WeatherDB

Release 0.0.40

Max Schmit

Aug 28, 2023

CONTENTS:

CHAPTER

ONE

WEATHERDB - MODULE

author:

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

Depending on the Database user privileges you can use more or less methods 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 GroupStation 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 grouped.
- 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 Install

To install the package use PIP to install the Github repository:

Or to upgrade use:

1.2 Get started

To get started you need to enter the credentials to access the Database. If this is an account with read only access, than only those method's, that read data from the Database are available. Enter those credentials in the secretSettings_weatherDB.py file. An example secretSettings_weatherDB.py file is in the source directory (see secretSettings_weatherDB_example.py)

If you use the database at the hydrology department of Freiburg, please go to the weather.hydro.intra.uni-freiburg.de. There you can create yourself an account and then download your login secretSettings file from your profile page, next to the "API Password".

The secretSettings_weatherDB.py file needs to be placed either:

- in a parent folder of the package (e.g. in the main folder of your virtual environment folder)
- some other directory that is in the PYTHONPATH environment variable. (You can also create a new directory and add it to the PATH environment)

- in the package source folder (e.g. ../path_to_venv/Lib/site-packages/weatherDB) !This might not be the best method, because an upgrade of the package could delete the file again!

1.3 How-to install python

To use this package you obviously need Python with several packages installed.

The easiest way to install python is by installing Anaconda.

After the installation you should create yourself a virtual environment. This is basically a folder with all your packages installed and some definition files to set the appropriate environment variables… To do so use (in Anaconda Terminal):

Afterwards you need to activate your environment and then install the requirements:

CHAPTER

TWO

QUICK-START

After installing and setting up the secretSettings_weatherDB.py file you are ready to use the package. This page should show you the basic usage of the package.

The package is divided in 2 main submodules:

- **weatherDB.station:** This module has a class for every type of station. E.g. StationN (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.
- **weatherDB.stations:** This module has grouping classes for all the stations of one parameter. E.G. StationsN (or StationsN) groups all the Precipitation Stations available. Those classes can get used to do actions on all the stations.

The basics of those modules are explained here, but every class and method has way more parameters to get exactly what you want. PLease use the API-reference for more information.

2.1 download

2.1.1 single station

If you want to download data for a single station, you have to create an object of the respective class, by gibing the station id you are interested in. This station object can then get used to download the data from the database.

If you want e.g. to download all the 10 minute, filled and Richter corrected precipitation data for the DWD station in Freiburg(station ID = 1443), then you can go like:

```
from weatherDB import StationN
stat_n = StationN(1443)
df = stat_n.get_corr()
```

If you are only interested in a small timespan, provide the period parameter with the upper and lower time limit. If e.g. you want the data for the years 2000-2010 do:

```
df = stat_n.get_corr(period=("2000-01-01", "2010-12-31"))
```

If you are not interested in the filled and Richter-corrected data, but want e.g. the raw data, add the kind parameter to your query. Like e.g.:

```
df = stat_n.get_raw()
```

Or use the more general function with the wanted kind parameter.

```
df = stat_n.get_df(kinds=["raw"])
```

There are 3-5 different kinds of timeseries available per station object depending on the class. So there is:

- "raw" : the raw measurements as on the DWD server
- "qc" : The quality checked data
- "filled" : The filled timeseries
- "filled_by" : The station ID of the station from which the data was taken to fill the measurements
- "corr" : The Richter corrected timeserie.

If you want more than just one kind of timeseries, e.g. the filled timeseries, together with the id from which station the respective field got filled with use:

```
df = stat_n.get_df(kinds=["filled", "filled_by"])
```

If you only need daily values, you can hand in the agg_to parameter. This will also make your query faster, because not as much data has to get transmitted over the network.

```
df = stat_n.get_df(agg_to="day")
```

Similar to the precipitation, you can also work with the Temperature and potential Evapotranspiration data:

```
from weatherDB import StationT, StationET
stat_t = StationT(1443)
stat_et = StationET(1443)
period = ("2000-01-01", "2010-12-31")
df_t = stat_t.get_df(
    period=period,
    kinds=["raw", "filled"])
df_et = stat_t.get_df(
    period=period,
    kinds=["raw", "filled"])
```

So to download the 3 parameters N, T and ET from one station you could create the 3 single station objects and then have 3 different timeseries. But the better solution is to use the GroupStation class. This class groups all the available parameters for one location. Here is an example, how you could use it to get a Dataframe with the filled data:

```
from weatherDB import GroupStation
stat = GroupStation(1443)
df = stat.get_df(
    period=("2000-01-01", "2010-12-31"),
    kind="filled",
    agg_to="day")
```

2.1.2 multiple stations

If you want to download the data for multiple stations. Like e.g. the station in Freiburg (1443) and the station on the Feldberg (1346) it is recommended to use the classes in the stations module.

