pypromice

Release 1.3.6

GEUS Glaciology and Climate

May 16, 2024

GETTING STARTED

pypromice is designed for processing and handling PROMICE and GC-Net automated weather station (AWS) data. The PROMICE (Programme for Monitoring of the Greenland Ice Sheet) weather station network monitors glacier mass balance in the melt zone of the Greenland Ice Sheet, providing ground truth data to calibrate mass budget models. GC-Net (Greenland Climate Network) weather stations measure snowfall and surface properties in the accumulation zone, providing valuable knowledge on the Greenland Ice Sheet's mass gain and climatology.

The PROMICE and GC-Net monitoring networks have unique AWS configurations and provide specialized data, therefore a toolbox is needed to handle and process their data. pypromice is the go-to toolbox for handling and processing climate and glacier datasets from the PROMICE and GC-Net monitoring networks. New releases of pypromice are uploaded alongside PROMICE AWS data releases to the GEUS Dataverse for transparency purposes and to encourage collaboration on improving our data.

If you intend to use PROMICE and GC-Net AWS data and/or pypromice in your work, please cite these publications below, along with any other applicable PROMICE publications where possible:

Fausto, R.S., van As, D., Mankoff, K.D., Vandecrux, B., Citterio, M., Ahlstrøm, A.P., Andersen, S.B., Colgan, W., Karlsson, N.B., Kjeldsen, K.K., Korsgaard, N.J., Larsen, S.H., Nielsen, S., Pedersen, A.Ø., Shields, C.L., Solgaard, A.M., and Box, J.E. (2021) Programme for Monitoring of the Greenland Ice Sheet (PROMICE) automatic weather station data, Earth Syst. Sci. Data, 13, 3819–3845, https://doi.org/10.5194/essd-13-3819-2021

How, P., Wright, P.J., Mankoff, K., Vandecrux, B., Fausto, R.S. and Ahlstrøm, A.P. (2023) pypromice: A Python package for processing automated weather station data, Journal of Open Source Software, 8(86), 5298, https://doi.org/10.21105/joss.05298

How, P., Lund, M.C., Nielsen, R.B., Ahlstrøm, A.P., Fausto, R.S., Larsen, S.H., Mankoff, K.D., Vandecrux, B., Wright, P.J. (2023) pypromice, GEUS Dataverse, https://doi.org/10.22008/FK2/3TSBF0

CHAPTER

ONE

QUICK INSTALL

The latest release of pypromice can installed using conda or pip:

```
$ conda install pypromice -c conda-forge
```

```
$ pip install pypromice
```

The eccodes package for pypromice's post-processing functionality needs to be installed specifically in the pip distribution:

```
$ conda install eccodes -c conda-forge
$ pip install pypromice
```

And for the most up-to-date version of pypromice, the package can be cloned and installed directly from the repo:

```
$ pip install --upgrade git+http://github.com/GEUS-Glaciology-and-Climate/pypromice.git
```

CHAPTER

TWO

DEVELOPER INSTALL

pypromice can be ran in an environment with the pypromice package forked or cloned from the GitHub repo.

```
$ conda create --name pypromice python=3.8
$ conda activate pypromice
$ git clone git@github.com:GEUS-Glaciology-and-Climate/pypromice.git
$ cd pypromice/
$ pip install .
```

pypromice is also provided with a conda environment configuration file for a more straightforward set-up, if needed:

```
$ conda env create --file environment.yml -n pypromice
```

The package has inbuilt unit tests, which can be run to test the package installation:

```
$ python -m unittest discover pypromice
```

Note: This command line unit testing only works if pypromice is installed in the active Python environment. Unit testing can be run directly from the cloned pypromice top directory also either by running each script or from the command line as so: `$ python -m unittest discover src/pypromice`

CHAPTER

THREE

USER GUIDE

3.1 Level 0 to Level 3 processing

Two components are needed to perform Level 0 to Level 3 processing: - A Level 0 dataset file (.txt), or a collection of Level 0 dataset files - A station config file (.toml)

Two test station datasets and config files are available with pypromice as an example of the Level 0 to Level 3 processing. These can be found on the Github repo here, in the `src/pypromice/test/` directory in the cloned repo.

These can be processed from Level 0 to a Level 3 data product as an `AWS` object in pypromice.

```
from pypromice.process import AWS

# Define input paths
config = "src/pypromice/test/test_config1.toml"
inpath = "src/pypromice/test/"

# Initiate and process
a = AWS(config, inpath)
a.process()

# Get Level 3
l3 = a.L3
```

All processing steps are executed in `AWS.process`. These can also be broken down into each Level processing

```
from pypromice.process import AWS

# Define input paths
config = "src/pypromice/test/test_config2.toml"
inpath = "src/pypromice/test/"

# Initiate
a = AWS(config, inpath)

# Process to Level 1
a.getL1()
l1 = a.L1

# Process to Level 2
a.getL2()
l2 = a.L2
```

(continues on next page)

(continued from previous page)

```
# Process to Level 3
a.getL3()
l3 = a.L3
```

Level 3 data can be saved as both NetCDF and csv formatted files using the `AWS.write` function.

```
a.write("src/pypromice/test/")
```

The Level 0 to Level 3 processing can also be executed from a CLI using the `getL3` command.

```
$ get_l3 -c src/pypromice/test/test_config1.toml -i src/pypromice/test -o src/pypromice/
↪test
```

3.2 Loading our data

3.2.1 Import from Dataverse (no downloads!)

The automated weather station (AWS) datasets from the PROMICE and GC-Net monitoring programmes are openly available on the GEUS Dataverse. These can be imported directly with pypromice, with no downloading required.

```
import pypromice.get as pget

# Import AWS data from station KPC_U
df = pget.aws_data("kpc_u_hour.csv")
```

All available AWS datasets are retrieved by station name. Use `aws_names()` to list all station names which can be used as an input to `aws_data()`.

```
n = pget.aws_names()
print(n)
```

3.2.2 Download with pypromice

AWS data can be downloaded to file with pypromice. Open up a CLI and use the `getData` command.

```
$ get_promice_data -n KPC_U_hour.csv
```

Files are downloaded to the current directory as a CSV formatted file. Use the `-h` help flag to explore further input variables.

```
$ get_promice_data -h
```

Note: Currently, this functionality within pypromice is only for our hourly AWS data. For daily and monthly AWS data, please download these from the GEUS Dataverse.

