loggertodb Documentation

Release 3.0.0

Antonis Christofides

Dec 05, 2022

Contents:

CHAPTER 1

Installation

1.1 Windows

`loggertodb` is just a single executable, `loggertodb.exe`. You download it and run it; there's no installer.

Download `loggertodb.exe` from https://github.com/openmeteo/loggertodb/releases/.

1.2 Linux

Simply execute this:

```
pip3 install loggertodb
```

CHAPTER 2

Usage

`loggertodb` reads a data file (or several data files), connects to Enhydris, determines which records in the file are newer than those stored in Enhydris, and appends them. The details of its operation are specified in the configuration file specified on the command line.

2.1 Quick start

2.1.1 Installation

See *Installation*.

2.1.2 How to run it

First, you need to create a configuration file with a text editor such as `vim`, `emacs`, `notepad`, or whatever. Create such a file and name it, for example, `/var/tmp/loggertodb.conf`, or, on Windows, something like `C:\Users\user\loggertodb.conf`, with the following contents (the contents don't matter at this stage, just copy and paste them from below):

```
[General]
base_url = https://openmeteo.org/
auth_token = 123456789abcdef0123456789abcdef012345678
loglevel = INFO
```

Then, open a command prompt and give it this command:

Unix/Linux:

```
loggertodb /var/tmp/loggertodb.conf
```

Windows:

```
C:\Program Files\Loggertodb\loggertodb.exe C:\Users\user\loggertodb.conf
```

(the details may differ; for example, in 64-bit Windows, it may be `C:Program Files (x86)` instead of `C:Program Files`.)

If you have done everything correctly, it should show an error message similar to "No stations have been specified". This means that, apart from the "General" section you have to add more sections to the configuration file.

2.1.3 Authentication

`loggertodb` needs to logon to Enhydris, and for this it needs an authentication token to be specified as the value of the `auth_token` parameter. You can get a token at the `/api/auth/login/` URL of Enhydris, such as https://openmeteo.org/api/auth/login/.

2.1.4 Configuration file examples

The following instructs `loggertodb` to use the single data file `zeno.data` and upload its data to `openmeteo.org`; the first field of each line (after the date and time) will be uploaded to time series group 232, the second to 233, and so on. The last field of each line will not be uploaded (symbolized with the 0). The time zone of the timestamps is 2 hours east of UTC:

```
[General]
loglevel = WARNING
logfile = /var/log/loggertodb/itiameteo.log
base_url = https://openmeteo.org/
auth_token = 123456789abcdef0123456789abcdef012345678

[NTUA]
station_id = 1334
path = /var/local/openmeteo/logger_data_files/ntua/zeno.data
storage_format = simple
date_format = %y/%m/%d %H:%M:%S
timezone = Etc/GMT-2
fields = 232,233,247,248,237,238,236,9141,5461,6659,9139,6661,240,6539,6541,230,234,0
```

The following instructs `loggertodb` to use two data files (one for meteorological station PRASINOS, one for VILIA; these are just labels to make it easy for you to read the file; that are not used anywhere). While reading each line's fields, the value "NAN" instead of a number will be interpreted as an empty (or missing, or null) value. The time zone of the timestamps changes because of daylight saving time (see *Daylight saving time*):

```
[General]
loglevel = WARNING
logfile = /var/log/loggertodb/defkalion.log
base_url = https://openmeteo.org/
auth_token = 123456789abcdef0123456789abcdef012345678

[PRASINOS]
station_id = 1345
path = /var/local/openmeteo/logger_data_files/defkalion/prasino.data
storage_format = simple
date_format = %d/%m/%Y %H:%M:%S
fields = 9180,9182,9184,9178
null = NAN
timezone = Europe/Athens
```

(continues on next page)

(continued from previous page)

```
[VILIA]
station_id = 1347
path = /var/local/openmeteo/logger_data_files/defkalion/vilia.data
storage_format = simple
date_format = %d/%m/%Y %H:%M:%S
fields = 9172,9174,9176,9170
null = NAN
timezone = Europe/Athens
```

