WeatherDB

Release 0.0.2

Max Schmit

Jan 18, 2022

CONTENTS:

CHAPTER

ONE

WEATHERDB - MODULE

author: Max Schmit

The weather-DB module offers an API to interact with the automaticaly filled weather Database.

Depending on the Database user privileges you can use more or less methodes of the classes.

There are 3 different sub modules with their corresponding classes.

- station: Has a class for every type of station. E.g. PrecipitationStation (or StationN). One object represents one Station with one parameter. This object can get used to get the corresponding timeserie. There is also a StationGroup class that groups the three parameters precipitation, temperature and evapotranspiration together for one station. If one parameter is not available this one won't get groupped.
- stations: Is a grouping class for all the stations of one measurement parameter. E.G. PrecipitationStations (or StationsN). Can get used to do actions on all the stations.
- broker: This submodule has only one class Broker. This one is used to do actions on all the stations together. Mainly only used for updating the DB.

1.1 Get started

To get started you need to enter the credentials to access the Database. If this is an account with read only acces, than only those methodes, that read data from the Database are available. Enter those credentials in the secretSettings.py file.

CHAPTER

TWO

WEATHERDB

2.1 weatherDB package

2.1.1 weatherDB.broker module

class weatherDB.broker.**Broker**

Bases: object

__init__()

create_roger_ts(*stid*, *folder*)

fillup(*paras=['n', 't', 'et']*)

Fillup the timeseries.

Parameters **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

initiate_db()

Initiate the Database.

Downloads all the data from the CDC server for the first time. Updates the multi-annual data and the richter-class for all the stations. Quality checks and fills up the timeseries.

last_imp_corr()

Richter correct the last imported precipitation data.

last_imp_fillup(*paras=['n', 't', 'et']*)

Fillup the last imported data.

Parameters **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

last_imp_quality_check(*paras=['n', 't', 'et']*, *with_fillup_nd=True*)

Quality check the last imported data.

Also fills up the daily precipitation data if the 10 minute precipitation data should get quality checked.

Parameters

- **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n", "t", "et"]. The default is ["n", "t", "et"].
- **with_fillup_nd** (*bool, optional*) – Should the daily precipitation data get filled up if the 10 minute precipitation data gets quality checked. The default is True.

new_import()

quality_check(*paras=['n', 't', 'et']*, *with_fillup_nd=True*)
[summary]

Parameters

- **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n", "t", "et"]. The default is ["n", "t", "et"].
- **with_fillup_nd** (*bool, optional*) – Should the daily precipitation data get filled up if the 10 minute precipitation data gets quality checked. The default is True.

update_db(*paras=['n_d', 'n', 't', 'et']*)
The regular Update of the database.

Downloads new data. Quality checks the newly imported data. Fills up the newly imported data.

Parameters **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

update_ma(*paras=['n_d', 'n', 't', 'et']*)
Update the multi-annual data from raster to table.

Parameters **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

update_meta(*paras=['n_d', 'n', 't', 'et']*)
Update the meta file from the CDC Server to the Database.

Parameters **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

update_period_meta(*paras=['n_d', 'n', 't', 'et']*)
Update the periods in the meta table.

Parameters **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

update_raw(*only_new=True*, *paras=['n_d', 'n', 't', 'et']*)
Update the raw data from the DWD-CDC server to the database.

Parameters

- **only_new** (*bool, optional*) – Get only the files that are not yet in the database? If False all the available files are loaded again. The default is True.
- **paras** (*list of str, optional*) – The parameters for which to do the actions. Can be one, some or all of ["n_d", "n", "t", "et"]. The default is ["n_d", "n", "t", "et"].

2.1.2 weatherDB.station module

class weatherDB.station.**EvapotranspirationStation**(*id*)
Bases: *weatherDB.station.StationTETBase*

__init__(*id*)

get_adj(*period=(None, None)*)
Get the adjusted timeserie.

The timeserie is adjusted to the multi annual mean. So the overall mean of the given period will be the same as the multi annual mean.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Returns A timeserie with the adjusted data.

Return type pandas.DataFrame

class weatherDB.station.**GroupStation**(*id*)

Bases: object

A class to group all possible parameters of one station.

