Seabird Documentation

Release 0.5.3

Guilherme Castelão

Feb 19, 2023

Contents

CHAPTER 1

User Documentation

Seabird at a glance

1.1 Overview

Seabird is a popular brand of sensors used for hydrographic measurements around the world, and that means a great deal of historical CTD data. These hydrographic profiles are usually available as ASCII files, containing the data itself, and plenty of fundamental metadata, such as position, date, calibration coefficients, and much more. Typically, these files are not hard for a human to interpret, but their format has changed over time, so it is a problem for automated processing.

While working with several years of CTD data from the project PIRATA, I realized that the first problem is just to be able to properly read all the data. I built this Python package with the goal to parse, in a robust way, the different historical Seabird output data file formats, and return that data in a uniform structure.

At this point, my goal is to have an object with attributes parsed from the header, and the data in (NumPy) Masked Arrays, so that the user doesn't need to manually determine the version and details of a .cnv file, but will still have it in a standard pattern, ready to use. Taking advantage of the basic library, this package includes some binary commands to output content as ASCII, but in a persistent format, or to convert it into a NetCDF file.

ATTENTION: this is not an official Sea-Bird package, so if you have trouble with it, please do not complain to Sea-Bird. Instead, open an issue at GitHub (https://github.com/castelao/seabird/issues), and I'll try to help you.

1.2 Installation

1.2.1 Requirements

- Python 2.7 or 3.X (recommended >=3.5)
- Numpy (>=1.1)

Optional requirement

- netCDF4, if you want to be able to export the data into netCDF files.
- CoTeDe, if you want to quality control your data with custom or pre-set group of checks.

1.2.2 Installing Seabird

Virtual Environments

You don't need to, but I strongly recommend to use virtualenv or conda.

Using pip

Currently, the most convenient way to install is with pip, by running in the terminal:

```
pip install seabird
```

If you don't have pip installed, you'll need to install pip first.

Alternative

To install with netCDF support:

```
pip install seabird[CDF]
```

To install with Quality Control support:

```
pip install seabird[QC]
```

1.3 Getting Started with Seabird

1.3.1 Inside python

```
>>> import seabird
```

In a python script, one can use like this:

```
>>> from seabird.cnv import fCNV
>>> profile = fCNV('your_file.cnv')
>>> profile.attributes  # It will return the header, as a dictionary.
>>> profile.keys() # It will list the available variables.
>>> profile['TEMP2'] # If TEMP2 was on the .keys(), this is how you get the data. It␣
↪will be a masked array.
```

The data from a profile is hence treated as it was a dictionary of Masked Arrays. To plot it, one could:

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(profile['depth'], profile['TEMP'], '.')
>>> plt.show()
```

1.3.2 From the terminal

One way to use is running on the shell the cnvdump. Independent of the historical version of the cnv file, it will return a default structure:

```
seabird cnvdump your_file.cnv
```

That can be used in a regular shell script. For example, let's consider a directory cruise1 with several sub directories, one for each leg of the cruise. One could list all the latitudes of each CTD cast like:

```
for file in `find ./cruise1 -iname '*.cnv'`
do seabird cnvdump $file | grep latitude
done
```

Now let's get that list ordered by the latitude:

```
for file in `find ./cruise1 -iname '*.cnv'`
do
    echo -n  `seabird cnvdump $file | grep latitude`
    echo -n " "
    echo $file
done | sort -n > mylist.txt
```

To convert a .cnv to a standard NetCDF, run:

```
seabird cnv2nc your_file.cnv
```

1.3.3 Quality Control

Until version 10.X the package CoTeDe would import PySeabird to apply QC, but since 11.X this relation inverted, and PySeabird now imports CoTeDe's resources on QC to evaluate CTD and TSG data.

To QC a cnv file, first load the QC function:

```
>>> from seabird.qc import fProfileQC
```

Now you're able to load the CTD data:

```
>>> pqc = fProfileQC('example.cnv')
```

The keys() will give you the data loaded from the CTD, similar to the output from the seabird.fCNV:

```
>>> pqc.keys()
```

To see one of the read variables listed on the previous step:

```
>>> pqc['temperature']
```

The flags are stored at pqc.flags and is a dictionary, being one item per variable evaluated. For example, to see the flags for the secondary salinity instrument, just do:

```
>>> pqc.flags['salinity2']
```

or for a specific test:

```
>>> pqc.flags['salinity2']['gradient']
```

To evaluate a full set of profiles at once, use the class ProfileQCCollection, like::

```
>>> dataset = ProfileQCCollection('/path/to/data/', inputpattern=".*\.cnv")
>>> dataset.flags['temperature'].keys()
```

The class cotede.qc.ProfileQCed is equivalent to the seabird.qc.ProfileQC, but it already mask the non approved data (flag != 1). Another it can also be used like::

```
>>> from seabird import cnv
>>> data = cnv.fCNV('example.cnv')

>>> import cotede.qc
>>> ped = cotede.qc.ProfileQCed(data)
```

1.3.4 More examples

I keep a notebooks collection of practical examples handling CTD data . If you have any suggestion, please let me know.

1.4 Command line (ctdqc)

A CTD data file can be quality controled from the shell script using the command line ctdqc. On this way it's easy to run the quality control from the shell, for example in a cron script for operational procedures.

In the shell one can run:

```
$ ctdqc MyData.cnv
```

A new file is created, MyData_qced.nc with depth, temperature and salinity, with the respective quality control flags. It's used the default cotede setup of tests.

With the command line it's easy to run in a full collection of cnv files, like:

```
for file in `find ./my_data_directory -iname '*.cnv'`;
do ctdqc $file;
done
```

This shell script will search for all .cnv files inside the directory ./my_data_directory (and sub-directories), evaluate each file and create on the side of the original data a netCDF with the QC flags.

In the future I'll turn this ctdqc command much more flexible.

1.5 Indices and tables

- genindex
- modindex
- search