To use the stations-module, you first have to create an object and then hand the station ids you are interested in when downloading it:

```
from weatherDB import StationsN
stats_n = StationsN()
df = stats_n.get_df(
    stids=[1443, 1346],
    period=("2000-01-01", "2010-12-31"),
    kind="filled")
```

2.2 create timeseries files

You can also use the module to quickly create the csv-timeseries needed by RoGeR. Either for one station:

```
from weatherDB import GroupStation
stat = GroupStation(1443)
df = stat.create_roger_ts(
    dir="path/to/the/directory/where/to/save")
```

or for multiple stations, you can use the GroupStations. This will create a subdirectory for ever station. It is also possible to save in a zip file, by simply giving the path to a zip file. (will get created):

```
from weatherDB import GroupStations
stats = GroupStations()
df = stats.create_roger_ts(
    stids=[1443, 1346],
    dir="path/to/the/directory/where/to/save")
```

If you don't want to use the RoGeR format for the timestamp you can use the `.create_ts()` method. This method also offers you way more possibilities to define the output, like e.g. adding the share of NAs in the aggregation step or adding the filled_by column.

```
from weatherDB import GroupStations
stats = GroupStations()
df = stats.create_ts(
    stids=[1443, 1346],
    dir="path/to/the/directory/where/to/save")

# or for one station
from weatherDB import GroupStation
stat = GroupStation(1443)
df = stat.create_ts(
    dir="path/to/the/directory/where/to/save")
```

2.3 get meta information

If you need more information about the stations you can get the meta data for a single station:

```
from weatherDB import StationN
stat = StationN(1443)
meta_dict = stat.get_meta()
```

or for multiple stations, you can use the Stations class and get a GeoDataFrame as output with all the stations information.

```
from weatherDB import StationsN
stats = StationsN(
    stids=[1443, 1346])
df = stats.get_meta()
```

Furthermore you can also get all the information for every parameter of one station by using the GroupStation class:

```
from weatherDB import GroupStation
gstat = GroupStation(1443)
df = gstat.get_meta()
```

To get an explanation about the available meta information you can use the get_meta_explanation method:

```
from weatherDB import StationN, StationsN
stat = StationN(1443)
explain_df = stat.get_meta_explanation()
# or
stats = StationsN()
explain_df = stats.get_meta_explanation()
```

If you are only interested in some information you can use the infos parameter like:

```
from weatherDB import StationN
stat = StationN(1443)
filled_period_1443 = stat.get_meta(infos=["filled_from", "filled_until"])
```

but to get the filled period you can also use the get_period method, like:

```
from weatherDB import StationN
stat = StationN(1443)
filled_period_1443 = stat.get_period_meta(kind="filled")
```

This should give you an first idea on how to use the package. Feel free to try out and read the API reference section to get more information.

CHAPTER

THREE

METHOD

Behind this package/website is a PostGreSQL-database. This database is build and updated with the same package, as for downloading the data. The only difference is, that the given connection in the secretSettings file needs to have write permissions for the database. Therefor everyone can look into the code to find out exactly how the database creation works. But as an overview this page will give basic explanations of the processes behind it.

The timeseries for Temperature, Evapotranspiration and Precipitation are going through a 3-4 step process.The result of every process is saved and can get downloaded, with the corresponding abbreviation:

- downloading the raw data –> "raw"
- quality check the data –> "qc"
- fillup the data –> "filled"
- richter correct the values –> "corr"

In the following chapters the processes will get explained furthermore.

3.1 downloading the data

The raw data is downloaded from the DWD-CDC server. The timeseries are downloaded and saved from the 1.1.1994 on. If there is historical data available for a measurement, they are preferred over recent values, because they are already quality checked a bit. The Temperature (T) and potential Evapotranspiration (ET) is downloaded on daily resolution. Where as the Precipitation (N) is downloaded as 10 minute and daily values, but only the 10 minute values are the basis for the downloads.

Table 1: The downloaded raw data, resolution and their source

parameter	resolution	source
Temperature	daily	DWD Climate Data Center (CDC): Historical daily station observations (temperature, pressure, precipitation, sunshine duration, etc.) for Germany, version v21.3, 2021, online available DWD Climate Data Center (CDC): Recent daily station observations (temperature, pressure, precipitation, sunshine duration, etc.) for Germany, quality control not completed yet, version recent, online available
pot. Evapotranspiration	daily	DWD Climate Data Center: Calculated daily values for different characteristic elements of soil and crops., Version v19.3, 2019, parameter "VPGB", online available DWD Climate Data Center: Calculated daily values for different characteristic elements of soil and crops,Version v19.3, 2019, parameter "VPGB", online available
Precipitation	10 minutes	DWD Climate Data Center (CDC): Historical 10-minute station observations of precipitation for Germany, version V1, 2019, online availableDWD Climate Data Center (CDC): Recent 10-minute station observations of precipitation for Germany,version recent, 2019, online available
Precipitation	daily	DWD Climate Data Center (CDC): Historical daily station observations (temperature, pressure, precipitation, sunshine duration, etc.) for Germany, version v21.3, 2021, online available DWD Climate Data Center (CDC): Recent daily station observations (temperature, pressure, precipitation, sunshine duration, etc.) for Germany, quality control not completed yet, version recent, online available

For computation improvements the downloaded files and their modification time is saved to the database, to be able to only download the updated data.

