version of the SnowMicroPen at the time of recording this profile.

smp_length
Returns the length on the SnowMicroPen used.

smp_serial
Returns the serial number of the SnowMicroPen used to record this profile.

smp_tipdiameter
Returns the tip diameter of SnowMicroPen used.

spatial_resolution
Returns the spatial resolution of this profile in mm (millimeters).

speed
Returns the speed used to record this profile in mm/s (millimeters per second).

surface
Convenience property to access value of 'surface' marker.

timestamp
Returns the timestamp when this profile was recorded. The timestamp is timezone aware.

2.6.3 Auto-detection of Ground & Surface

snowmicropyn contains algorithms to detect begin and end of the snowpack automatically. This algorithms may fail, so you may check the values before you process your data any further.

snowmicropyn.detection.**detect_ground**(*profile*)
Automatic detection of ground (end of snowpack).

Parameters **profile** (snowmicropyn.Profile) – The profile to detect ground in.

Returns Distance where ground was detected.

Return type float

snowmicropyn.detection.**detect_surface**(*profile*)
Automatic detection of surface (begin of snowpack).

Parameters **profile** – The profile to detect surface in.

Returns Distance where surface was detected.

Return type float

2.6.4 Shot Noise Model (Löwe, 2011)

2.6.5 SSA & Density (Proksch, 2015)

Calculation of density and ssa.

This module implements the methods to derive density and specific surface area (SSA) from SnowMicroPen's signal as described in publication Density, specific surface area, and correlation length of snow measured by highresolution penetrometry by Martin Proksch, Henning Löwe and Martin Schneebeli, publicised in Journal of Geophysical Research: Earth Surface, Volume 120, Issue 2, February 2015.

snowmicropyn.proksch2015.**calc**(*samples*, *window=2.5*, *overlap=50*)

Calculate ssa and density from a pandas dataframe containing the samples of a SnowMicroPen recording.

Parameters

- **samples** – A pandas dataframe containing the columns 'distance' and 'force'.
- **window** – Size of window in millimeters.
- **overlap** – Overlap factor in percent.

Returns A pandas dataframe with the columns 'distance', 'P2015_density' and 'P2015_ssa'.

snowmicropyn.proksch2015.**calc_from_loewe2012**(*shotnoise_dataframe*)

Calculate ssa and density from a pandas dataframe containing shot noise model values.

Parameters **shotnoise_dataframe** – A pandas dataframe containing shot noise model values.

Returns A pandas dataframe with the columns 'distance', 'P2015_density' and 'P2015_ssa'.

snowmicropyn.proksch2015.**calc_step**(*median_force*, *element_size*)

Calculation of density and ssa from median of force and element size.

This is the actual math described in the publication.

Parameters

- **median_force** – Median of force.
- **element_size** – Element size.

Returns Tuple containing density and ssa value.

2.6.6 Under the hood

snowmicropyn.Profile uses the method `load()` of class *snowmicropyn.Pnt* to get the raw data of pnt file. You probably won't ever use this class yourself.

class snowmicropyn.**Pnt**

Low level pnt loading functionality.

An example:

```
from snowmicropyn import Pnt

header, raw_samples = Pnt.load('S31M0067.pnt')

print(header[Pnt.Header.TIMESTAMP_YEAR].value)
print(raw_samples[2000:2005])
```

This may prints lines like `2017` and `(40, 41, 42, 43, 42)`.

class Header
Identifiers for pnt header entries

AMPLIFIER_RANGE = 'amplifier.range'
Amplifier range

AMPLIFIER_SERIAL = 'amplifier.serial'
Serial number of amplifier

AMPLIFIER_TYPE = 'amplifier.type'
Amplifier type value

BATTERY_VOLTAGE = 'battery.voltage'
Voltage of battery. **NOT IN USE**.

CAL_END = 'cal.end'
cal end... **NOT IN USE**.

CAL_START = 'cal.start'
cal start... **NOT IN USE**.

COMMENT_CONTENT = 'comment.content'
Comment content. **NOT IN USE**.

COMMENT_LENGTH = 'comment.length'
Comment length. **NOT IN USE**.

FILENAME = 'filename'
Filename of recording

GPS_CH1903_X = 'gps.ch1903.x'
CH1903 coordinate X. **NOT IN USE**.