__init__(*id*)

create_roger_ts(*dir*, *period=(None, None)*, *kind='best'*)

Create the timeserie files for RoGeR.

Parameters

- **dir** (*pathlib like object*) – The directory to store the timeseries in.
- **period** (*TimestampPeriod like object, optional*) – The period for which to get the timeseries. If (None, None) is entered, then the maximal possible period is computed. The default is (None, None)
- **kind** (*str*) – The data kind to look for filled period. Must be a column in the timeseries DB. Must be one of "raw", "qc", "filled", "adj". If "best" is given, then depending on the parameter of the station the best kind is selected. For Precipitation this is "corr" and for the other this is "filled". For the precipitation also "qn" and "corr" are valid.

Raises **Warning** – If there are NAs in the timeseries or the period got changed.

get_df(*period=(None, None)*, *kind='best'*, *paras='all'*)

Get a DataFrame with the corresponding data.

Parameters

- **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).
- **kind** (*str*) – The data kind to look for filled period. Must be a column in the timeseries DB. Must be one of "raw", "qc", "filled", "adj". If "best" is given, then depending on the parameter of the station the best kind is selected. For Precipitation this is "corr" and for the other this is "filled". For the precipitation also "qn" and "corr" are valid.

Returns A DataFrame with the timeseries for this station and the given period.

Return type pd.Dataframe

get_filled_period(*kind='best'*)

get_geom()

get_name()

get_possible_paras(*type='long'*)

Get the possible parameters for this station.

Returns A list of the long parameter names that are possible for this station to get.

Return type list of str

class weatherDB.station.**PrecipitationDailyStation**(*id*)
Bases: *weatherDB.station.StationNBase*

__init__(*id*)

property get_adj
(!) Disallowed inherited

property get_corr
(!) Disallowed inherited

property get_qc
(!) Disallowed inherited

property last_imp_quality_check
(!) Disallowed inherited

property quality_check
(!) Disallowed inherited

class weatherDB.station.**PrecipitationStation**(*id*)
Bases: *weatherDB.station.StationNBase*

__init__(*id*)

corr(*period=(None, None)*)

get_corr(*period=(None, None)*)

get_horizon()
Get the value for the horizon angle. (Horizontabschirmung)

This value is defined by Richter (1995) as the mean horizon angle in the west direction as: H'=0,15H(S-SW) +0,35H(SW-W) +0,35H(W-NW) +0, 15H(NW-N)

Returns The mean western horizon angle

Return type float or None

get_qn(*period=(None, None)*)

get_richter_class(*update_if_fails=True*)
Get the richter class for this station.

Provide the data from the meta table.

Parameters **update_if_fails** (*bool, optional*) – Should the richter class get updatet if no exposition class is found in the meta table? If False and no exposition class was found None is returned. The default is True.

Returns The corresponding richter exposition class.

Return type string

richter_correct(*period=(None, None)*)
Do the richter correction on the filled data for the given period.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Raises **Exception** – If no richter class was found for this station.

update_horizon(*skip_if_exist=True*)
Update the horizon angle (Horizontabschirmung) in the meta table.

Get new values from the raster and put in the table.

Parameters **skip_if_exist** (*bool, optional*) – Skip updating the value if there is already a value in the meta table. The default is True.

Returns The horizon angle in degrees (Horizontabschirmung).

Return type float

update_richter_class(*skip_if_exist=True*)
Update the richter class in the meta table.

Get new values from the raster and put in the table.

Parameters **skip_if_exist** (*bool, optional*) – Skip updating the value if there is already a value in the meta table. The default is True

Returns The richter class name.

Return type str

class weatherDB.station.**StationBase**(*id*)
Bases: object

__init__(*id*)

download_raw(*only_new=False*)

fillup(*period=(None, None)*)
Fill up missing data with measurements from nearby stations.

get_adj(*period=(None, None)*)
Get the adjusted timeserie.

The timeserie is adjusted to the multi annual mean. So the overall mean of the given period will be the same as the multi annual mean.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Returns A timeserie with the adjusted data.