3.2 quality check

To quality check the data it is very dependent on which parameter is treated. Therefor this chapter is grouped into subchapters.

Although every quality check can get computed for different periods:

- on all of the data by using e.g. `station.StationT(3).quality_check()`
- only a specified period, with e.g. `station.StationN(3).quality_check(period=("2010-01-01", "2020-12-31"))`
- the last imported period, with e.g. `station.StationET(3).last_imp_quality_check()`

3.2.1 Temperature and Evapotranspiration

For T and ET quality check the data is compared to the 5 nearest neighbors data. To get the nearest stations, the selection is done on a yearly basis. For this neighbor stations having data for more than 6 months in that year are considered.

Furthermore also the difference in elevation between the two stations is considered to select the closest stations. This is done, because the topographie has a big influence on the Temperature and Evapotranspiration. The weighted distance is thereby calculated with the LARSIM formula:

$L_{gewichtet} = L_{horizontal} * (1 + (\frac{|\delta H|}{P_1})^{P_2})$

In Larsim those parameters are defined as $P_1 = 500$ and $P_2 = 1$. Stoelzle et al. (2016) found that $P_1 = 100$ and $P_2 = 4$ is better for Baden-Würtemberg to consider the quick changes in topagraphie. or all of Germany, those parameter values are giving too much weight to the elevation difference, which can result in getting neighbor stations from the border of the Czech Republic for the Feldberg station. Therefor the values $P_1 = 250$ and $P_2 = 1.5$ are used as default values.

The data of every neighbor station is regionalized, based on the DWD grids (see chapter regionalisation), to the examined station. Then the mean value of those 5 values is taken to compare to the station data. If this mean value is too far away from the own value, then the measurement point is deleted.

Table 2: Limits for the quality check of the T and ET measurements, when compared with neighbor stations

parameter	compare equation	lower limit	upper limit	treshold
Temperature	$\Delta T = T_{Stat} - \overline{T}_{neighbors}$	$\Delta T < -5C$	$\Delta T > 5C$	$-50C < T_{Stat} < 50C$
pot. Evapotranspiration	$\delta ET = \frac{ET_{Stat}}{\overline{ET}_{neighbors}}$	$\begin{cases} \delta ET < 25\% \\ ET_{Stat} > 2\frac{mm}{d} \end{cases}$	$\begin{cases} \delta ET > 200\% \\ ET_{Stat} > 3\frac{mm}{d} \end{cases}$	$0\frac{mm}{d} =< ET_{Stat} < 200\frac{mm}{d}$

For the evapotranspiration there are two rules that need to be fulfilled to be unplausible. One relative and one nominal. This is because, low measurement values tend to have high relative differences and would then get deleted too often.

To consider the possible inversion weather phenomena, stations higher than 800 m.a.S. are only tested against the lower limit in winter months (October-March).

Furthermore there are maximum and minimum treshold values to which the ET and T values are compared to. Those treshold values are set to reasonable values for germany.

3.2.2 Precipitation

The precipitation measurements must pass through multiple quality checks.

3.2.2.1 daily sum is zero

As some precipitation station (e.g. Feldberg station) have measurements of 0mm in the 10 minutes dataset even though the daily dataset has measurements. This is especially true for the measurement points in 20th and early 21th century. This is probably the result of new measurement equipment to measure the 10 minutes values, but without a good quality control. back then the daily values are often measured with other equipments and have a better quality check they are going through, so that the values are more reliable.

To filter those wrongly measurements of 0 mm out of the data, the data is compared to the daily values from the DWD at the same location. For this reason the daily measurements are first filled up (see next chapter). The 10 minutes measurements get aggregated to daily values. If this daily sum is 0 mm, even though the daily data from the DWD is not 0, all the 10 minutes measurements of that day are deleted. Furthermore if the daily measurement is more than twice the aggregated 10 minutes values, all the 10 minutes data of that day are removed.

This check is the only reason why the daily precipitation values were downloaded in the first place.

3.2.2.2 consecutive equal values

Sometimes there are several consecutive 10 minutes values that are exactly the same. As the accuracy of the measurement is 0.01 mm, this is very improbable to be a real measurement and is more likely the result of splitting e.g. an hourly value into 6 values.