GPS_CH1903_Y = 'gps.ch1903.y'
CH1903 coordinate Y. **NOT IN USE**.

GPS_CH1903_Z = 'gps.ch1903.z'
CH1903 coordinate Z. **NOT IN USE**.

GPS_FIXMODE = 'gps.fixmode'
GPS fix mode value

GPS_NUMSATS = 'gps.numsats'
Number of satellites when location was determined. **NOT IN USE**.

GPS_PDOP = 'gps.pdop'
Positional DOP, geometric dilution of precision

GPS_STATE = 'gps.state'
GPS state

GPS_WGS84_EAST = 'gps.wgs84.east'
Part of WGS 84 coordinates: E eastern, W for western.

GPS_WGS84_HEIGHT = 'gps.wgs84.height'
WGS84 altitude. **NOT IN USE**.

GPS_WGS84_LATITUDE = 'gps.wgs84.latitude'
WGS 84 latitude

GPS_WGS84_LONGITUDE = 'gps.wgs84.longitude'
WGS 84 longitude

GPS_WGS84_NORTH = 'gps.wgs84.north'
Part of WGS 84 coordinates: N for northern hemisphere, S for southern hemisphere

LOCAL_THETA = 'local.theta'
Local theta. **NOT IN USE**.

LOCAL_X = 'local.x'
Local X... **NOT IN USE**.

LOCAL_Y = 'local.y'
Local Y... **NOT IN USE**.

LOCAL_Z = 'local.z'
Local Z... **NOT IN USE**.

LOOPSIZE = 'loopsize'
Loop size... **NOT IN USE**.

RESERVED1 = 'reserved.1'
Reserved space 1

RESERVED2 = 'reserved.2'
Reserved space 2

RESERVED3 = 'reserved.3'
Reserved space 3

RESERVED4 = 'reserved.4'
Reserved space 4

SAMPLES_CONVFACTOR_FORCE = 'samples.conv.force'
Conversion factor of force

SAMPLES_CONVFACTOR_PRESSURE = 'samples.conv.pressure'
Conversion factor of pressure

SAMPLES_COUNT = 'samples.count'
Number of samples

SAMPLES_COUNT_FORCE = 'samples.force.count'
Number of force samples

SAMPLES_COUNT_TEMP = 'samples.temp.count'
Number of temperature samples. **NOT IN USE**.

SAMPLES_OFFSET_FORCE = 'samples.force.offset'
Offset value for force values. **NOT IN USE**.

SAMPLES_SPATIALRES = 'samples.spatialres'
Spatial resolution of distance

SAMPLES_SPEED = 'samples.speed'
Penetration speed

SENSOR_HANDOP = 'sensor.handop'
Hand operation. **NOT IN USE**.

SENSOR_OVERLOAD = 'sensor.overload'
Overload value

SENSOR_RANGE = 'sensor.range'
Sensor range

SENSOR_SENSITIVITIY = 'sensor.sensitivity'
Sensor sensitivity value

SENSOR_SERIAL = 'sensor.serial'
Serial number of sensor

SENSOR_TEMPOFFSET = 'sensor.tempoffset'
Sensor temperature offset value

SENSOR_TYPE = 'sensor.type'
Sensor type value

SMP_FIRMWARE = 'smp.firmware'
Version of firmware of SnowMicroPen used

SMP_LENGTH = 'smp.length'
SnowMicroPen's length

SMP_SERIAL = 'smp.serial'
SnowMicroPen's serial number

SMP_TIPDIAMETER = 'smp.diameter'
Diameter of SnowMicroPen's tip

TIMESTAMP_DAY = 'timestamp.day'
Timestamp's day

TIMESTAMP_HOUR = 'timestamp.hour'
Timestamp's hour

TIMESTAMP_MINUTE = 'timestamp.minute'
Timestamp's minute

TIMESTAMP_MONTH = 'timestamp.month'
Timestamp's month

TIMESTAMP_SECOND = 'timestamp.second'
Timestamp's second

TIMESTAMP_YEAR = 'timestamp.year'
Timestamp's year

WAYPOINTS = 'waypoints'
Way points... **NOT IN USE**.

static load(*file*)
Loads the raw data of a pnt file

This is the low level method used by class *snowmicropyn.Profile* to load the content of a pnt file. The method returns a tuple: A header (dict) and the raw measurement values (tuple). The header dictionary contains the header entries. Each entry has a label (`.label`), a unit (`.unit`) and a actual value (`.value`). Each entry can be `None`. Mostly this is the case for unit.