Return type pandas.DataFrame

get_coef(*other_stid*)
Get the regionalisation coefficients due to the height.

Those are the values from the dwd grid or regnie grids.

Parameters **other_stid** (*int*) – The Station Id of the other station from wich to regionalise for own station.

Returns A list of

Return type list of floats or None

get_df(*kinds*, *period=(None, None)*, *db_unit=False*)
Get a timeseries DataFrame from the database.

Parameters

- **kinds** (*str or list of str*) – The data kinds to update. Must be a column in the timeseries DB. Must be one of "raw", "qc", "filled", "adj". For the precipitation also "qn" and "corr" are valid.
- **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).
- **db_unit** (*bool, optional*) – Should the result be in the Database unit. If False the unit is getting converted to normal unit, like mm or °C. The default is False.

Returns The timeserie Dataframe with a DatetimeIndex.

Return type pandas.DataFrame

get_dist(*period=(None, None)*)

Get the timeserie with the infomation from which station the data got filled and the corresponding distance to this station.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeserie. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Returns The timeserie for this station and the given period with the station_id and the distance in meters from which the data got filled from.

Return type pd.DataFrame

get_filled(*period=(None, None)*, *with_dist=False*)

Get the filled timeserie.

Either only the timeserie is returned or also the id of the station from which the station data got filled, together with the distance to this station in m.

Parameters

- **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeserie. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).
- **with_dist** (*bool, optional*) – Should the distance to the stations from which the timeseries got filled be added. The default is False.

Returns The filled timeserie for this station and the given period.

Return type pd.DataFrame

get_filled_period(*kind*)

Get the min and max Timestamp for which there is data in the corresponding table.

Parameters **kind** (*str*) – The data kind to look for filled period. Must be a column in the timeseries DB. Must be one of "raw", "qc", "filled", "adj". If "best" is given, then depending on the parameter of the station the best kind is selected. For Precipitation this is "corr" and for the other this is "filled". For the precipitation also "qn" and "corr" are valid.

Raises

- **NotImplementedError** – If the given kind is not valid.
- **ValueError** – If the given kind is not a string.

Returns A TimestampPeriod of the filled timeserie. (NaT, NaT) if the timeserie is all empty or not defined.

Return type util.TimestampPeriod

`get_geom`(*format='EWKT'*)
Get the point geometry of the station.

Parameters **format** (*str or None, optional*) – The format of the geometry to return. Needs to be a format that is understood by Postgresql. ST_AsXXXXX function needs to exist in postgresql language. If None, then the binary representation is returned. the default is "EWKT".

Returns string or bytes representation of the geometry, depending on the selected format.

Return type str or bytes

`get_geom_shp`()
Get the geometry of the station as a shapely Point object.

Returns The location of the station as shapely Point.

Return type shapely.geometries.Point

`get_last_imp_period`(*all=False*)
Get the last imported Period for this Station.

Parameters **all** (*bool, optional*) – Should the maximum Timespan for all the last imports be returned. If False only the period for this station is returned. The default is False.

Returns (minimal datetime, maximal datetime)

Return type TimespanPeriod or tuple of datetime.datetime

`get_ma`()

`get_meta`(*infos='all'*)
Get Informations from the meta table.

Parameters **infos** (*list of str or str, optional*) – A list of the informations to get from the database. If "all" then all the informations are returned. The default is "all".

Returns list with the informations.

Return type list

`get_multi_annual`()
Get the multi annual value(s) for this station.

Returns The corresponding multi annual value. For T en ET the yearly value is returned. For N the winter and summer half yearly sum is returned in tuple.

Return type list or number

`get_name`()

`get_neighboor_stids`(*n=5*, *only_real=True*)
Get a list with Station Ids of the nearest neighboor stations.

Parameters

- **n** (*int, optional*) – The number of stations to return. If None, then all the possible stations are returned. The default is 5.
- **only_real** (*bool, optional*) – Should only real station get considered? If false also virtual stations are part of the result. The default is True.

Returns A list of station Ids in order of distance. The closest station is the first in the list.