It is assumed, that filling the measurements up with values from the neighbor stations is more accurate than this dissemination. Therefor 3 consecutive same measurements are deleted, if their “Qualitätsnorm” from the DWD is not 3 (meaning that the measurements didn’t get a good quality control from the DWD).

3.3 gap filling

To have complete timeseries, the gaps in the quality checked timeseries are filled with data from the neighbor stations. The neighboring stations are selected in the order of horizontal difference for the precipitation stations and in order of the elevation weighted distance (see chapter quality check - Temperature and Evapotranspiration) for T and ET stations. This is done by regionalising the neighbors measurements value to the station that is gap filled. Starting with the nearest neighbor station all available stations are taken until the timeserie is completely filled.

For the reginalisation, the multi-annual values for every station for the climate period of 1991-2020 are computed from the corresponding DWD grid.

Table 3: The raster grids that are the basis for the regionalisation

parameter	source
Precipitation	DWD Climate Data Center (CDC): HYRAS grids of multi-annual precipitation, period 1991-2020, online available
Temperature	DWD Climate Data Center (CDC): Multi-annual means of grids of air temperature (2m) over Germany, period 1991-2020, version v1.0. online available
potential Evapotranspiration	DWD Climate Data Center (CDC): Multi-annual grids of potential evapotranspiration over grass, 0.x, period 1991-2020, online available

As those grids have a coarse resolution with 1 km2, they got refined to a resolution of 25 meters, on the basis of a DEM25 (Copernicus). To refine the rasters the 1 km2 DEM that was used by the DWD, was used. Together with the multi-annual raster value of the neighbor cells a linear regression is defined for every cell. The size of the window to produce the linear regression depends on the topology. Starting with a 5 x 5 km window, the standard deviation of the topology is computed. If this is smaller than 4 meters, than the window is increased by 1 km to each side. This step is repeated until the standard deviation is greater than 4 meters or the size of the window is greater than 13 x 13 km. This regression is then used on the DEM25 cells inside the 1 km2 cell, to calculate the new multi-annual values.

Then to get a regionalisation factor the multi-annual values of both stations are compared. For T and ET only the yearly mean is taken into account. For the precipitation one winter(October-March) and one summer (April-September) factor is computed. The following equation explain the calculation of the filling values for the different parameters, based on their multi-annual mean(ma).

$$T_{fillup} = T_{neighbor} + (T_{station,ma} - T_{neighbor,ma})$$

$$ET_{fillup} = ET_{neighbor} * \frac{ET_{station,ma}}{ET_{neighbor,ma}}$$

$$N_{fillup} = \begin{cases} N_{neighbor} * \frac{N_{station,ma,winter}}{N_{neighbor,ma,winter}} if month \in [4:9] \\ N_{neighbor} * \frac{N_{station,ma,summer}}{N_{neighbor,ma,summer}} if month \notin [4:9] \end{cases}$$

For the precipitation and evapotranpiration stations only the closest station with quality checked data is taken to fill missing values. For the temperature stations the median of the regionalised values from the 5 closest stations (but not more than 100 km away) to fill missing values.

For the precipitation values the 10 minutes values are furthermore adjusted to the daily measurements. Therefor the daily sum is computed. Then the quotient with the daily measurement is calculated and multiplied to every 10 minute measurement. So the difference to the daily measurement is added relatively to the measured value. In the end the gap filled 10 minutes precipitation values sum up to the same daily values as the daily values from the DWD.

3.4 Richter correction

This step is only done for the 10 minutes precipitation values. Here the filled precipitation values, get corrected like defined in Richter (1995).

First of all, the horizon angle ("Horizontabschirmung") is calculated from a DGM25 and if the DGM25 was out of bound also from a DGM80. The DGM80 is bigger than the german border and therefor for stations around the border this is gives better results than the DGM20 which is only for the german territory. Therefore the rasters are sampled for their values on one single line of 75km, starting from the station. Then the angle to every point from the station is calculated. The Point with the biggest angle is taken as horizon angle for this line. This step is repeated for several lines ranging from north to south in 3° steps. Afterwards the Richter horizon angle is computed as:

$$H' = 0,15 * H_{S-SW} + 0,35 * H_{SW-W} + 0,35 * H_{W-NW} + 0,15 * H_{NW-N}$$

With this horizon angle the Richter exposition class is defined for every station.

Afterwards the daily correction is calculated with the following table and equation, based on the filled daily temperature at the station.

$$\Delta N = b * N^E$$

Table 4: The Richter correction coeeficients. (Richter (1995) S. 67)

precipitation typ	temperature	E	bno-protection	blittle-protection	bpro-tected	bheavy-protection
precip_sommer	>= 3 °C	0,38	0,345	0,31	0,28	0,245
precip_winter	>= 3 °C	0,46	0,34	0,28	0,24	0,19
mix	-0,7 °C < T < 3 °C	0,55	0,535	0,39	0,305	0,185
snow	<= -0,7°C	0,82	0,72	0,51	0,33	0,21

The daily correction (ΔN) is then distributed to every 10 minute measurement where precipitation was measured. So the daily correction is applied as a block to the 10 minutes values. This results in relatively high corrections, when there was only little precipitation and relatively low corrections when the measured precipitation was high. As the systematic error is mostly due to wind and has therefor more effect, when there is low precipitation, this approach is better than adding it relatively to the measurement.