Parameters **file** – Path-like object

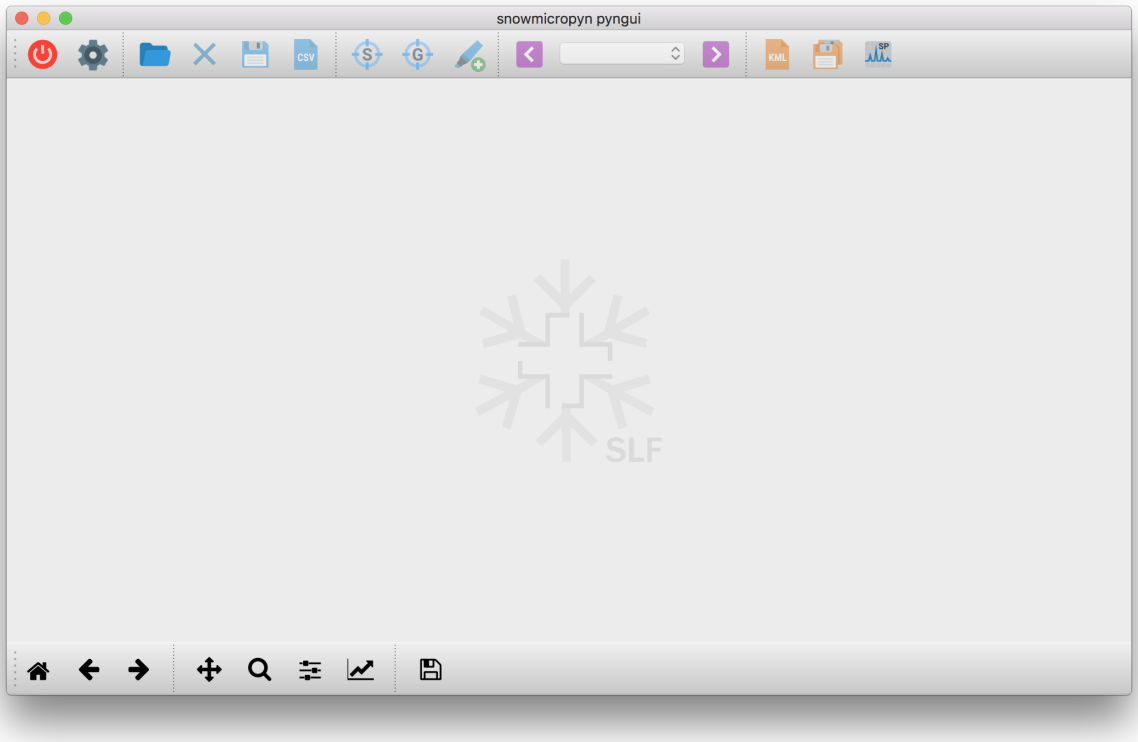
2.7 pyngui

2.7.1 What is pyngui?

pyngui is a desktop application to read, visualise and export files recorded by SnowMicroPen (pnt files).

2.7.2 Launch `pyngui`

When the *snowicropyn* package is installed, a simple script to start **`pyngui`** is registered too. Open a Terminal Window and type `pyngui` and hit `return`. A window should open which looks alike this screenshot:



Probably, this command fails to launch **`pyngui`**. Try to launch it manually then. Type:

```
python -m snowmicropyn.pyngui.app
```

or:

```
python -m snowmicropyn.pyngui.app
```

2.7.3 Features & Tips

Save your changes!

`pyngui` does not prompt or warn for unsaved changes. Don't forget to save your markers, otherwise they will be lost.

Surface & ground

`pyngui` uses the marker labels `surface` and `ground` to mark the begin and end of the snowpack. You can let **`pyngui`** auto detect those markers for you by clicking the according icons in the toolbar.

Drift, Offset & Noise

For each profile, the **pyngui** calculates drift, offset and noise and displays those values in the sidebar. This data is useful to check for a bad signal. The values are calculated for a section within the signal. Where this section starts and end is indicated in the sidebar. In case you want to specify the section yourself, set markers called `drift_begin` and `drift_end`. To simplest way to do so is context clicking into the plot.