The next is very similar to the previous one, but it's for Windows, it uses a star for null values, and the fields in the files are delimited with commas instead of spaces. In addition, the sixth field of each line (after the date and time) is not uploaded:

```
[General]
loglevel = INFO
logfile = C:\a2a\loggertodb-kostilata.log
base_url = https://openmeteo.org/
auth_token = 123456789abcdef0123456789abcdef012345678

[ANO_KOSTILATA]
station_id = 1387
path = C:\a2a\ano_kostilata_20130601.txt
storage_format = simple
delimiter = ,
date_format = %d-%m-%Y %H:%M:%S
fields = 9290,9285,9292,9294,9295,0,9291,9289,9288,9286
null = *
timezone = Europe/Athens

[KATO_KOSTILATA]
station_id = 1388
path = C:\a2a\ano_kostilata_20130601.txt
storage_format = simple
delimiter = ,
date_format = %d-%m-%Y %H:%M:%S
fields = 9279,9274,9281,9283,9284,0,9280,9278,9277,9275
null = *
timezone = Europe/Athens
```

Finally, an example of a configuration that uses the files produced by Davis WeatherLink. In this case, `C:\WeatherLink\komboti` is the directory that contains the .WLK files (it is necessary to read more below about *WDAT5 units* and *the WDAT5 format*):

```
[General]
loglevel = INFO
logfile = C:\WeatherLink\komboti\loggertodb.log
base_url = https://openmeteo.org/
auth_token = 123456789abcdef0123456789abcdef012345678

[KOMBOTI]
station_id = 1389
path = C:\WeatherLink\komboti
storage_format = wdat5
outsideTemp = 1256
hiOutsideTemp = 1257
```

(continues on next page)

(continued from previous page)

```
rain = 1652
timezone = Europe/Athens
temperature_unit = F
rain_unit = inch
```

2.1.5 Running automatically

You probably want to have `loggertodb` automatically update the data. To do this, either run it periodically (from `cron` on Unix and `Task Scheduler` on Windows), or, if the software you use to download the data from the meteorological station has the feature, add `loggertodb` as a trigger.

2.2 Configuration file reference

The configuration file has the format of INI files. There is a `[General]` section with general parameters, and any number of other sections, which we will call "file sections", each file section referring to one file to be processed; this makes it possible to process many files in a single `loggertodb` execution using a single configuration file and fewer HTTP requests.

2.2.1 General parameters

loglevel Can have the values `ERROR`, `WARNING`, `INFO`, `DEBUG`, indicating the amount of output requested from `loggertodb`. The default is `WARNING`.

logfile The full pathname of a log file. If unspecified, log messages will go to the standard error.

base_url The base url of the Enhydris installation to connect to, such as `https://openmeteo.org/`.

auth_token The token `loggertodb` will use to authenticate with Enhydris. Obviously the user to whom the token corresponds must have write permissions for all time series that will be uploaded.

2.2.2 File parameters

station_id The id of the station.

path The full pathname of the data storage.

storage_format The format of the data storage. See *Supported formats*.

fields (Not for the wdat5 format.) A series of comma-separated integers representing the ids of the time series groups to which the data file fields correspond (time series groups are what Enhydris lists as "Data" in the page for a station). A zero indicates that the field is to be ignored. The first number corresponds to the first field after the date (and possibly other fixed fields depending on data file format, such as the subset identifier) and should be the id of the corresponding time series group, or zero if the field is dummy; the second number corresponds to the second field after the fixed fields, and so on.