3.5 Sources

- Richter, D. 1995. Ergebnisse methodischer Untersuchungen zur Korrektur des systematischen Meßfehlers des Hellmann-Niederschlagsmessers. Offenbach am Main: Selbstverl. des Dt. Wetterdienstes.
- Coperniicus. 2016. European Digital Elevation Model (EU-DEM), version 1.1. online available
- Stoelzle, Michael & Weiler, Markus & Steinbrich, Andreas. (2016) Starkregengefährdung in Baden-Württemberg – von der Methodenentwicklung zur Starkregenkartierung. Tag der Hydrologie.
- LARSIM Dokumentation, Stand 06.04.2023, online unter https://www.larsim.info/dokumentation/LARSIM-Dokumentation.pdf

CHAPTER

FOUR

WEATHERDB

4.1 weatherDB

4.1.1 broker

4.1.1.1 Broker

class `weatherDB.broker.`**`Broker`**

Bases: `object`

A class to manage and update the database.

Can get used to update all the stations and parameters at once.

This class is only working with super user privileges.

Public Methods:

`__init__()`	
update_raw([only_new, paras])	Update the raw data from the DWD-CDC server to the database.
update_meta([paras])	Update the meta file from the CDC Server to the Database.
update_ma([paras])	Update the multi-annual data from raster to table.
update_period_meta([paras])	Update the periods in the meta table.
quality_check([paras, with_fillup_nd])	Do the quality check on the stations raw data.
last_imp_quality_check([paras, with_fillup_nd])	Quality check the last imported data.
fillup([paras])	Fillup the timeseries.
last_imp_fillup([paras])	Fillup the last imported data.
richter_correct()	Richter correct all of the precipitation data.
last_imp_corr()	Richter correct the last imported precipitation data.
update_db([paras])	The regular Update of the database.
initiate_db()	Initiate the Database.
vacuum([do_analyze])	
get_setting(key)	Get a specific settings value.
set_setting(key, value)	Set a specific setting.
get_db_version()	Get the package version that the databases state is at.
set_db_version([version])	Set the package version that the databases state is at.
set_is_broker_active(is_active)	Set the state of the broker.
get_is_broker_active()	Get the state of the broker.
check_is_broker_active()	Check if another broker instance is active and if so raise an error.

---

check_is_broker_active()

Check if another broker instance is active and if so raise an error.

Raises

RuntimeError – If the broker is not active.

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”].

get_db_version()

Get the package version that the databases state is at.

Returns

The version of the database.

Return type

version

`get_is_broker_active()`

Get the state of the broker.

Returns
Whether the broker is active.

Return type
bool

`get_setting(key)`

Get a specific settings value.

Parameters
key (*str*) – The key of the setting.

Returns
value – The version of the database.

Return type
str

`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.

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

Do the quality check on the stations raw data.

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.

`richter_correct()`

Richter correct all of the precipitation data.

`set_db_version`(*version=<Version('0.0.40')>*)

Set the package version that the databases state is at.

Parameters
`version` (*pv.Version, optional*) – The Version of the python package The default is the version of this package.

`set_is_broker_active`(*is_active: bool*)

Set the state of the broker.

Parameters
`is_active` (*bool*) – Whether the broker is active.

`set_setting`(*key: str*, *value: str*)

Set a specific setting.

Parameters

- **key** (*str*) – The key of the setting.
- **value** (*str*) – The value of the setting.

`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"].

vacuum(*do_analyze=True*)

4.1.2 Subpackages

4.1.2.1 lib package

4.1.2.1.1 utils

Some utilities functions and classes that are used in the module.

class weatherDB.lib.utils.**TimestampPeriod**(*start*, *end*, *tzinfo='UTC'*)

Bases: `object`

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

Initiate a TimestampPeriod.

Parameters

- **start** (*pd.Timestamp or similar*) – The start of the Period.
- **end** (*pd.Timestamp or similar*) – The end of the Period.
- **tzinfo** (*str or datetime.timezone object or None, optional*) – The timezone to set to the timestamps. If the timestamps already have a timezone they will get converted. If None, then the timezone is not changed or set. The default is "UTC".

contains(*other*)

Does this TimestampPeriod contain another TimestampPeriod?

Parameters

other (*Timestampperiod or tuple of 2 Timestamp or Timestamp strings*) – The other Timestamp to test against. Test if this TimestampPeriod contains the other.