Return type list of int

`get_qc`(*period=(None, None)*)

`get_quality_checked`(*period=(None, None)*)

`get_raster_value`(*raster*)

`get_raw`(*period=(None, None)*)

`get_zipfiles`(*only_new=True*, *ftp_file_list=None*)
Get the zipfiles on the CDC server with the raw data.

Parameters

- **only_new** (*bool, optional*) – Get only the files that are not yet in the database? If False all the available files are loaded again. The default is True
- **ftp_file_list** (*list of (strings, datetime), optional*) – A list of files on the FTP server together with their modification time. If None, then the list is fetched from the server. The default is None

Returns A DataFrame of zipfiles and the corresponding modification time on the CDC server to import.

Return type pandas.DataFrame or None

`is_last_imp_done`(*kind*)
Is the last import for the given kind already worked in?

Parameters **kind** (*str*) – The data kind to look for filled period. Must be a column in the timeseries DB. Must be one of "raw", "qc", "filled", "adj", "best". If "best" is given, then depending on the parameter of the station the best kind is selected. For Precipitation this is "corr" and for the other this is "filled". For the precipitation also "qn" and "corr" are valid.

Returns True if the last import of the given kind is already treated.

Return type bool

`is_virtual`()
Check if the station is a real station or only a virtual one.

Real means that the DWD is measuring here. Virtual means, that there are no measurements here, but the station got created to have timeseries for every parameter for every precipitation station.

Returns true if the station is virtual, false if it is real.

Return type bool

`isin_db`()
Check if Station is already in a timeseries table.

Returns True if Station has a table in DB, no matter if it is filled or not.

Return type bool

`isin_ma`()
Check if Station is already in the multi annual table.

Returns True if Station is in multi annual table.

Return type bool

`isin_meta`()
Check if Station is already in the meta table.

Returns True if Station is in meta table.

Return type bool

last_imp_fillup(*last_imp_period=None*)
Do the filling up of the last import.

last_imp_quality_check()
Do the quality check of the last import.

plot(*period=(None, None)*, *kind='filled'*, ***kwargs*)
Plot the data of this station.

Parameters

- **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).
- **kind** (*str, optional*) – The data kind to plot. Must be a column in the timeseries DB. Must be one of "raw", "qc", "filled", "adj". For the precipitation also "qn" and "corr" are valid. The default is "filled.

quality_check(*period=(None, None)*)
Quality check the raw data for a given period.

Parameters **period** (*util.TimestampPeriod or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

update_ma(*skip_if_exist=True*)
Update the multi annual values in the stations_raster_values table.

Get new values from the raster and put in the table.

update_period_meta()
Update the time period in the meta file.

update_raw(*only_new=True*, *ftp_file_list=None*)
Download data from CDC and upload to database.

Parameters

- **only_new** (*bool, optional*) – Get only the files that are not yet in the database? If False all the available files are loaded again. The default is True
- **ftp_file_list** (*list of (strings, datetime), optional*) – A list of files on the FTP server together with their modification time. If None, then the list is fetched from the server. The default is None

Returns The raw Dataframe of the Stations data.

Return type pandas.DataFrame

weatherDB.station.**StationET**
alias of *weatherDB.station.EvapotranspirationStation*

weatherDB.station.**StationN**
alias of *weatherDB.station.PrecipitationStation*

class weatherDB.station.**StationNBase**(*id*)
Bases: *weatherDB.station.StationBase*

get_adj(*period=(None, None)*)
Get the adjusted timeserie.

The timeserie is adjusted to the multi annual mean. So the overall mean of the given period will be the same as the multi annual mean.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Returns A timeserie with the adjusted data.

Return type pandas.DataFrame

weatherDB.station.**StationND**
alias of *weatherDB.station.PrecipitationDailyStation*

weatherDB.station.**StationT**
alias of *weatherDB.station.TemperatureStation*

class weatherDB.station.**StationTETBase**(*id*)
Bases: *weatherDB.station.StationBase*

get_adj(*period=(None, None)*)
Get the adjusted timeserie.

The timeserie is adjusted to the multi annual mean. So the overall mean of the given period will be the same as the multi annual mean.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Returns A timeserie with the adjusted data.