3.2.3 Load from NetCDF file

AWS data can be loaded from a local NetCDF file with `xarray`.

```
import xarray as xr
ds = xr.open_dataset("KPC_U_hour.nc")
```

3.2.4 Load from CSV file

AWS data can be loaded from a local CSV file and handled as a `pandas.DataFrame`.

```
import pandas as pd
df = pd.read_csv("KPC_U_hour.csv", index_col=0, parse_dates=True)
```

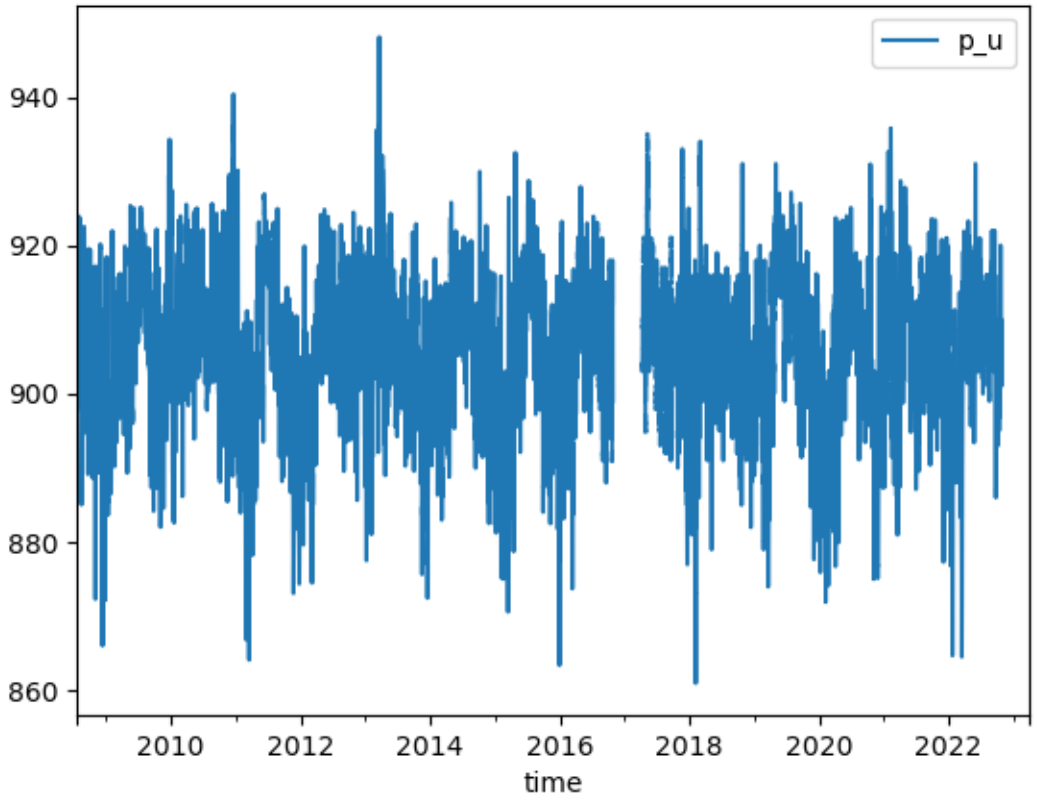
If you would rather handle the AWS data as an `xarray.Dataset` object then the `pandas.DataFrame` can be converted.

```
ds = xr.Dataset.from_dataframe(df)
```

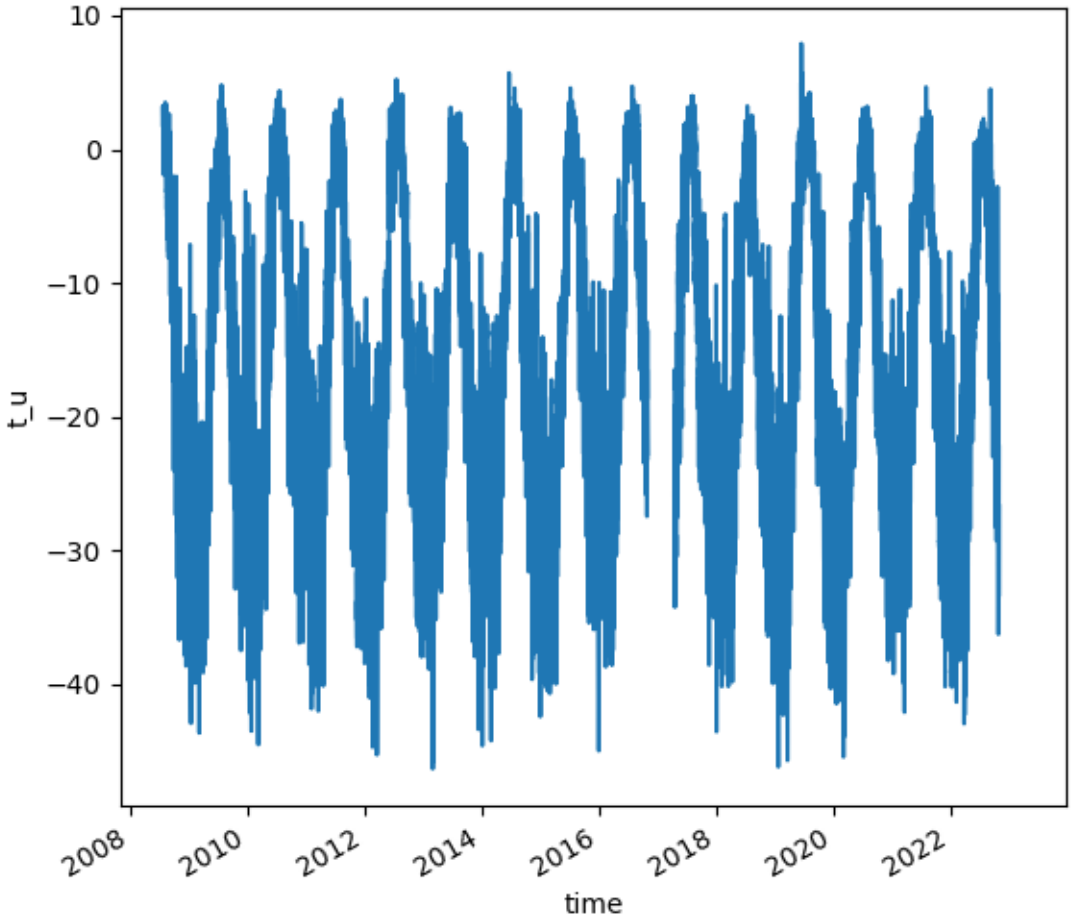
3.3 Plotting our data

Once loaded, variables from an AWS dataset can be simply plotted with using pandas or xarray.

```
# Plot variable with pandas
# In this case, we will plot air pressure
df.plot(kind='line', y='p_u', use_index=True)
```




```
# Plot variable with xarray
# In this case, we will plot air temperature
ds['t_u'].plot()
```



Note: Variable names are provided in the dataset metadata, or can be found on in our variables look-up table. For more complex plotting, please see either the xarray or pandas plotting documentation.

Warning: Plotting with either xarray or pandas requires matplotlib. This is not supplied as a dependency with pypromice, so please install matplotlib separately if you wish to do so.