Returns

True if this TimestampPeriod contains the other. Meaning that the start is smaller or equal than the others starts and the end is higher than the others end.

Return type

bool

copy()

Copy this TimestampPeriod.

Returns

a new TimestampPeriod object that is equal to this one.

Return type

TimestampPeriod

expand_to_timestamp()

get_interval()

Get the interval of the TimestampPeriod.

Returns

The interval of this TimestampPeriod. E.G. Timedelta(2 days 12:30:12)

Return type
pd.Timedelta

`get_middle()`

Get the middle Timestamp of the TimestampPeriod.

Returns
The middle Timestamp of this TimestampPeriod.

Return type
Timestamp

`get_period()`

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

Get the dictionary to use in sql queries.

Parameters
format (*str, optional*) – The Timestamp-format to use. The Default is "'%Y%m%d %H:%M'"

Returns
a dictionary with 2 keys (min_tstp, max_tstp) and the corresponding Timestamp as formated string.

Return type
dict

`has_NaT()`

Has the TimestampPeriod at least one NaT.

This means that the start or end is not given. Normally this should never happen, because it makes no sense.

Returns
True if the TimestampPeriod has at least on NaT. False if the TimestampPeriod has at least a start or a end.

Return type
bool

`has_only_NaT()`

Has the TimestampPeriod only NaT, meaning is empty.

This means that the start and end is not given.

Returns
True if the TimestampPeriod is empty. False if the TimestampPeriod has a start and an end.

Return type
bool

`inside(`*other*`)`

Is the TimestampPeriod inside another TimestampPeriod?

Parameters
other (*Timestampperiod or tuple of 2 Timestamp or Timestamp strings*) – The other Timestamp to test against. Test if this TimestampPeriod is inside the other.

Returns
True if this TimestampPeriod is inside the other. Meaning that the start is higher or equal than the others starts and the end is smaller than the others end.

Return type
bool

is_empty()

Is the TimestampPeriod empty.

This means that the start and end is not given.

Returns
True if the TimestampPeriod is empty. False if the TimestampPeriod has a start and an end.

Return type
bool

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

Convert the TimestampPeriod to a list of strings.

Formates the Timestamp as a string.

Parameters
format (*str, optional*) – The Timestamp-format to use. The Default is "%Y-%m-%d %H:%M:%S"

Returns
A list of the start and end of the TimestampPeriod as formated string.

Return type
list of 2 strings

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_cdc_file_list**(*ftp_folders*)

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

4.1.2.1.2 max_fun

4.1.2.1.2.1 import_DWD

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

`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 bigger than this value, the older "von_datum" is overwritten in the meta GeoDataFrame. The default is 2.

Returns

a GeoDataFrame of the meta file

Return type

geopandas.GeoDataFrame

CHAPTER

FIVE

CHANGE-LOG

5.1 Version 0.0.40

- change roger_toolbox format to keep minute and hour values even if daily aggregation

5.2 Version 0.0.39

- add the RoGeR Toolbox format as timeseries format. See https://github.com/Hydrology-IFH/roger for more specifications on the format
- only insert needed download time if DB_ENG is super user
- add possibility to change the column names and filenames of written out weather timeserires

5.3 Version 0.0.38

- fix problem when updating the Richter correction to only a period. Previously the Richter correction did only work, when applied to the whole period (period=(None,None)). When a smaller period was selected, everything outside of this period got set to NULL. This problem existed since Version 0.0.36
- update pattern to find meta file, DWD has a second file in kl daily folder, having "mn4" in name

5.4 Version 0.0.37

- create_ts: skip period check if already done in GroupStation or GroupStations -> previously this got checked 3 times
- add functionality to StationsBase.get_df to get multiple columns
- fix error in richter_correct from previous version

5.5 Version 0.0.36

- throw error if Richter correction is done on empty filled timeserie
- add test for filled daily values before adjusting the 10 minute values in the fillup
- fix errors in fillup for Temperature stations
- set autocommit for _drop method
- richter_correct: only update corr when new values -> way faster
- only give aggregated value if at least 80% data is available

5.6 Version 0.0.35

- set filled_by for T stations default to NULL not [NULL] -> works better with other methods
- change date parsing for read dwd function, to work with pandas version >2.0

5.7 Version 0.0.34

- StationsBase.get_meta: strip whitespace in str columns
- add min/max-tresholds for T and ET
- add -9999 as possible NA value for DWD data

5.8 Version 0.0.33

- change quality control of T- & ET-Stations -> add inversion consideration for stations above 800m altitude Those stations values are only sorted out in winter if their difference to the median neighbor station is negative (lower limit)
- change quality control of T and ET -> the values are now getting compared to the median of 5 neighbors, not the mean
- change fillup method: has now the possibility to take the median of multiple neighboring stations to fillup. This possibility is now used for Temperature stations, where 5 neighboring stations are considered.