Each time series group contains variations of the same time series, such as initial, checked and aggregated. `loggertodb` uploads the data to the "initial" time series of the group. If such a time series does not exist, it is created.

nfields_to_ignore This is used only in the `simple` format; it's an integer that represents a number of fields before the date and time that should be ignored. The default is zero. If, for example, the date and time are preceded by a record id, set `nfields_to_ignore=1` to ignore the record id.

subset_identifiers Some file formats mix two or more sets of measurements in the same file; for example, there may be ten-minute and hourly measurements in the same file, and for every 6 lines with ten-minute measurements there may be an additional line with hourly measurements (not necessarily the same variables). `loggertodb` processes only one set of lines each time. Such files have one or more additional distinguishing fields in each line, which helps to distinguish which set it is. `subset_identifiers`, if present, is a comma-separated list of identifiers, and will cause `loggertodb` to ignore lines with different subset identifiers. (Which fields are the subset identifiers depends on the data file format.)

null Indicates how null values are represented in the source file. For example, if `null = *`, then a `*` in place of a number in the source file is interpreted as a missing value.

If the value is a number, e.g. `null = -9999`, then any string whose numeric value is that number will be interpreted as a missing value, e.g. `-9999`, `-9999.00` and `-9999.000000` will all be interpreted as missing values. The comparison is made with a tolerance of 1e-6.

(`nullstr` is a deprecated synonym of `null`.)

delimiter, decimal_separator, date_format Some storage formats may be dependent upon regional settings; these formats support `delimiter`, `decimal_separator`, and `date_format`. `date_format` is specified in the same way as for strftime(3).

ignore_lines For storage formats that are text files, it specifies a regular expression that, if it matches, the line will be ignored. This is useful to ignore header lines or otherwise lines that shouldn't be processed.

encoding For storage formats that are text files, it specifies the encoding. The default is utf8. List of possible encodings.

timezone The time zone of the timestamps of the data. This is necessary to know because Enhydris stores the timestamps in UTC, so if they are in another time zone they need to be converted. The value of this parameter is a name from the Olson database (see also Wikipedia's copy, which is handier than the official), such as `Europe/Athens`, but it can also be one the constant offset time zones such as `Etc/GMT` or `Etc/GMT-2`. If you use such a constant offset time zone, beware that the sign is reverted: `Etc/GMT-2` is 2 hours **east** of UTC.

While `loggertodb` will handle daylight saving time, it is recommended to configure your loggers to use a constant offset time zone. See *Daylight saving time* for more.

temperature_unit, rain_unit, wind_speed_unit, pressure_unit, matric_potential_unit In the wdat5 format, you can select some of the units; C or F for temperature, mm or inch for rain and evapotranspiration, m/s or mph for wind speed, hPa or inch Hg for pressure, centibar or cm (of water) for matric potential. The defaults are C, mm, m/s, hPa, centibar.

outsideTemp, hiOutsideTemp, etc. Only for wdat5 format; see its description below.

2.3 Supported formats

Don't create yet another conversion script

Many people think they should create a script to convert their file to a format that will be acceptable to `loggertodb` and then use `loggertodb` to read it. Don't do that. Don't have yet another script and yet another file—it increases the complexity of the system. If `loggertodb` does not support your existing file directly, contact us so that we add it (or add it yourself if you speak Python, the API is documented).

The following formats are currently supported:

simple The `simple` format is lines of which the first one or two fields are the date and time and the rest of the fields hold time series values. If the first field (after stripping any double quotation marks) is more than 10 characters in length, it is considered to be a date and time; otherwise it is a date only, and the second field is considered to

be the time; in this case the two fields are joined with a space to form the date/time string. The field delimiter is white space, unless the `delimiter` parameter is specified. The date and/or time and the values can optionally be enclosed in double quotation marks. The format of the date and time is specified by the `date_format` parameter (enclosing quotation marks are removed before parsing; also if the date and time are different fields, they are joined together with a space before being parsed). If `date_format` is not specified, then the date and time are considered to be in ISO8601 format, optionally using a a space instead of `T` as the date/time separator, and ignoring any seconds. If `date_format` is specified, it must include a second specifier if the times contain seconds, but these seconds are actually subsequently ignored.

The `nfields_to_ignore` parameter can be used to ignore a number of fields in the beginning of each line; this is useful in some formats where the date and time are preceeded by a record id or other field.