CHAPTER

FOUR

DEVELOPER GUIDE

Contributions, bug reports and fixes, documentation improvements, enhancements and ideas are welcome. A good starting place to look at is:

1. *PROMICE-AWS-data-issues <https://github.com/GEUS-Glaciology-and-Climate/PROMICE-AWS-data-issues>_*, where we report suspicious or incorrect data
2. pypromice's GitHub Issues, for an overview of known bugs, developments and ideas

4.1 Data reports

Automatic weather station (AWS) data from the Greenland Ice Sheet are often imperfect due to the complexity and conditions involved in installing and maintaining the AWS.

If you are using our AWS data and something seems suspicious or erroneous, you can check the PROMICE-AWS-data-issues space to see if has previously been flagged and/or fixed. If not, then please follow the conventions stated in the repository and open an issue.

Note: Data visualisations best demonstrate data problems and are greatly appreciated in solving data issues. If you are unsure, see examples of our closed issues in PROMICE-AWS-data-issues

---

4.2 Bug reports and enhancement requests

Bug reports are essential to improving the stability and usability of pypromice. These should be raised on pypromice's GitHub Issues. A complete and reproducible report is essential for bugs to be resolved easily, therefore bug reports must:

1. Include a concise and self-contained Python snippet reproducing the problem. For example:

```
df = pget.aws_data(...)
```

2. Include a description of how your pypromice configuration is set up, such as from pip install and repository cloning/forking. If installed from pip or locally built, you can find the version of pypromice you are using with the following function.

```
from importlib import metadata
print(metadata.version('pypromice'))
```

3. Explain why the current behaviour is wrong or not desired, and what you expect instead

Note: Before submitting an issue, please make sure that your installation is correct and working from either the pip installation or the main branch of the pypromice repository.

4.3 Contributing to pypromice

You can work directly with pypromice's development if you have a contribution, such as a solution to an issue or a suggestion for an enhancment.

4.3.1 Forking

In order to contribute, you will need your own fork of the pypromice GitHub repository to work on the code. Go to the repo and choose the `Fork` option. This now creates a copy in your own GitHub space, which is connected to the upstream pypromice repository.

4.3.2 Creating a development branch

From your forked space, make sure you have a Python Environment for running pypromice, as described in *Developer install*. Then create and checkout a branch to make your developments on.

```
$ git checkout -b my-dev-branch
```

Keep changes in this branch specific to one bug or enhancement, so it is clear how this branch contributes to pypromice.

4.3.3 Creating a pull request

To contribute your changes to pypromice, you need to make a pull request from your forked development branch to pypromice's main branch. Before doing so, retrieve the most recent version of the main repository to keep this branch up to date with pypromice's main branch.

```
$ git fetch
$ git merge upstream/main
```

And then open a pull request as documented here. Make sure to include the following in your pull request description:

1. The aim of your changes
2. Details of what these changes are
3. Any limitations or further development needed

Your pull request will be reviewed and, if valid and suitable, will be accepted. Following this, you will be listed as a contributor to pypromice!

CHAPTER

FIVE

DATA TYPES

PROMICE automated weather station (AWS) data undergoes three main processing steps to reach a usable data standard. The output of each step forms an intermediary data version, which we label as sequential levels.

5.1 Level 0

Level 0 is raw, untouched data in one of three formats:

- [X] Copied from CF card in the field (`raw`)
- [X] Downloaded from logger box in the field as a Slim Table Memory (`stm`)
- [X] Transmitted via satellite and decoded (`tx`)

5.2 Level 1

- [X] Engineering units (e.g. current or volts) converted to physical units (e.g. temperature or wind speed)
- [X] Invalid/bad/suspicious data flagged
- [X] Multiple data files merged into one time series per station

5.3 Level 2

- [X] Calibration using secondary sources (e.g. radiometric correction requires input of tilt sensor)
- [X] Observation corrections applied
- [X] Station position relative to sun calculated

5.4 Level 3

- [X] Derived products calculated (e.g. sensible and latent heat flux)
- [X] Data merged, patched, and filled into one product

CHAPTER

SIX

TRANSMISSION TO LEVEL 0 PROCESSING

pypromice's `tx` module contains functionality for processing data transmissions from an AWS to a Level 0 data product:

- Fetching payload messages from the Iridium SBD service
- Payload message decoding from binary
- Level 0 data compiling

6.1 Payload handling

`SbdMessage` handles the SBD message, either taken from an `email.message.Message` object or a .sbd file.

`EmailMessage` handles the email message (that the SBD message is attached to) to parse information such as sender, subject, date, and to check for attachments.

6.2 Payload decoder

`PayloadFormat` handles the message types and decoding templates. These can be imported from file, with two default CSV files provided with pypromice - payload_formatter.csv and payload_type.csv.

6.3 Payload processing

`L0tx` handles the processing and output of the L0 transmission dataline. This object inherits from `EmailMessage` and `PayloadFormat` to read and decode messages

To reprocess old messages, these can be retrieved from the mailbox by rolling back the `uid` counter or by reading from .sbd file.

The following function can be executed from a CLI to fetch `L0` transmission messages from all valid stations:

```
$ get_l0tx -a accounts.ini -p credentials.ini -c tx/config
-u last_aws_uid.ini -o tx
```

Note: Credentials are needed to access our AWS transmissions. We do not provide these to external users, so purely include this workflow in pypromice to demonstrate transparency in our processing.

CHAPTER

SEVEN

LEVEL 0 TO LEVEL 3 PROCESSING

The `process` module is for processing AWS observations from Level 0 to Level 3 (end-user product) data products.

To process from L0>>L3, the following function can be executed in a CLI.

```
$ get_l3 -c config/KPC_L.toml -i . -o ../../aws-l3/tx"
```

And in parallel through all configuration .toml files `$imei_list`

```
$ parallel --bar "get_l3 -c ./{} -i . -o ../../aws-l3/tx" ::: $(ls $imei_list)
```

7.1 Station configuration

Each Level 0 file that will be processed must have an entry in the TOML-formatted configuration file. The config file can be located anywhere, and the processing script receives the config file and the location of the Level 0 data.

```
# Station configuration example, NSE

station_id = 'NSE'
logger_type = 'CR1000X'
number_of_booms = 2
nodata = ['-999', 'NAN']
modem = [['300534062416350', '2021-06-19 12:00:00']]  #Formatting [[modem,start,end],
↪[modem,start,end]]

['NSE_300534062416350_1.txt']
format     = 'TX'
skiprows = 0
latitude =  66.48
longitude = 42.49

dsr_eng_coef = 12.66
usr_eng_coef = 13.90
dlr_eng_coef = 8.55
ulr_eng_coef = 11.26
tilt_y_factor = -1

columns = ['time','rec','p_l','p_u','t_l','t_u','rh_l','rh_u',
          'wspd_l','wdir_l','wspd_u','wdir_u','dsr',
          'usr','dlr','ulr','t_rad','z_boom_l','z_boom_u',
```

(continues on next page)

(continued from previous page)

```
        't_i_1','t_i_2','t_i_3','t_i_4','t_i_5','t_i_6','t_i_7',
        't_i_8','t_i_9','t_i_10','t_i_11','tilt_x','tilt_y','rot',
        'precip_l','precip_u','gps_time','gps_lat','gps_lon',
        'gps_alt','gps_hdop','fan_dc_l','fan_dc_u','batt_v', 'p_i',
        't_i','rh_i','wspd_i','wdir_i','msg_i']
```

The TOML config file has the following expectations and behaviors:

- Properties can be defined at the top level or under a section
- Each file that will be processed gets its own section
- Properties at the top level are copied to each section (assumed to apply to all files)
- Top-level properties are overridden by file-level properties if they exist in both locations

---

Note: Be aware the column names should follow those defined in pypromice's variables look-up table. Any column names provided that are not in this look-up table will be passed through the processing untouched.