5.9 Version 0.0.32

- add elevation consideration for the selection of neighboring stations, based on LARSIM formula for the quality_check and fillup procedure of T and ET. So not only the closest stations are selected but sometimes also a station that is further away, but has les difference in height.
- get neighboring stations for T and ET quality check for full years, to always have around 5 neighboring stations
- fix problem in get_multi_annual for T Station if no ma found
- fix error because timeseries did only get created when, station T or ET is in meta_n table, even if they exist in meta_t or meta_et. So e.g a T Station exists in meta table because of own data, but is not added because no P station is there.

5.10 Version 0.0.31

- only compare to neighboring stations if at least 2 stations have data in the quality check of T and ET
- add settings to the database and broker now updates the whole database if a new version is loaded
- stop broker execution if another broker instance is activly updating the database

5.11 Version 0.0.30

- fix MAJOR error in Temperature quality check: The coefficient did not get converted to the database unit. This had as a consequence, that the neighboring values did not get regionalised correctly to the checked station. So if the neighboring station has big difference in the multi annual temperature value, too many values got kicked out. This error existed probably since version 0.0.15

5.12 Version 0.0.29

-add calculation of dropped values in quality check

5.13 Version 0.0.28

- MAJOR Error fix: The quality check for T and ET did not consider the decimal multiplier for the limits. So the table 2 from the Method documentation should have looked like this until now, in bold are the numbers that were wrong in the code:

parameter	compare equation	lower limit	upper limit
Temperature	$\$\Delta T = T_{Stat} - \overline{T}_{neighbors}$	$\Delta T < -\mathbf{0.5}C$	$\Delta T > \mathbf{0.5}C\$$
pot. Evapotranspiration	$\delta ET = \frac{ET_{Stat}}{\overline{ET}_{neighbors}}$	$\begin{cases} \delta ET < \mathbf{20}\% \\ ET_{Stat} > \mathbf{0.2}\frac{mm}{d} \end{cases}$	$\begin{cases} \delta ET > 200\% \\ ET_{Stat} > \mathbf{0.3}\frac{mm}{d} \end{cases}$

Those limits got corrected to correspond now to:

parameter	compare equation	lower limit	upper limit
Temperature	$\Delta T = T_{Stat} - \overline{T}_{neighbors}$	$\Delta T < -\mathbf{5}C$	$\Delta T > \mathbf{5}C$
pot. Evapotranspiration	$\delta ET = \frac{ET_{Stat}}{\overline{ET}_{neighbors}}$	$\begin{cases} \delta ET < \mathbf{25}\% \\ ET_{Stat} > \mathbf{2}\frac{mm}{d} \end{cases}$	$\begin{cases} \delta ET > 200\% \\ ET_{Stat} > \mathbf{3}\frac{mm}{d} \end{cases}$

- fixed error that came up in version 0.0.27 for richter correction. The horizon was only calculated from west to south not from north to south.
- correct update_horizon to also consider that the distance between grid cells can be diagonal to the grid, so miultiply with $\sqrt{2}$

5.14 Version 0.0.27

- fixed major error with update_horizon method. Therefor the Richter Exposition classe changes for many stations. This error existed since Version 0.0.15
- add multiprocess ability to update_richter_class

5.15 Version 0.0.26

- fix error with sql statements
- fix logging

5.16 Version 0.0.25

version has major problems, use version 0.0.26

- change logging.py submodule name, because of import conflicts with python logging package

5.17 Version 0.0.24

- add text wrapper from sqlalchemy to work with sqlalchemy version >2.0
- add compatibility for shapely >2.0

5.18 Version 0.0.23

- change pandas to_csv parameter line_terminator to lineterminator, for newer versions
- change logging procedure, to not log to file as a standard way, but only after calling setup_file_logging from logging.py

5.19 Version 0.0.22

- add qc_from and qc_until to the meta informations
- fix removal of old log files

5.20 Version 0.0.21

- add additional parameter sql_add_where to define a sql where statement to filter the created results in the database
- add postgresql error messages that will cause the execution to wait and restart
- import Station(s)-classes imediatly when module is imported, so now this works

```
import weatherDB as wdb
wdb.StationsN()
```

5.21 Version 0.0.20

- change secretSettings_weatherDB names to DB_PWD, DB_NAME and DB_USER
- add min and max to the temperature timeseries

5.22 Version 0.0.19

- fix error of updating raw_files table after new import.
- change log file name to weatherDB_%host%_%user%.log
- change the use of append method to pandas concat method
- changed pandas method iteritems to items, due to deprecation warning

5.23 Version 0.0.18

- correct spelling error "methode" to "method"
- add progressbar to count_holes method
- add para to raw_files db-table, because some files get used for several parameters (T and N_D)