Return type pandas.DataFrame

isin_meta_n()
Check if Station is in the precipitation meta table.

Returns True if Station is in the precipitation meta table.

Return type bool

quality_check(*period=(None, None)*)
Quality check the raw data for a given period.

Parameters **period** (*util.TimestampPeriod or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

class weatherDB.station.**TemperatureStation**(*id*)
Bases: *weatherDB.station.StationTETBase*

__init__(*id*)

get_adj(*period=(None, None)*)
Get the adjusted timeserie.

The timeserie is adjusted to the multi annual mean. So the overall mean of the given period will be the same as the multi annual mean.

Parameters **period** (TimestampPeriod *or (tuple or list of datetime.datetime or None), optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).

Returns A timeserie with the adjusted data.

Return type pandas.DataFrame

get_multi_annual()
Get the multi annual value(s) for this station.

Returns The corresponding multi annual value. For T en ET the yearly value is returned. For N the winter and summer half yearly sum is returned in tuple.

Return type list or number

weatherDB.station.**prop_disallow**
(!) You are not a super user of the Database and therefor this function is not available.

2.1.3 weatherDB.stations module

class weatherDB.stations.**EvapotranspirationStations**
Bases: *weatherDB.stations.StationsTETBase*

class weatherDB.stations.**GroupStations**
Bases: object

A class to group all possible parameters of all the stations.

__init__()

create_roger_ts(*dir*, *period=(None, None)*, *kind='best'*, *stids='all'*)
Get a list with all the stations as Station-objects.

Parameters

- **only_real** (*bool, optional*) – Whether only real stations are returned or also virtual ones. True: only stations with own data are returned. The default is True.
- **stids** (*string or list of int, optional*) – The Stations to return. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Returns returns a list with the corresponding station objects.

Return type Station-object

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

create_roger_ts_tempzip(*period=(None, None)*, *kind='best'*, *stids='all'*)

get_meta(*columns=['Station_id', 'von_datum', 'bis_datum', 'geometry']*)
Get the meta Dataframe from the Database.

Parameters

- **columns** (*list, optional*) – A list of columns from the meta file to return The default is: ["Station_id", "von_datum", "bis_datum", "geometry"]
- **only_real** (*bool, optional*) – Whether only real stations are returned or also virtual ones. True: only stations with own data are returned. The default is True.

Returns The meta DataFrame.

Return type pandas.DataFrame or geopandas.GeoDataFrae

get_stations(*stids='all'*)
Get a list with all the stations as Station-objects.

Parameters **stids** (*string or list of int, optional*) – The Stations to return. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Returns returns a list with the corresponding station objects.

Return type Station-object

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

class weatherDB.stations.**PrecipitationDailyStations**
Bases: *weatherDB.stations.StationsBase*

class weatherDB.stations.**PrecipitationStations**
Bases: *weatherDB.stations.StationsBase*

update_richter_class(*stids='all'*)
Update the Richter exposition class.

Get the value from the raster, compare with the richter categories and save to the database.

Parameters **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

class weatherDB.stations.**StationsBase**
Bases: object

__init__()

download_meta()
Download the meta file(s) from the CDC server.

Returns The meta file from the CDC server. If there are several meta files on the server, they are joined together.

Return type geopandas.GeoDataFrame

fillup(*only_real=False*, *stids='all'*)
Fill up the quality checked data with data from nearby stations to get complete timeseries.

Parameters

- **only_real** (*bool, optional*) – Whether only real stations are computed or also virtual ones. True: only stations with own data are returned. The default is True.
- **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

get_meta(*columns=['Station_id', 'von_datum', 'bis_datum', 'geometry']*, *only_real=True*)
Get the meta Dataframe from the Database.

Parameters

- **columns** (*list, optional*) – A list of columns from the meta file to return The default is: ["Station_id", "von_datum", "bis_datum", "geometry"]

- **only_real** (*bool, optional*) – Whether only real stations are returned or also virtual ones. True: only stations with own data are returned. The default is True.

Returns The meta DataFrame.

Return type pandas.DataFrame or geopandas.GeoDataFrae

get_stations(*only_real=True*, *stids='all'*)
Get a list with all the stations as Station-objects.