CHAPTER

EIGHT

PROCESS

8.1 process.aws

AWS data processing module

class process.aws.**AWS**(*config_file*, *inpath*, *var_file=None*, *meta_file=None*)

Bases: `object`

AWS object to load and process PROMICE AWS data

addAttributes(*L3*)

Add variable and attribute metadata

Parameters
L3 (*xr.Dataset*) – Level-3 data object

Returns
L3 – Level-3 data object with attributes

Return type
xr.Dataset

getL1()

Perform L0 to L1 data processing

getL2()

Perform L1 to L2 data processing

getL3()

Perform L2 to L3 data processing, including resampling and metadata and attribute population

loadConfig(*config_file*, *inpath*)

Load configuration from .toml file

Parameters

- **config_file** (*str*) – TOML file path
- **inpath** (*str*) – Input folder directory where L0 files can be found

Returns
conf – Configuration parameters

Return type
dict

loadL0()

Load level 0 (L0) data from associated TOML-formatted config file and L0 data file

Try readL0file() using the config with msg_lat & msg_lon appended. The specific ParserError except will occur when the number of columns in the tx file does not match the expected columns. In this case, remove msg_lat & msg_lon from the config and call readL0file() again. These station files either have no data after Nov 2022 (when msg_lat & msg_lon were added to processing), or for whatever reason these fields did not exist in the modem message and were not added.

Returns
ds_list – List of L0 xr.Dataset objects

Return type
list

process()

Perform L0 to L3 data processing

readL0file(*conf*)

Read L0 .txt file to Dataset object using config dictionary and populate with initial metadata

Parameters
conf (*dict*) – Configuration parameters

Returns
ds – L0 data

Return type
xr.Dataset

write(*outpath*)

Write L3 data to .csv and .nc file

writeArr(*outpath*)

Write L3 data to .nc and .csv hourly and daily files

Parameters

- **outpath** (*str*) – Output directory
- **L3** (*AWS.L3*) – Level-3 data object

class process.aws.**TestProcess**(*methodName='runTest'*)

Bases: TestCase

testAddAll()

Test variable and metadata attributes added to Dataset

testCLIgetl3()

Test get_l3 CLI

testCLIjoinl3()

Test join_l3 CLI

testL0toL3()

Test L0 to L3 processing

testgetMeta()

Test AWS names retrieval

testgetVars()

Test variable table lookup retrieval

process.aws.**addBasicMeta**(*ds*, *vars_df*)

Use a variable lookup table DataFrame to add the basic metadata to the xarray dataset. This is later amended to finalise L3

Parameters

- **ds** (*xr.Dataset*) – Dataset to add metadata to
- **vars_df** (*pd.DataFrame*) – Metadata dataframe

Returns
ds – Dataset with added metadata

Return type
xr.Dataset

process.aws.**addMeta**(*ds*, *meta*)

Add metadata attributes from file to dataset

Parameters

- **ds** (*xarray.Dataset*) – Dataset to add metadata attributes to
- **meta** (*dict*) – Metadata file

Returns
ds – Dataset with metadata

Return type
xarray.Dataset

process.aws.**addVars**(*ds*, *variables*)

Add variable attributes from file to dataset

Parameters

- **ds** (*xarray.Dataset*) – Dataset to add variable attributes to
- **variables** (*pandas.DataFrame*) – Variables lookup table file

Returns
ds – Dataset with metadata

Return type
xarray.Dataset

process.aws.**calculateSaturationVaporPressure**(*t*, *T_0=273.15*, *T_100=373.15*, *es_0=6.1071*, *es_100=1013.246*, *eps=0.622*)

Calculate specific humidity

Parameters

- **T_0** (*float*) – Steam point temperature. Default is 273.15.
- **T_100** (*float*) – Steam point temperature in Kelvin
- **t** (*xarray.DataArray*) – Air temperature
- **es_0** (*float*) – Saturation vapour pressure at the melting point (hPa)
- **es_100** (*float*) – Saturation vapour pressure at steam point temperature (hPa)

Returns

- *xarray.DataArray* – Saturation vapour pressure with regard to water above 0 C (hPa)
- *xarray.DataArray* – Saturation vapour pressure where subfreezing timestamps are with regards to ice (hPa)

process.aws.**getColNames**(*vars_df*, *booms=None*, *data_type=None*, *bedrock=False*)

Get all variable names for a given data type, based on a variables look-up table

Parameters

- **vars_df** (*pd.DataFrame*) – Variables look-up table
- **booms** (*int, optional*) – Number of booms. If this parameter is empty then all variables regardless of boom type will be passed. The default is None.
- **data_type** (*str, optional*) – Data type, "tx", "STM" or "raw". If this parameter is empty then all variables regardless of data type will be passed. The default is None.

Returns
Variable names

Return type
list

process.aws.**getConfig**(*config_file*, *inpath*, *default_columns: Sequence[str] = ('msg_lat', 'msg_lon')*)

Load configuration from .toml file. PROMICE .toml files support defining features at the top level which apply to all nested properties, but do not overwrite nested properties if they are defined

Parameters

- **config_file** (*str*) – TOML file path
- **inpath** (*str*) – Input folder directory where L0 files can be found

Returns
conf – Configuration dictionary

Return type
dict

process.aws.**getL0**(*infile*, *nodata*, *cols*, *skiprows*, *file_version*, *delimiter=','*, *comment='#'*, *time_offset: Optional[float] = None*) → Dataset

Read L0 data file into pandas DataFrame object

Parameters

- **infile** (*str*) – L0 file path
- **nodata** (*list*) – List containing value for nan values and reassigned value
- **cols** (*list*) – List of columns in file
- **skiprows** (*int*) – Skip rows value
- **file_version** (*int*) – Version of L0 file
- **delimiter** (*str*) – String delimiter for L0 file
- **comment** (*str*) – Notifier of commented sections in L0 file
- **time_offset** (*Optional[float]*) – Time offset in hours for correcting for non utc time data.