5.24 Version 0.0.17

- get_df now also accepts filled_share as kind
- added function to count the holes in the timeseries depending on there length

5.25 Version 0.0.16

- repaired the update_raw function of StationND
- change data source from REGNIE to HYRAS for precipitation regionalisation
- add ability to get nearby ma value from rasters, up to 1km from the station
- change day definition for precipitation to run from 5:50 to 5:50 as written in dwd cdc description. (previously it was 5:40 - 5:40, as 5:40 was the last value of the previous day)
- add ability to get all the meta information with get_meta
- save last_imp period but only for df without NAs -> else the marking of last_imp_qc… will not work, as the period will always be smaller than the last_imp period

5.26 Version 0.0.15

- change append with pandas concat function. -> faster
- don't import complete module on installation

5.27 Version 0.0.14

- added type test, if parameter gets checked for "all"
- specify that secrets_weatherDB file should be on PYTHONPATH environment variable
- Changed DGM5 to Copernicus DGM25, because of license advantages
- adjusted update_horizon method to be able to work with different CRS
- add kwargs to update_richter_class of StationsN
- fix get_geom with crs transforamation

5.28 Version 0.0.13

- change the timezone allocation method of the precipitation download df
- set freq to 10 minutes of precipitation download, to be able to overwrite Values with NAs
- add remove_nas parameter to overwrite new NAs in the database. (mainly for programming changes)
- define the name of the geometry column in get_meta.

5.29 Version 0.0.12

- add quality check for precipitation stations: delete values were the aggregated daily sum is more than double of the daily measurement
- when filling up also replace the filled_by column if it got changed
- TimestampPeriod class now also detects string inputs as date
- major error fixed: the coefficients calculation in the fillup method was the wrong way around
- for daily parameters the expand_timeseries_to_period ads now 23:50 to max_tstp_last_imp to get the period
- add vacuum cleanup method in Broker
- check precipitation df_raw for values below 0
- add stids parameter to last_imp methods of stations classes
- add an update method to stations classes, to do a complete update of the stations database data (update_raw + quality_check + fillup + richter_correct)
- only set start_tstp_last_imp values in db if update_raw is done for all the stations

5.30 Version 0.0.11

- add fallback on thread if multiprocessing is not working
- cleaning up ftplib use. Always recreate a new instance and don't try to reuse the instance. This resolves some problems with the threading of the instances.
- clean raw updates of only recent files by the maximum timestamp of the historical data.

5.31 Version 0.0.10

- fixed get_adj compare Timestamp with timezone

5.32 Version 0.0.9

- fixed future warning in stations.GroupStations().create_ts
- stations.GroupStations().create_roger_ts fixed
- removed join_how from _check_period as it was not used
- fixed StationND().get_adj, because the StationNBase.get_adj was only for 10 minute resolution
- get_adj always based on "filled" data

5.33 Version 0.0.8

- fixed installation (psycopg2 problem and DB_ENG creation)
- fixed importing module when not super user

5.34 Version 0.0.7

- convert timezone of downloaded precipitation data, because (before 200 the data is in “MEZ” afterwards in “UTC”)
- update_ma:
  - Rasters now also have proj4 code, if necessary. Because the postgis database is not supporting transformation to EPSG:31467
  - small speed improvement
- StationCanVirtual._check_meta updated to check separately if station is in meta table and if it has a timeseries table
- Added timezone support. The database timezone is UTC.

5.35 Version 0.0.6

- error fixed with is_virtual (!important error!)
- human readable format for the period in log message added
- some spelling errors fixed in documentation
- kwargs added to child methods of get_df (like get_raw…)
- in get_df and consecutive methods:
  - filled_share column added if aggregating and filled_by selected
  - possibility to download filled_by added
  - nas_allowed option added
  - add_na_share option added. (give the share of NAs if aggregating)
- in create_ts option to save several kinds added
- get_max_period method
- error in check_stids fixed
- error in ma_update fixed

5.36 Version 0.0.5

- The et_et0 parameter gor renamed to r_r0 in the create_ts method
- The r_r0 is now possible to add as pd.Serie or list, when creating a timeserie file
- get_meta method of single stations updated
- get_meta for GroupStation(s) updated
- get_df for GroupStation added
- Quickstart added to the documentation
- documentation has now a TOC tree per class and a method TOC tree on top
- option to skip the check if a station is in the meta file, this is used for computational advantages in the stations classes, because they test already before creating the objects if they are in the meta table.
- …_von and …_bis columns got renamed to the english name …_from and …_until
- the quot_… fields got all normed to % as unit
- dropping stations from meta while updating checks now if stid is in downloaded meta file

5.37 Version 0.0.4

- The method part was added to the documentation
- the connection method got updated

5.38 Version 0.0.3

This is the first released version

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX

S

T

U

V

W