If `path` contains one of the characters **?[]*, it is considered to be a pattern that matches many files whose concatenation (ignoring any headers) would be the complete list of records. glob is used to find the matching files. `loggertodb` does not assume the filenames are ordered in any way; it determines the order by opening all the files and reading a date from each one.

CR1000 Date and time in ISO8601, the first two fields after the date are ignored (they are a record number and a station id), and uses subset identifiers in the next field. It is not clear whether it is debugged and works properly, neither whether its features are a matter of different data logger model or different data logger configuration.

deltacom The `deltacom` format is space-delimited lines of which the first field is the date and time in ISO8601 format `YYYY-MM-DDTHH:mm`, and the rest of the fields are either dummy or hold time series values, optionally followed by one of the four flags #, $, %, or &.

lastem The `lastem` format is dependent on regional settings, and uses the `delimiter`, `decimal_separator`, and `date_format` parameters. It is lines delimited with the specified delimiter, of which the first three fields are the subset identifiers, the fourth is the date, and the rest are either dummy or hold time series values.

pc208w The `pc208w` format is comma-delimited items in the following order: subset identifier, logger id (ignored), year, day of year, time in `HHmm`, measurements.

wdat5 The `wdat5` format is a binary format used by Davis WeatherLink; the files have a `wlk` extension. When using it, set `path` to the directory name where your `wlk` files are stored (one file per month).

You can specify time serie group ids like this:

```
outsideTemp = 1256
hiOutsideTemp = 1257
rain = 1652
```

The full list of variables is outsideTemp, hiOutsideTemp, lowOutsideTemp, insideTemp, barometer, outsideHum, insideHum, rain, hiRainRate, windSpeed, hiWindSpeed, windDirection, hiWindDirection, numWindSamples, solarRad, hiSolarRad, UV, hiUV, leafTemp1, leafTemp2, leafTemp3, leafTemp4, extraRad, newSensors1, newSensors2, newSensors3, newSensors4, newSensors5, newSensors6, forecast, ET, soilTemp1, soilTemp2, soilTemp3, soilTemp4, soilTemp5, soilTemp6, soilMoisture1, soilMoisture2, soilMoisture3, soilMoisture4, soilMoisture5, soilMoisture6, leafWetness1, leafWetness2, leafWetness3, leafWetness4, extraTemp1, extraTemp2, extraTemp3, extraTemp4, extraTemp5, extraTemp6, extraTemp7, extraHum1, extraHum2, extraHum3, extraHum4, extraHum5, extraHum6, extraHum7.

Many of these fields may be reserved by Davis for future use or they may not be used in the particular installation; just don't use them. It is also recommended to ignore the calculated values such as ET (evapotranspiration). More information about the meaning of the parameters can be found in the Davis manuals and in the WeatherLink README file.

odbc The sane place for loggers and logger software to store meteorological data is a plain text file. Databases shouldn't be used for that purpose. However, I've come across a system which was using MS Access, so I wrote this. It's only tested on Windows and MS Access, though in theory it should be usable anywhere. In that case, `path` is not actually a file name but an ODBC connection string, such as `DRIVER=Microsoft Access`

`Driver (*.mdb);DBQ=C:\Somewhere\mydb.mdb`. `table` specifies the database table in which the data is stored; each variable should be in a plain text column, and there should also be an `id` column indicating order. `date_sql` is an SQL expression that selects the date and time from the table (the resulting date and time format is defined by `date_format`). `data_columns` is a comma-separated list of (text) columns to retrieve from the table; `fields` must have as many entries as `data_columns`.

You see that this was a hack made for a specific installation, but if you are unfortunate enough to really need it, we can elaborate it further.

2.4 Daylight saving time

Important

Set your loggers to permanently use your winter time or any time that does not change.

In case this was not understood:

Set your loggers to permanently use your winter time or any time that does not change.

`Loggertodb` contains limited functionality to deal with cases where your loggers change time to DST. However, you should never, ever, use that functionality. Instead, you should configure your loggers to not do such an insane thing. If you use some kind of software+hardware stack that makes it necessary to configure your loggers to change to DST (something completely unnecessary, you can perfectly and easily store everything in one time zone and display it in another time zone), call your supplier and tell them they suck.