Returns
ds – L0 Dataset

Return type
xarray.Dataset

process.aws.**getMeta**(*m_file=None*, *delimiter=','*)

Load metadata table

Parameters

- **m_file** (*str*) – Metadata file path
- **delimiter** (*str*) – Metadata character delimiter. The default is ","

Returns
meta – Metadata dictionary

Return type
dict

process.aws.**getVars**(*v_file=None*)

Load variables.csv file

Parameters
v_file (*str*) – Variable lookup table file path

Returns
Variables dataframe

Return type
pandas.DataFrame

process.aws.**popCols**(*ds*, *names*)

Populate dataset with all given variable names

8.1.1 Parammeters

ds
[xr.Dataset] Dataset

names
[list] List of variable names to populate

process.aws.**populateMeta**(*ds*, *conf*, *skip*)

Populate L0 Dataset with metadata dictionary

Parameters

- **ds** (*xarray.Dataset*) – L0 dataset
- **conf** (*dict*) – Metadata dictionary
- **skip** (*list*) – List of column names to skip parsing to metadata

Returns
ds – L0 dataset with metadata populated as Dataset attributes

Return type
xarray.Dataset

process.aws.**resampleL3**(*ds_h*, *t*)

Resample L3 AWS data, e.g. hourly to daily average. This uses pandas DataFrame resampling at the moment as a work-around to the xarray Dataset resampling. As stated, xarray resampling is a lengthy process that takes ~2-3 minutes per operation: ds_d = ds_h.resample({'time':"1D"}).mean() This has now been fixed, so needs implementing: https://github.com/pydata/xarray/issues/4498#event-6610799698

Parameters

- **ds_h** (*xarray.Dataset*) – L3 AWS dataset either at 10 min (for raw data) or hourly (for tx data)
- **t** (*str*) – Resample factor, same variable definition as in pandas.DataFrame.resample()

Returns

ds_d – L3 AWS dataset resampled to the frequency defined by t

Return type

xarray.Dataset

process.aws.**roundValues**(*ds*, *df*, *col='max_decimals'*)

Round all variable values in data array based on pre-defined rounding value in variables look-up table DataFrame

Parameters

- **ds** (*xr.Dataset*) – Dataset to round values in
- **df** (*pd.Dataframe*) – Variable look-up table with rounding values
- **col** (*str*) – Column in variable look-up table that contains rounding values. The default is "max_decimals"

process.aws.**writeAll**(*outpath*, *station_id*, *l3_h*, *l3_d*, *l3_m*, *csv_order=None*)

Write L3 hourly, daily and monthly datasets to .nc and .csv files

outpath

[str] Output file path

station_id

[str] Station name

l3_h

[xr.Dataset] L3 hourly data

l3_d

[xr.Dataset] L3 daily data

l3_m

[xr.Dataset] L3 monthly data

csv_order

[list, optional] List order of variables

process.aws.**writeCSV**(*outfile*, *Lx*, *csv_order*)

Write data product to CSV file

Parameters

- **outfile** (*str*) – Output file path
- **Lx** (*xr.Dataset*) – Dataset to write to file
- **csv_order** (*list*) – List order of variables

process.aws.**writeNC**(*outfile*, *Lx*, *col_names=None*)

Write data product to NetCDF file

Parameters

- **outfile** (*str*) – Output file path
- **Lx** (*xr.Dataset*) – Dataset to write to file

8.2 process.L0toL1

AWS Level 0 (L0) to Level 1 (L1) data processing

process.L0toL1.**addTimeShift**(*ds*, *vars_df*)

Shift times based on file format and logger type (shifting only hourly averaged values, and not instantaneous variables). For raw (10 min), all values are sampled instantaneously so do not shift. For STM (1 hour), values are averaged and assigned to end-of-hour by the logger, so shift by -1 hr. For TX (time frequency depends on v2 or v3) then time is shifted depending on logger type. We use the 'instantaneous_hourly' boolean from variables.csv to determine if a variable is considered instantaneous at hourly samples.

This approach creates two separate sub-dataframes, one for hourly-averaged variables and another for instantaneous variables. The instantaneous dataframe should never be shifted. We apply shifting only to the hourly average dataframe, then concat the two dataframes back together.

It is possible to use pandas merge or join instead of concat, there are equivalent methods in each. In this case, we use concat throughout.

Fausto et al. 2021 specifies the convention of assigning hourly averages to start-of-hour, so we need to retain this unless clearly communicated to users.

Parameters

- **ds** (*xarray.Dataset*) – Dataset to apply time shift to
- **vars_df** (*pd.DataFrame*) – Metadata dataframe

Returns
ds_out – Dataset with shifted times

Return type
xarray.Dataset

process.L0toL1.**decodeGPS**(*ds*, *gps_names*)

Decode GPS information based on names of GPS attributes. This should be applied if gps information does not consist of float values

Parameters

- **ds** (*xr.Dataset*) – Data set
- **gps_names** (*list*) – Variable names for GPS information, such as "gps_lat", "gps_lon" and "gps_alt"

Returns
ds – Data set with decoded GPS information

Return type
xr.Dataset

process.L0toL1.**getPressDepth**(*z_pt*, *p*, *pt_antifreeze*, *pt_z_factor*, *pt_z_coef*, *pt_z_p_coef*)

Adjust pressure depth and calculate pressure transducer depth based on pressure transducer fluid density

Parameters

- **z_pt** (*xr.Dataarray*) – Pressure transducer height (corrected for offset)
- **p** (*xr.Dataarray*) – Air pressure
- **pt_antifreeze** (*float*) – Pressure transducer anti-freeze percentage for fluid density correction
- **pt_z_factor** (*float*) – Pressure transducer factor
- **pt_z_coef** (*float*) – Pressure transducer coefficient
- **pt_z_p_coef** (*float*) – Pressure transducer coefficient

Returns

- **z_pt_cor** (*xr.Dataarray*) – Pressure transducer height corrected
- **z_pt** (*xr.Dataarray*) – Pressure transducer depth

process.L0toL1.**getTiltDegrees**(*tilt*, *threshold*)

Filter tilt with given threshold, and convert from voltage to degrees. Voltage-to-degrees converseion is based on the equation in 3.2.9 at https://essd.copernicus.org/articles/13/3819/2021/#section3

Parameters

- **tilt** (*xarray.DataArray*) – Array (either 'tilt_x' or 'tilt_y'), tilt values (voltage)
- **threshold** (*int*) – Values below this threshold (-100) will not be retained.

Returns
dst.interpolate_na() – Array (either 'tilt_x' or 'tilt_y'), tilt values (degrees)

Return type
xarray.DataArray

process.L0toL1.**interpTemp**(*temp*, *var_configurations*, *max_interp=Timedelta('0 days 12:00:00')*)

Clip and interpolate temperature dataset for use in corrections

Parameters

- **temp** (*xarray.DataArray*) – Array of temperature data
- **vars_df** (*pandas.DataFrame*) – Dataframe to retrieve attribute hi-lo values from for temperature clipping
- **max_interp** (*pandas.Timedelta*) – Maximum time steps to interpolate across. The default is 12 hours.

