Ipais Documentation

Release 0.0.2

Loïc Peron

Mar 12, 2019

documentation:

1 documentation **1**
- 1.1 ais module . . . 1

2 lpais **5**

3 install and test **7**
- 3.1 install from pypi . . . 7
- 3.2 dev install . . . 7
- 3.3 run the tests . . . 8
- 3.4 build the doc . . . 8

4 Documentation **9**

5 Meta **11**

6 Indices and tables **13**

Python Module Index **15**

CHAPTER 1

documentation

1.1 ais module

create functions to decode AIS message from NMEA lines.

Note: Thanks to **Kurt Schwehr** work on libais

Kurt Schwehr: schwehr@gmail.com, schwehr@google.com

This library uses libais and is largely inspired by **ais.stream** module but decode messages on a line by line basis.

Note the **stats** part has been removed as we believe this should be implemented by the user if needed.

example:

```
import lpais.ais as ais
decode = ais.decoder()

with open(data) as inputs:
    for line in inputs:
        line = line.strip()
        if line:
            data = decode(line)
            # data is None if line didn't result in a new message
            if data:
                # data is a dictionary containing AIS fields
                print(data)
```

use `keep_nmea` option to keep NMEA data into output dictionary, if enabled a '**nmea**' field will contain the NMEA line(s), which will be a concatenation of original lines if it's a multiline msg.

example:

```
import lpais.ais as ais
decode = ais.decoder(keep_nmea=True)

with open(data) as inputs:
    for line in inputs:
        line = line.strip()
        if line:
            data = decode(line)
            if data:
                print('NMEA': data['nmea'])
```

See also:

decoder for more details

exception lpais.ais.**AISError**(**kw)
base class for custom exceptions.

initialized with kw args that are used to format the string representation.

the base implementation add description field from **self.description**, the implementation class is then likely to define a description class attribute.

the base representation (str) will use the following fields from kw:

- 'description' : details above
- 'line' : the text line on which the exception occured

__init__(**kw)
Initialize self. See help(type(self)) for accurate signature.

exception lpais.ais.**DecodeError**(**kw)
Error while decoding AIS.

the representation will use the following fields from kw, in addition to the fields used by *AISError*:

- 'error_type' : error type
- 'error' : error message

exception lpais.ais.**DifferingTimestampsError**(**kw)
Timestamps not all the same.

the representation will use the following fields from kw, in addition to the fields used by *AISError*:

- 'timestamp' : value of the timestamp
- 'parts' : parts of the multiline message

exception lpais.ais.**InvalidChecksumError**(**kw)

exception lpais.ais.**InvalidChecksumInConstructedError**(**kw)

exception lpais.ais.**MissingTimestampsError**(**kw)
Timestamps missing.

the representation will use the following fields from kw, in addition to the fields used by *AISError*:

- 'parts' : parts of the multiline message

exception lpais.ais.**NoStationFoundError**(**kw)

exception lpais.ais.**OnlyMessageEndError**(**kw)
Do not have the preceeding packets for a multiline message.

the representation will use the following fields from `kw`, in addition to the fields used by *AISError*:

- 'bufferSlot' : information to identify the message

exception lpais.ais.**TooFewFieldsError**(***kw*)
Too few fields.

the representation will use the following fields from `kw`, in addition to the fields used by *AISError*:

- 'fields' : number of fields

lpais.ais.**decoder**(**args*, *keep_nmea=False*, *handle_err=<bound method Logger.error of <Logger lpais.ais (WARNING)>>*, ***kwargs*)
create a decoder function used to process NMEA lines one by one.

The created function will take a single text line as input arg and return either:

- `None` if the line didn't result as a new decoded AIS message

  this will be the case if it's a part of a multiline message or thie line could not be properly decoded or processed...

- a `dict` containing all the AIS message's fields.

The parameters used to create the decoder function are:

See also:

except for `keep_nmea`, all the parameters are forwarded to *normalizer* function.

- `validate_checksum`: (optional) wether or not to control cs.
- `allow_unknown` : (optional) allow no station.
- `window` : (optional) number of seconds to allow the later parts of a multiline message to span.
- `ignore_tagblock_station` : (optional) dont look for station in tagblock_station.
- `treat_ab_equal` : (optional) dont use A or B to identify msgs.
- `pass_invalid_checksums` : (optional) accept invalid cs.
- `allow_missing_timestamps` : (optional) accept missing ts.
- `handle_err` : (optional) called for every exception, default is `logger.error`.
- `keep_nmea` : (optional) keep origin NMEA line(s) and add it as 'nmea' field in output dictionary, all NMEA lines will be concatenated if multiline message.

lpais.ais.**normalizer**(*validate_checksum=True*, *allow_unknown=False*, *window=2*, *ignore_tagblock_station=False*, *treat_ab_equal=False*, *pass_invalid_checksums=False*, *allow_missing_timestamps=False*, *handle_err=<bound method Logger.error of <Logger lpais.ais (WARNING)>>*)
create a function which assembles single or multiline messages.

The created function will take a single text line as input arg and return either:

- `None` if the line didn't result in a new message

  this will be the case if it's a part of a multiline message or thie line could not be properly processed...

- a tuple (`tagblock`, `line`, `origin`) where `tagblock` is a `dict` containing tagblock info, `line` is the complete AIS message to decode, `origin` is the original NMEA line(s) composed of a concatenation of the lines if it's a multiline message.

See also:

decoder for args description

CHAPTER 2

lpais

use libais to decode ais messages, replacing **ais.stream**

CHAPTER 3

install and test

Warning: user is encouraged to use gcc to compile libais (*gcc* and *g++*), otherwise it's likely to fail. Make sure **gcc** will be used as default compliler or specify it when install / build

example:

```
$ CC=gcc-8 CXX=g++-8 pip install lpais
```

3.1 install from pypi

using pip:

```
$ pip install lpais
```

3.2 dev install

There is a makefile in the project root directory:

```
$ make dev
```

Using pip, the above is equivalent to:

```
$ pip install -r requirements-dev.txt
$ pip install -e .
```

3.3 run the tests

Use the makefile in the project root directory:

```
$ make test
```

This runs the tests generating a coverage html report

3.4 build the doc

The documentation is made with sphinx, you can use the makefile in the project root directory to build html doc:

```
$ make doc
```

CHAPTER 4

Documentation

Documentation on Read The Docs.

CHAPTER 5

Meta

loicpw - peronloic.us@gmail.com

Distributed under the MIT license. See `LICENSE.txt` for more information.

https://github.com/loicpw

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

lpais.ais, 1

Index

Symbols

__init__() (lpais.ais.AISError method), 2

A

AISError, 2

D

DecodeError, 2
decoder() (in module lpais.ais), 3
DifferingTimestampsError, 2

I

InvalidChecksumError, 2
InvalidChecksumInConstructedError, 2

L

lpais.ais (module), 1

M

MissingTimestampsError, 2

N

normalizer() (in module lpais.ais), 3
NoStationFoundError, 2

O

OnlyMessageEndError, 2

T

TooFewFieldsError, 3