Parameters

- **only_real** (*bool, optional*) – Whether only real stations are returned or also virtual ones. True: only stations with own data are returned. The default is True.
- **stids** (*string or list of int, optional*) – The Stations to return. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Returns returns a list with the corresponding station objects.

Return type Station-object

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

last_imp_fillup()
Do the filling of the last import.

last_imp_quality_check()
Do the quality check of the last import.

quality_check(*period=(None, None)*, *only_real=True*, *stids='all'*)
Quality check the raw data for a given period.

Parameters

- **period** (*tuple or list of datetime.datetime or None, optional*) – The minimum and maximum Timestamp for which to get the timeseries. If None is given, the maximum or minimal possible Timestamp is taken. The default is (None, None).
- **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

update_ma(*stids='all'*)
Update the multi annual values for the stations.

Get a multi annual value from the corresponding raster and save to the multi annual table in the database.

Parameters **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

update_meta()
Update the meta table by comparing to the CDC server.

The "von_datum" and "bis_datum" is ignored because it is better to set this by the filled period of the stations in the database. Often the CDC period is not correct.

update_period_meta(*stids='all'*)
Update the period in the meta table of the raw data.

Parameters **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

update_raw(*only_new=True*, *only_real=True*, *stids='all'*)
Download all stations data from CDC and upload to database.

Parameters

- **only_new** (*bool, optional*) – Get only the files that are not yet in the database? If False all the available files are loaded again. The default is True
- **only_real** (*bool, optional*) – Whether only real stations are tried to download. True: only stations with a date in von_datum in meta are downloaded. The default is True.
- **stids** (*string or list of int, optional*) – The Stations to return. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

weatherDB.stations.**StationsET**
alias of *weatherDB.stations.EvapotranspirationStations*

weatherDB.stations.**StationsN**
alias of *weatherDB.stations.PrecipitationStations*

weatherDB.stations.**StationsND**
alias of *weatherDB.stations.PrecipitationDailyStations*

weatherDB.stations.**StationsT**
alias of *weatherDB.stations.TemperatureStations*

class weatherDB.stations.**StationsTETBase**
Bases: *weatherDB.stations.StationsBase*

fillup(*only_real=False*, *stids='all'*)
Fill up the quality checked data with data from nearby stations to get complete timeseries.

Parameters

- **only_real** (*bool, optional*) – Whether only real stations are computed or also virtual ones. True: only stations with own data are returned. The default is True.
- **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

update_exp_fact(*stids='all'*)
Update the exposition factors.

Get the value from the raster and save to the database.

Parameters **stids** (*string or list of int, optional*) – The Stations for which to compute. Can either be "all", for all possible stations or a list with the Station IDs. The default is "all".

Raises **ValueError** – If the given stids (Station_IDs) are not all valid.

class weatherDB.stations.**TemperatureStations**
Bases: *weatherDB.stations.StationsTETBase*

2.1.4 Subpackages

weatherDB.lib package

weatherDB.lib.connections module

class weatherDB.lib.connections.**FTP**(*host=''*, *user=''*, *passwd=''*, *acct=''*, *timeout=<object object>*, *source_address=None*, ***, *encoding='utf-8'*)

Bases: ftplib.FTP

login(***kwargs*)
Login, default anonymous.

weatherDB.lib.utils module

class weatherDB.lib.utils.**TimestampPeriod**(*start*, *end*)

Bases: object

A class to save a Timespan with a minimal and maximal Timestamp.

COMPARE = {'inner': {0: <built-in function max>, 1: <built-in function min>}, 'outer': {0: <built-in function min>, 1: <built-in function max>}}

__init__(*start*, *end*)
Initiate a TimestampPeriod.

Parameters

- **start** (*pd.Timestamp or similar*) – The start of the Period.
- **end** (*pd.Timestamp or similar*) – The end of the Period.

contains(*other*)

get_period()

get_sql_format_dict(*format='%Y%m%d %H:%M'*)

has_NaT()

has_only_NaT()

inside(*other*)

is_empty()

strftime(*format='%Y-%m-%d %H:%M:%S'*)

union(*other*, *how='inner'*)
Unite 2 TimestampPeriods to one.