Returns
temp_interp – Array of interpolatedtemperature data

Return type
xarray.DataArray

process.L0toL1.**reformatGPS**(*pos_arr*, *attrs*)

Correct latitude and longitude from native format to decimal degrees.

v2 stations should send "NH6429.01544","WH04932.86061" (NUK_L 2022) v3 stations should send coordinates as "6628.93936","04617.59187" (DY2) or 6430,4916 (NUK_Uv3) decodeGPS should have decoded these strings to floats in ddmm.mmmm format v1 stations however only saved decimal minutes (mm.mmmmm) as

float<=60. ‘ In this case, we use the integer part of the latitude given in the config file and append the gps value after it.

Parameters

- **pos_arr** (*xr.Dataarray*) – Array of latitude or longitude measured by the GPS
- **attrs** (*dict*) – The global attribute ‘latitude’ or ‘longitude’ associated with the file being processed. It is the standard latitude/longitude given in the config file for that station.

Returns
pos_arr – Formatted GPS position array in decimal degree

Return type
xr.Dataarray

process.L0toL1.**smoothTilt**(*tilt*, *win_size*)

Smooth tilt values using a rolling window. This is translated from the previous IDL/GDL smoothing algorithm: tiltX = smooth(tiltX,7,/EDGE_MIRROR,MISSING=-999) & tiltY = smooth(tiltY,7,/EDGE_MIRROR, MISSING=-999) endif In Python, this should be dstxy = dstxy.rolling(time=7, win_type='boxcar', center=True).mean() But the EDGE_MIRROR makes it a bit more complicated

Parameters

- **tilt** (*xarray.DataArray*) – Array (either ‘tilt_x’ or ‘tilt_y’), tilt values (can be in degrees or voltage)
- **win_size** (*int*) – Window size to use in pandas ‘rolling’ method. e.g. a value of 7 spans 70 minutes using 10 minute data.

Returns
tdf_rolling – The numpy array is the tilt values, smoothed with a rolling mean

Return type
tuple, as: (str, numpy.ndarray)

process.L0toL1.**toL1**(*L0*, *vars_df*, *T_0=273.15*, *tilt_threshold=-100*)

Process one Level 0 (L0) product to Level 1

Parameters

- **L0** (*xarray.Dataset*) – Level 0 dataset
- **vars_df** (*pd.DataFrame*) – Metadata dataframe
- **T_0** (*int*) – Air temperature for sonic ranger adjustment
- **tilt_threshold** (*int*) – Tilt-o-meter threshold for valid measurements

Returns
ds – Level 1 dataset

Return type
xarray.Dataset

8.3 process.L1toL2

AWS Level 1 (L1) to Level 2 (L2) data processing

process.L1toL2.**toL2**(*L1: Dataset*, *vars_df: DataFrame*, *T_0=273.15*, *ews=1013.246*, *ei0=6.1071*, *eps_overcast=1.0*, *eps_clear=9.36508e-06*, *emissivity=0.97*) → Dataset

Process one Level 1 (L1) product to Level 2

Parameters

- **L1** (*xarray.Dataset*) – Level 1 dataset
- **vars_df** (*pd.DataFrame*) – Metadata dataframe
- **T_0** (*float*) – Ice point temperature in K. The default is 273.15.
- **ews** (*float*) – Saturation pressure (normal atmosphere) at steam point temperature. The default is 1013.246.
- **ei0** (*float*) – Saturation pressure (normal atmosphere) at ice-point temperature. The default is 6.1071.
- **eps_overcast** (*int*) – Cloud overcast. The default is 1..
- **eps_clear** (*float*) – Cloud clear. The default is 9.36508e-6.
- **emissivity** (*float*) – Emissivity. The default is 0.97.

Returns
ds – Level 2 dataset

Return type
xarray.Dataset

8.4 process.L2toL3

AWS Level 2 (L2) to Level 3 (L3) data processing

process.L2toL3.**calcDirWindSpeeds**(*wspd*, *wdir*, *deg2rad=0.017453292519943295*)

Calculate directional wind speed from wind speed and direction

Parameters

- **wspd** (*xr.Dataarray*) – Wind speed data array
- **wdir** (*xr.Dataarray*) – Wind direction data array
- **deg2rad** (*float*) – Degree to radians coefficient. The default is np.pi/180

Returns

- **wspd_x** (*xr.Dataarray*) – Wind speed in X direction
- **wspd_y** (*xr.Datarray*) – Wind speed in Y direction

process.L2toL3.**calcHeatFlux**(*T_0*, *T_h*, *Tsurf_h*, *rho_atm*, *WS_h*, *z_WS*, *z_T*, *nu*, *q_h*, *p_h*, *kappa=0.4*, *WS_lim=1.0*, *z_0=0.001*, *g=9.82*, *es_0=6.1071*, *eps=0.622*, *gamma=16.0*, *L_sub=2830000.0*, *L_dif_max=0.01*, *c_pd=1005.0*, *aa=0.7*, *bb=0.75*, *cc=5.0*, *dd=0.35*)

Calculate latent and sensible heat flux using the bulk calculation method

Parameters

- **T_0** (*int*) – Steam point temperature
- **T_h** (*xarray.DataArray*) – Air temperature
- **Tsurf_h** (*xarray.DataArray*) – Surface temperature
- **rho_atm** (*float*) – Atmopsheric density
- **WS_h** (*xarray.DataArray*) – Wind speed
- **z_WS** (*float*) – Height of anemometer
- **z_T** (*float*) – Height of thermometer
- **nu** (*float*) – Kinematic viscosity of air
- **q_h** (*xarray.DataArray*) – Specific humidity
- **p_h** (*xarray.DataArray*) – Air pressure
- **kappa** (*int*) – Von Karman constant (0.35-0.42). Default is 0.4.
- **WS_lim** (*int*) – Default is 1.
- **z_0** (*int*) – Aerodynamic surface roughness length for momention, assumed constant for all ice/snow surfaces. Default is 0.001.
- **g** (*int*) – Gravitational acceleration (m/s2). Default is 9.82.
- **es_0** (*int*) – Saturation vapour pressure at the melting point (hPa). Default is 6.1071.
- **eps** (*int*) – Ratio of molar masses of vapor and dry air (0.622).
- **gamma** (*int*) – Flux profile correction (Paulson & Dyer). Default is 16..
- **L_sub** (*int*) – Latent heat of sublimation (J/kg). Default is 2.83e6.
- **L_dif_max** (*int*) – Default is 0.01.
- **c_pd** (*int*) – Specific heat of dry air (J/kg/K). Default is 1005..
- **aa** (*int*) – Flux profile correction constants (Holtslag & De Bruin '88). Default is 0.7.
- **bb** (*int*) – Flux profile correction constants (Holtslag & De Bruin '88). Default is 0.75.
- **cc** (*int*) – Flux profile correction constants (Holtslag & De Bruin '88). Default is 5.
- **dd** (*int*) – Flux profile correction constants (Holtslag & De Bruin '88). Default is 0.35.