If you ignore this warning and set your loggers to use DST, don't expect `loggertodb` to do miracles. It can help of course, and it might work while things work smoothly. But whenever your government changes the date or time of the DST switch, or whenever something else goes wrong, you will be trying to fix a big mess instead of doing something useful. Really, you should get a life and set your loggers to permanently use your winter time or any time that does not change.

A time series is composed of records with timestamps. If we don't know exactly what these timestamps mean, the whole time series is meaningless. So, assuming you are in Germany, do you know exactly what 2012-10-28 02:30 means? No, you don't, because it might mean two different things. It could mean 02:30 CEST (00:30 UTC) or 02:30 CET (01:30 UTC). (In fact, several makes of loggers discard one of the two ambiguous hours during the switch from DST, meaning that if an incredible storm occurs at that time, you will lose it. Insane but true.)

Enhydris stores all timestamps in UTC so there's never any ambiguity and there's no switch to DST. The `timezone` parameter in the `loggertodb` configuration file is necessary so that the timestamps are correctly converted to UTC. If you are have loggers that switch to DST and are unable to change their configuration, `loggertodb` can attempt to convert it for you. In that case the `timezone` parameter should be set to a time zone that changes to DST, like "Europe/Athens":

```
timezone = Europe/Athens
```

`loggertodb` assumes that the time change occurs exactly when it is supposed to occur, not a few hours earlier or later. For the switch towards DST, things are simple. For the switch from DST to winter time, things are more complicated, because there's an hour that appears twice. If the ambiguous hour occurs twice, `loggertodb` will usually do the correct thing; it will consider that the second occurence is after the switch and the first is before the switch. If according to the computer's clock the switch hasn't occurred yet, any references to the ambiguous hour are considered to have occurred before the switch.

CHAPTER 3

Release notes

3.1 Version 3.0 (2022-12-05)

Briefly, this version is timezone-aware.

Enhydris 4.0 handles time zones better than 3.x, but it has some breaking changes in the API, namely that when uploading data the time zone must be specified. Accordingly, loggertodb 3.0 is compatible with Enhydris 4.0 and incompatible with previous Enhydris versions.

In this loggertodb version, the `timezone` configuration parameter must always be specified, and it is used to interpret all timestamps (whereas in previous versions it was used only for DST changes).

3.2 Version 2.2 (2021-01-13)

This makes loggertodb compatible with a new version of Enhydris that no longer has a "raw" time series type, and instead has an "initial" time series type. (This update has occurred in an internal unnumbered Enhydris release that precedes the release of Enhydris 3).

3.2.1 Changes in 2.2 microversions

2.2.1 (2022-11-06)

- Bug fix: If an error occurs during data uploading, the error is correctly shown and logged (and the traceback also logged in DEBUG level), whereas in 2.2.0 it crashed with an irrelevant message.

3.3 Version 2.1 (2020-11-17)

The `nullstr` parameter has been renamed to `null`, keeping `nullstr` as a deprecated synonym. If the value of `null` is a number, it is interpreted as a number instead of as a string.

3.4 Version 2.0 (2020-10-14)

`loggertodb` version 2 is not compatible with Enhydris versions earlier than 3. Use `loggertodb` version 1 for Enhydris version 2.

3.4.1 Token authentication

`loggertodb` used to use a username and password to logon to Enhydris. Now it uses a token instead. Accordingly, the `username` and `password` configuration file parameters have been abolished and `auth_token` has been introduced. See the *documentation on authentication* for more detailed instructions.

3.4.2 Time series groups

Enhydris 3 contains the notion of a time series group—a group of related time series for the same variable, e.g. the initial, checked and aggregated versions of a time series. Accordingly, `loggertodb` has been changed so that the `fields` parameter and the wdat5-specific meteorological parameters specify time series group ids rather than time series ids. `loggertodb` will always upload data in the "initial" time series of the time series group; if such a time series does not exist, it is automatically created.