Compares the Periods and computes a new one.

Parameters

- **other** (TimestampPeriod) – The other TimestampPeriod with whome to compare.
- **how** (*str, optional*) – How to compare the 2 TimestampPeriods. Can be “inner” or “outer”. “inner”: the maximal Timespan for both is computed. “outer”: The minimal Timespan for both is computed. The default is “inner”.

Returns A new TimespanPeriod object uniting both TimestampPeriods.

Return type *TimestampPeriod*

weatherDB.lib.utils.**get_ftp_file_list**(*ftp_conn*, *ftp_folders*)
Get a list of files in the folders with their modification dates.

Parameters

- **ftp_conn** (*ftplib.FTP*) – Ftp connection.
- **ftp_folders** (*list of str or pathlike object*) – The directories on the ftp server to look for files.

Returns A list of Tuples. Every tuple stands for one file. The tuple consists of (filepath, modification date).

Return type list of tuples of strs

Subpackages

weatherDB.lib.max_fun package

weatherDB.lib.max_fun.import_DWD module

A collection of functions to import data from the DWD-CDC Server.

class weatherDB.lib.max_fun.import_DWD.**FTP**(*host=''*, *user=''*, *passwd=''*, *acct=''*, *timeout=<object object>*, *source_address=None*, *, *encoding='utf-8'*)
Bases: ftplib.FTP

login(***kwargs*)
Login, default anonymous.

weatherDB.lib.max_fun.import_DWD.**dwd_id_to_str**(*id*)
Convert a station id to normal DWD format as str.

Parameters **id** (*int or str*) – The id of the station.

Returns string of normal DWD Station id.

Return type str

weatherDB.lib.max_fun.import_DWD.**get_dwd_data**(*station_id*, *ftp_folder*)
Get the weather data for one station from the DWD server.

Parameters

- **station_id** (*str or int*) – Number of the station to get the weather data from.
- **ftp_folder** (*str*) – the base folder where to look for the stations_id file. e.g. ftp_folder = "climate_environment/CDC/observations_germany/climate/hourly/precipitation/historical/". If the parent folder, where "recent"/"historical" folder is inside, both the historical and recent data gets merged.

Returns The DataFrame of the selected file in the zip folder.

Return type pandas.DataFrame

weatherDB.lib.max_fun.import_DWD.**get_dwd_file**(*zip_filepath*)
Get a DataFrame from one single (zip-)file from the DWD FTP server.

Parameters **zip_filepath** (*str*) – Path to the file on the server. e.g.

- ”/climate_environment/CDC/observations_germany/climate/10_minutes/air_temperature/recent/10minutenwerte_T
- ”/climate_environment/CDC/derived_germany/soil/daily/historical/derived_germany_soil_daily_historical_73.txt.g

Returns The DataFrame of the selected file in the zip folder.

Return type pandas.DataFrame

weatherDB.lib.max_fun.import_DWD.**get_dwd_meta**(*ftp_folder*, *min_years=0*, *max_hole_d=9999*)

Get the meta file from the ftp_folder on the DWD server.

Downloads the meta file of a given folder. Corrects the meta file of missing files. So if no file for the station is in the folder the meta entry gets deleted. Reset “von_datum” in meta file if there is a biger gap than max_hole_d. Delets entries with less years than min_years.

Parameters

- **ftp_folder** (*str*) – The path to the directory where to search for the meta file. e.g. “climate_environment/CDC/observations_germany/climate/hourly/precipitation/recent/”.
- **min_years** (*int, optional*) – filter the list of stations by a minimum amount of years, that they have data for. 0 if the data should not get filtered. Only works if the meta file has a timerange defined, e.g. in “observations”. The default is 0.
- **max_hole_d** (*int*) – The maximum amount of days missing in the data allowed. If there are several files for one station and the time hole is biger than this value, the older “von_datum” is overwriten in the meta GeoDataFrame. The default is 2.

Returns a GeoDataFrame of the meta file

Return type geopandas.GeoDataFrame

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W