Returns

- **SHF_h** (*xarray.DataArray*) – Sensible heat flux
- **LHF_h** (*xarray.DataArray*) – Latent heat flux

process.L2toL3.**calcHumid**(*T_0*, *T_100*, *T_h*, *es_0*, *es_100*, *eps*, *p_h*, *RH_cor_h*)

Calculate specific humidity

Parameters

- **T_0** (*float*) – Steam point temperature. Default is 273.15.
- **T_100** (*float*) – Steam point temperature in Kelvin
- **T_h** (*xarray.DataArray*) – Air temperature
- **eps** (*int*) – ratio of molar masses of vapor and dry air (0.622)
- **es_0** (*float*) – Saturation vapour pressure at the melting point (hPa)
- **es_100** (*float*) – Saturation vapour pressure at steam point temperature (hPa)

- **p_h** (*xarray.DataArray*) – Air pressure
- **RH_cor_h** (*xarray.DataArray*) – Relative humidity corrected

Returns
Specific humidity data array

Return type
xarray.DataArray

process.L2toL3.**calcVisc**(*T_h*, *T_0*, *rho_atm*)

Calculate kinematic viscosity of air

Parameters

- **T_h** (*xarray.DataArray*) – Air temperature
- **T_0** (*float*) – Steam point temperature
- **rho_atm** (*xarray.DataArray*) – Surface temperature

Returns
Kinematic viscosity

Return type
xarray.DataArray

process.L2toL3.**cleanHeatFlux**(*SHF*, *LHF*, *T*, *Tsurf*, *p*, *WS*, *RH_cor*, *z_boom*)

Find invalid heat flux data values and replace with NaNs, based on air temperature, surface temperature, air pressure, wind speed, corrected relative humidity, and boom height

Parameters

- **SHF** (*xarray.DataArray*) – Sensible heat flux
- **LHF** (*xarray.DataArray*) – Latent heat flux
- **T** (*xarray.DataArray*) – Air temperature
- **Tsurf** (*xarray.DataArray*) – Surface temperature
- **p** (*xarray.DataArray*) – Air pressure
- **WS** (*xarray.DataArray*) – Wind speed
- **RH_cor** (*xarray.DataArray*) – Relative humidity corrected
- **z_boom** (*xarray.DataArray*) – Boom height

Returns

- **SHF** (*xarray.DataArray*) – Sensible heat flux corrected
- **LHF** (*xarray.DataArray*) – Latent heat flux corrected

process.L2toL3.**cleanSpHumid**(*q_h*, *T*, *Tsurf*, *p*, *RH_cor*)

Find invalid specific humidity data values and replace with NaNs, based on air temperature, surface temperature, air pressure, and corrected relative humidity

Parameters

- **q_h** (*xarray.DataArray*) – Specific humidity
- **T** (*xarray.DataArray*) – Air temperature
- **Tsurf** (*xarray.DataArray*) – Surface temperature
- **p** (*xarray.DataArray*) – Air pressure

- **RH_cor** (*xarray.DataArray*) – Relative humidity corrected

Returns
q_h – Specific humidity corrected

Return type
xarray.DataArray

process.L2toL3.**toL3**(*L2*, *T_0=273.15*, *z_0=0.001*, *R_d=287.05*, *eps=0.622*, *es_0=6.1071*, *es_100=1013.246*)

Process one Level 2 (L2) product to Level 3 (L3)

Parameters

- **L2** (*xarray:Dataset*) – L2 AWS data
- **T_0** (*int*) – Steam point temperature. Default is 273.15.
- **z_0** (*int*) – Aerodynamic surface roughness length for momention, assumed constant for all ice/snow surfaces. Default is 0.001.
- **R_d** (*int*) – Gas constant of dry air. Default is 287.05.
- **eps** (*int*) – Default is 0.622.
- **es_0** (*int*) – Saturation vapour pressure at the melting point (hPa). Default is 6.1071.
- **es_100** (*int*) – Saturation vapour pressure at steam point temperature (hPa). Default is 1013.246.

CHAPTER

NINE

POSTPROCESS

9.1 postprocess.csv2bufr

CHAPTER

TEN

QC

10.1 qc.persistence

qc.persistence.**count_consecutive_persistent_values**(*data: Series*, *max_diff: float*) → Series

qc.persistence.**find_persistent_regions**(*data: Series*, *min_repeats: int*, *max_diff: float*) → Series

Algorithm that ensures values can stay the same within the outliers_mask

qc.persistence.**persistence_qc**(*ds: Dataset*, *variable_thresholds: Optional[Mapping] = None*) → Dataset

Detect and filter data points that seems to be persistent within a certain period.

TODO: It could be nice to have a reference to the logger or description of the behaviour here. The AWS logger program is know to return the last successfully read value if it fails reading from the sensor.

Parameters

- **ds** (*xr.Dataset*) – Level 1 datset
- **variable_thresholds** (*Mapping*) – Define threshold dict to hold limit values, and the difference values. Limit values indicate how much a variable has to change to the previous value period is how many hours a value can stay the same without being set to NaN * are used to calculate and define all limits, which are then applied to *_u, *_l and *_i

Returns

ds_out – Level 1 dataset with difference outliers set to NaN

Return type

xr.Dataset

CHAPTER

ELEVEN

GET

11.1 get.get

AWS data retrieval module

class get.get.**TestGet**(*methodName='runTest'*)

Bases: TestCase

testGetCLI()

Test get_promice_data

testURL()

Test URL retrieval

get.get.**aws_data**(*aws_name*)

Return PROMICE and GC-Net AWS L3 v3 hourly observations

Returns
df – AWS observations dataframe

Return type
pandas.DataFrame

get.get.**aws_names**()

Return PROMICE and GC-Net AWS names that can be used in get.aws_data() fetching

get.get.**lookup_table**(*base_dois*, *server='https://dataverse.geus.dk'*)

Fetch dictionary of data files and download URLs from a DOI entry in the GEUS Dataverse

Parameters

- **base_dois** (*list*) – List of DOIs to search
- **server** (*str, optional*) – DOI server. The default is "https://dataverse.geus.dk"

get.get.**watson_discharge**(*t='hour'*)

Return PROMICE hourly Watson river discharge

Parameters
t (*str*) – Temporal resolution of the data - "hour", "day" or "year"

Returns
df – Watson river discharge dataframe

Return type
pandas.DataFrame

CHAPTER

TWELVE

TX

12.1 tx.tx

AWS Level 0 (L0) data transmission fetching module

class tx.tx.**EmailMessage**(*email_msg*, *sender_name*)

Bases: *SbdMessage*

Email message object

checkEmail(*email_msg*)

Check if email is Message object

checkSender(*sender_name*)

Check email message from field matches sender name or names

getEmailBody()

Get email message body

getEmailInfo()

Parse message in email object

getIMEI()

Get modem identifier from email subject string

tx.tx.**GFP2toDEC**(*Bytes*)