3.4.3 How to upgrade from version 1

If you have a `loggertodb` v1 configuration file, `loggertodb` v2 can convert it. Enter this:

```
loggertodb --upgrade loggertodb.conf
```

where *`loggertodb.conf`* is the configuration file you want to upgrade. `loggertodb` will make requests to Enhydris in order to determine the `auth_token` from `username` and `password`, and the time series group ids from the time series ids. The configuration file will be upgraded accordingly. The original file will be backed up by adding the `.bak` extension (e.g. `loggertodb.conf.bak`). (If the backup file already exists and is different from the original, `loggertodb` will terminate with an error.)

3.4.4 Changes in 2.0 microversions

2.0.1 (2020-10-21)

- Fixed inability of the 2.0.0 Windows executable to run on Windows 7.

2.0.2 (2020-10-23)

- Fixed malfunctioning Windows executable (it had been built with wrong dependencies).

3.5 History up to Version 1

3.5.1 1.0.0 (2019-10-29)

- Improved handling of switch from DST to winter time.

3.5.2 0.2.2 (2019-08-20)

- Improved error message in multi-file simple format when nfields_to_ignore was 1 or more and a line did not have enough fields.

3.5.3 0.2.1 (2019-07-17)

- Fixed a crash when a file was empty in multi-file simple format.
- Improved error messages in multi-file simple format when the timestamps were badly ordered in a file or overlapping between files.

3.5.4 0.2.0 (2019-07-16)

- Added multi-file option to simple format.
- Added configuration parameters “encoding” and “ignore_lines”.

3.5.5 0.1.3 (2019-06-07)

- Upgraded htimeseries to 1.0.
- Made dependencies more robust.

3.5.6 0.1.2 (2019-05-27)

- Made parsing dates more robust in simple format.
- Fixed extreme slowness when thousands of records had to be inserted.
- Fixed unhelpful error message when file was out of order.

3.5.7 0.1.1 (2019-04-18)

- Fixed a bug that prevented using a log file.

3.5.8 0.1.0 (2019-04-18)

- Initial release

CHAPTER 4

For developers

Loggers are machines to which sensors are connected; they log the measurements of the sensors. In order to avoid confusion with Python Logger objects, here and in the code we call them **meteologgers**. But the user should not be bothered with that, so user documentation uses "loggers" for meteologgers.

4.1 MeteologgerStorage objects

In its simplest and most usual form, a "meteologger storage" is a text file to which the meteologger software appends records. Each line of the file has a timestamp and the values of the various measurements.

Some kinds of meteologgers (or meteologger software), however, use different kinds of storage. They may be automatically starting a new file daily or monthly. In such cases, the "meteologger storage" is a directory that has these files.

The `MeteoLoggerStorage` class is an abstract base class that is meant to be subclassed. It provides functionality that is common to all kinds of meteologger storages. Since each make of meteologger, or even each type of software used to unload data from a meteologger, provides a different kind of meteologger storage, MeteologgerStorage subclasses are specialized, each to a specific kind of meteologger storage.

class MeteologgerStorage(*parameters*[, *logger=None*])

A *MeteologgerStorage* instance should never be constructed; instead, a subclass should be constructed; however, the call for constructing any subclass has the same arguments. *parameters* is a dictionary containing values for configuration parameters such as *path* and *fields*. The *logger* argument is a Logger object to which error, progress, and debugging information is written.

When *MeteologgerStorage* is initialized, it checks the correctness of *parameters* and raises `ConfigurationError` if anything's wrong. See *get_required_parameters()* and *get_optional_parameters()* for more information.

The following *MeteologgerStorage* attributes are initialized from *parameters*:

path

The pathname to the storage; a filename or directory name.

null
A representation of values that will be treated as NaN (or null) (see *Usage* for more information).