2.8 Information for Developers of *snowmicropyn*

2.8.1 Necessary Accounts

To **develop** on *snowmicropyn*, you need a Github account. In case you're got write access to the repository, you can push you changes directly. Otherwise you have to send a pull request.

To **release** new versions of *snowmicropyn*, you need accounts on PyPI and test PyPI. The project maintainer must grant your account the necessary rights so your user is able to deploy releases.

To **release** updated **documentation**, you need an account on Read the Docs. The project maintainer must grant your account the necessary rights so your user is able to deploy releases.

2.8.2 Git Hash

To identify a version of *snowmicropyn* more accurately than just its version string, its git hash is added while publishing the package to PyPI. So in case a developer forgets to update a version string (in file:*__init__.py*) before releasing an update of *snowmicropyn*, it's still possible to identify what exactly was released.

The git hash is written to the file `snowmicropyn/githash` in the script **publish_to_pypi.sh**.

When using *snowmicropyn*, the hash of a release can be retrieved by method *snowmicropyn.githash()*.

2.8.3 Compiling Icons Into a Python File

The **pyngui** application requires icons used in menus and toolbar buttons. They are stored in the folder `resources`. For easy deployment, they are compiled into a python source file using the **pyrcc5** tool, which comes with the Qt package. Execute the following command when you did do changes in the resources folder:

```
pyrcc5 -o snowmicropyn/pyngui/icons.py resources/icons.qrc
```

This command generates an updated version of the file called `icons.py`. Don't ever edit this file manually.

WTF, the icons are gone!

In case you're suddenly see no more icons when running **pyngui**, it's likely due to your IDE has optimized your imports and dropped the statement

```
import snowmicropyn.pyngui.icons
```

as it seems to not have an effect. But it actually does. No icons without this import statement!

2.8.4 Releasing a New Version of *snowmicropyn*

1. Commit your changes

```
git commit -m "Some nice words about your changes"

Also make sure you updated the documentation if necessary!
```

1. Update version string (`__version__`) in file `snowmicropyn/__init__.py`

   Some examples for <version-number>, also consider reading **PEP 440**:

   - `v0.2.dev21` (Development Release)
   - `v0.2a4` (Alpha Release)
   - `v0.2b7` (Beta Release)
   - `v0.2.0` (Final Release)
   - `v0.2.11` (Bugfix Release)

2. MAKE SURE YOU UPDATED THE VERSION STRING!
3. Add an annotated tag in your repo

   ```
   git tag -a v<version-number> -m "Version v<version-number>"
   ```

   Note: It's common to add a 'v' character in front of the version number

   in a git version tag.

4. Push the Tag to GitHub

   ```
   git push origin
   ```

5. Use the script **`publish_to_pypi.sh`** to publish this release on PyPI. You have to provide the git tag which you want to release a a first parameter. In case you want to release to the hot PyPI (not test PyPI), you have to provide they string LIVE as a second parameter.

   The script will ask for your username and password on PyPI.

   ```
   publish_to_pypi.sh <version-number> LIVE
   ```

   Note: **`publish_to_pypi.sh`** is a unix shell script. You won't be able to run it on Windows unless you install Cygwin, Gow or a similar tool.

   If all goes fine, you should be able to install the release using the following commands:

   ```
   pip install --upgrade --no-cache-dir snowmicropyn
   ```

   In case you released to test PyPI:

   ```
   pip install --index-url https://test.pypi.org/simple/ --upgrade --no-cache-dir␣
   ↪snowmicropyn
   ```

6. Release new documentation on Read the Docs

CHAPTER 3

Contributors

- Sascha Grimm, SLF
- Henning Löwe, SLF
- Thiemo Theile, SLF
- Marcel Schoch, SLF

CHAPTER 4

License

This software and its documentation are released under GPL.

CHAPTER 5

Acknowledgements

Thanks to PyPI, GitHub and Read the Docs for hosting our project!

Also, many thanks to the people behind the products who made developing this package possible in reasonable time:

- The beloved language of Python.
- The beautiful Qt toolkit and the python binding PyQt.
- The awesome python packages matplotlib, numpy, scipy, pandas and pytz.

Python Module Index

S

Index