Two-bit decoder

Parameters
Bytes (*list*) – List of two values

Returns
Decoded value

Return type
float

tx.tx.**GLI4toDEC**(*Bytes*)

Four-bit decoder

Parameters
Bytes (*list*) – List of four values

Returns
Decoded value

Return type
float

class tx.tx.**L0tx**(*email_msg*, *format_file=None*, *type_file=None*, *sender_name=['sbdservice', 'ice@geus.dk', 'emailrelay@konectgds.com']*, *UnixEpochOffset=0*, *CRbasicEpochOffset=631152000*)

Bases: EmailMessage, PayloadFormat

L0 tranmission data object

check2BitNAN(*msg*, *type_letter*, *letter_flag=['g', 'n', 'e']*, *nan_value=8191*)

Check if byte is a 2-bit NAN. This occurs when the GPS data is not available and the logger sends a 2-bytes NAN instead of a 4-bytes value

checkByte(*b*)

Check byte format against payload formatter object

checkLength()

checkPayload()

Check message payload

getByteValue(*ValueBytesCount*, *BinaryMessage*, *idx*)

Get values from byte range in binary message

getDataLine()

Get data line from transmission message

Returns
Dataline string if found

Return type
str or None

getFirstByte()

Get first byte in payload

getFormat()

Get binary format type from first byte in payload

Returns

- **bval** (*int or None*) – Format value
- **bfor** (*str or None*) – Format string characters
- **bname** (*str or None*) – Format name
- **blength** (*int or None*) – Expected format length
- **bidx** (*int*) – Format index
- *bool* – Valid format flag

isDiagnostics(*DataLine*)

Flag if message is diagnostics

isObservations(*DataLine*)

Flag if message is observations

isSummer(*DataLine*)

Flag if message is summer message

isWatsonObservation(*DataLine*)

Flag if message is Watson River measurement

isWithInstance(*DataLine*)

Flag if message is with instance

updateByteCounter(*value*)

Update byte counter for decoding message

writeEntry(*entry*, *i*)

Write out comma-formatted data entry from message

class tx.tx.**PayloadFormat**(*format_file=None*, *type_file=None*)

Bases: object

Payload formatter object

readFile(*in_file*)

Read lines from file

Parameters
in_file (*str*) – Input file path

Returns
lines – List of file line contents

Return type
list

readFormatter(*in_file*, *delimiter=','*)

Read payload formatter from file. Outputted dictionary set as key[number]: [expected_length, format_characters, description]. Flag column (info[4]) used to signify if entry should be written to output

Parameters

- **in_file** (*str*) – Input file path
- **delimiter** (*str, optional*) – File delimiter. The default is ","

Returns
payload_fmt – Payload format information

Return type
dict

readPkgFile(*fname*)

Read lines from internal package file

Parameters
fname (*str*) – Package file name

Returns
lines – List of file line contents

Return type
list

readType(*in_file*, *delimiter=','*)

Read payload type setter from file. Outputted dictionary set as key[type_letter]: number_of_bytes

Parameters

- **in_file** (*str*) – Input file path

- **delimiter** (*str, optional*) – File delimiter. The default is ","

Returns
payload_typ – Payload type information

Return type
dict

tx.tx.**RAWtoSTR**(*Bytes*)

Byte-to-string decoder

Parameters
Bytes (*list*) – List of values

Return type
Decoded string characters

class tx.tx.**SbdMessage**(*content*, *attach*, *imei*)

Bases: object

SBD transmission message object

checkAttachment(*attach*)

Check if attachment is present in email.message.Message object

checkAttachmentName(*attach_file*)

Check if attachment is .sbd file

getKeyValue(*content*, *seps*, *key*, *integer=True*)

Get attribute from email via keyword

getLocation(*content*, *seps=' '*, *key='Unit Location'*)

Get latitude longitude unit location from email message

getPayloadFromEmail(*attach*, *message_size*)

Get Sbd payload from email object

getPayloadFromFile(*attach*)

Read Sbd payload from .sbd file

getStatus(*content*, *seps1=': '*, *seps2=' '*, *key='Session Status'*)

Get session status from email message

class tx.tx.**TestTX**(*methodName='runTest'*)

Bases: TestCase

testCLIl0tx()

Test get_l0tx CLI

testCLImsg()

Test get_msg CLI

testCLIwatson()

Test get_watsontx CLI

testEmailMessage()

Test EmailMessage object initialisation from .msg file

testL0tx()

Test L0tx object initialisation

testPayloadFormat()

Test PayloadFormat object initialisation

tx.tx.**addTail**(*in_file*, *out_dir*, *aws_name*, *header_names=''*, *lines_limit=100*)

Generate tails file from L0tx file

Parameters

- **in_file** (*str*) – Input L0tx file
- **out_dir** (*str*) – Output directory for tails file
- **aws_name** (*str*) – AWS name
- **header_names** (*str, optional*) – Header names. The default is ‘’.
- **lines_limit** (*int, optional*) – Number of lines to append to tails file. The default is 100.

tx.tx.**findDuplicates**(*lines*)

Find duplicates lines in list of strings

Parameters
lines (*list*) – List of strings

Returns
unique_lines – List of unique strings

Return type
list

tx.tx.**findLine**(*content*, *key*)

Find keyword in line

Parameters

- **content** (*str*) – String to find keyword in
- **key** (*str*) – Keyword to find in string

Returns
line – Line that keyword appears on

Return type
str

tx.tx.**getMail**(*mail_server*, *last_uid=1*)

Retrieve new mail

Parameters

- **mail_server** (*imaplib.IMAP_SSL*) – Mail server object
- **last_uid** (*int, optional*) – Mail uid to start retrieval from. The default is 1.

Yields

- *str* – Mail uid
- *str* – Mail message

tx.tx.**loadMsg**(*fname*)

Load .msg email file into format compatible with EmailMessage and SbdMessage objects

Parameters
fname (*str*) – File path to .msg file

Returns
Email message object

Return type
email.message.Message

tx.tx.**parseValue**(*line*, *seps*)

Parse last value from line according to separating characters

Parameters

- **line** (*str*) – String to split
- **sep** (*str*) – Separator characters to split line by

Returns
value – Value extracted from line

Return type
str

tx.tx.**readSBD**(*sbd_file*)

Read encoded .sbd transmission file

Parameters
sbd_file (*str*) – Filepath to encoded .sbd file

Returns
data – Transmission message byte object

Return type
bytes

tx.tx.**saveMsg**(*msg*, *fname*)

Save email message object to .msg file

Parameters

- **msg** (*email.message.Message*) – Email object to save to file
- **fname** (*str*) – File path to outputted .msg file

tx.tx.**sortLines**(*in_file*, *out_file*, *replace_unsorted=True*)

Sort lines in text file

Parameters

- **in_file** (*str*) – Input file path
- **out_file** (*str*) – Output file path
- **replace_unsorted** (*bool, optional*) – Flag to replace unsorted files with sorted files. The default is True.

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

L

M

P

Q

R

S

T

U

W