(`nullstr` is a deprecated synonym of `null`.)

fields
A comma-separated list of integers representing the ids of the time series groups to which the fields correspond; a zero indicates that the field is to be ignored. The first number corresponds to the first field after the date (and other fixed fields, such as the possible subset identifier; which are those fields depends on the file format, that is, the specific *MeteologgerStorage* subclass) and should be the id of the corresponding time series group, or zero if the field is dummy; the second number corresponds to the second field after the fixed fields, and so on.

nfields_to_ignore
This is used only in the simple format; it's an integer that represents a number of fields before the date and time that should be ignored. The default is zero. If, for example, the date and time are preceded by a record id, set `nfields_to_ignore=1` to ignore the record id.

subset_identifiers
This is used only on some *MeteologgerStorage* subclasses. Some file formats mix two or more sets of measurements in the same file; for example, there may be ten-minute and hourly measurements in the same file, and for every 6 lines with ten-minute measurements there may be an additional line with hourly measurements (not necessarily the same variables). Such files have one or more additional distinguishing fields in each line, which helps to distinguish which set it is. We call these fields, which depend on the specific file format, the **subset identifiers**.

MeteologgerStorage (in fact its subclass) processes only one set of lines each time, and *subset_identifiers* specifies which subset it is. *subset_identifiers* is a comma-separated list of identifiers, and will cause *MeteologgerStorage* (in fact its subclass) to ignore lines with different subset identifiers.

delimiter

decimal_separator

date_format
Some file formats may be dependent on regional settings; these formats (i.e. these *MeteologgerStorage* subclasses) support *delimiter*, *decimal_separator*, and *date_format*. *date_format* is specified in the same way as for strftime.

MeteologgerStorage also has the following methods and properties:

timeseries_group_ids
A list of time series group ids. This is extracted from *fields* (zeros are ignored).

get_recent_data(*ts_group_id*, *ts_id*, *after_timestamp*)
Read the storage and extract the last part of the time series that is specified by *ts_group_id* and *ts_id*; specifically, provide the part that is more recent than *after_timestamp* (an aware `datetime` object). Returns that part of the time series as a pandas dataframe.

get_recent_data() will actually extract the last part of all time series from storage, but only return the data for the requested time series. It will cache the rest and have them ready to return for subsequent calls. However, if in subsequent calls *after_timestamp* is earlier than in the first call, it will need to re-extract the time series from storage. Therefore, for better performance, use the smallest *after_timestamp* first.

_raise_error(*line*, *msg*)
This is only meant to be used internally, i.e. called by subclasses whenever an error is found in a data file. The method raises an exception. *line* and *msg* are strings used in the error message.

_is_null(*value*)
This is only meant to be called by subclasses whenever they want to check whether a given value is null.

MeteologgerStorage subclasses need to define the following methods:

_subset_identifiers_match(*line*)
Return `True` if *line* matches the *subset_identifiers*. The base method always returns `True`, and subclasses only need to redefine it if the file format has subsets.

_extract_timestamp(*line*)
Parse *line* and extract and return the date and time as an aware datetime object.

_extract_value_and_flags(*line*, *seq*)
Extract the value and flags in sequence *seq* from *line*, and return it as a tuple. *seq*=1 is the first field after the fixed field, and so on (similar to *fields*).

get_required_parameters()
Return a set of parameters that are required. The base method returns {"path", "storage_format", "file_fields"} and must be overridden to add items to the list if the subclass requires more parameters.

get_optional_parameters()
Return a list of optional parameters. The base method returns an empty set and must be overridden if the subclass allows a different set.

CHAPTER 5

Copyright and credits

`loggertodb` was written by Antonis Christofides. It is derived from `autoupdate`, also written by Antonis Christofides, for the old openmeteo.org database. `loggertodb` is essentially `autoupdate` adapted to the hydria database for the Odysseus project, and later to the Enhydris database. This version of `loggertodb` has nothing to do with versions older than 2005, which were completely different, in a different programming language (Perl rather than Python), and not based on `autoupdate`.

This program was funded:

- In 2013-2017 by the TEI of Epirus as part of the IRMA project.
- In 2018-2021 by NTUA and ICCS as part of the OpenHi project, funded from the EU-Greece Sectoral Structural framework "Antagonistikotita".

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index