Wetterturnier

Generated by Doxygen 1.6.1

Tue Aug 4 11:56:16 2015

Contents

1 Todo List

Member pywetterturnier::database::database::current_tournament Reto just take care of the idea that we cold start two tournaments in a row. Can this method then handle the requests?

Member pywetterturnier::database::database::get_bet_data Reto the deadman does ont get bets - he just gets points. Maybe I can disable/remove the 'all' function if I am not using it anymore.

Member pywetterturnier::database::database::get_cities Would be nice to return cityclass objects or something. However, needs some effort as I have to chnage a few lines of code.

2 Bug List

Member pywetterturnier::database::database::get_bet_data Reto you are including all bets, even if a user has only filled in a subset of parameters. Therefore I had the active' flag somewhen. Maybe would be better to croscheck the betstat table as the betstattable should only include full bets. Moreover betstat table is used to display the data on the front end and can easily be screened online.

Member pywetterturnier::utils::inputcheck Input argument what is not in use. Maybe kill it or at least set some defaults.

Member pywetterturnier::utils::usage change iputcheck, add propper usage.

3 Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

6 Namespace Documentation

6.1 pywetterturnier Namespace Reference

Namespaces

- namespace database
- namespace getobs
- namespace groupbets
- namespace importbets
- namespace judgingclass20021206
- namespace migrategroups
- namespace mitteltip
- namespace stationclass
- namespace utils

6.2 pywetterturnier::database Namespace Reference

Classes

- class database

  This is the main database handler class.

6.3 pywetterturnier::getobs Namespace Reference

Classes

- class getobs

6.4 pywetterturnier::groupbets Namespace Reference

6.5 pywetterturnier::importbets Namespace Reference

Classes

- class importbets

  USED TO IMPORT OLD ARCHIVE DATA FROM THE OLD WETTERTURNIER.

6.6 pywetterturnier::judgingclass20021206 Namespace Reference

Classes

- class judging

  This is a judgingclass - a class used to compute the points a user gets on a specific weekend.

6.7 pywetterturnier::migrategroups Namespace Reference

Classes

- class migrategroups

6.8 pywetterturnier::mitteltip Namespace Reference

Functions

- def mitteltip

6.8.1 Function Documentation

6.8.1.1 def pywetterturnier::mitteltip::mitteltip (*db*, *typ*, *ID*, *city*, *tdate*)

Function returning Mitteltips or group bets.

Parameters:

db. Database handler object, see database::database

typ. String. Forwarded to database::database::get_bet_data. Please read the manual there for more information.

ID. Integer. Forwarded to database::database::get_bet_data. Please read the manual there for more information.

city. Integer, city ID.

tdate. Integer, tournament date. Days since 1970-01-01'

Returns:

dict containing the mitteltip.

6.9 pywetterturnier::stationclass Namespace Reference

Classes

- class stationclass

  A small class holding all infors for a specific WMO station.

6.10 pywetterturnier::utils Namespace Reference

Functions

- def inputcheck
- def usage
- def readconfig
- def tdate2string
- def nicename
- def exit

6.10.1 Function Documentation

6.10.1.1 def pywetterturnier::utils::exit (*msg*)

Simple exit wrapper. Can be used to create kind of user-defined exit handling.

6.10.1.2 def pywetterturnier::utils::inputcheck (*what*)

Checking input arguments for several scripts. Using the same inputcheck routine for most of the Wetterturnier python backend scripts. The arguments are evaluated in here and added to a list object which will be returned. Please have a look into the usage (most scripts can be called with input argument -h/--help to display the usage).

Bug

Input argument what is not in use. Maybe kill it or at least set some defaults.

what. String, to handle some specal cases. At the moment the input argument is not used at all!

6.10.1.3 def pywetterturnier::utils::nicename (*string*)

Creates a nice username. Mainly used for migration where (from the text files of the old wetterturnier archive) a few usernames were including special characters, blanks, or other special things. The new Wetterturnier only allows propper non-special-character usernames. This function converts possibly impropper usernames to its propper equivalent.

string. Possibly impropper user name.

Returns:

string with nice username without special characters and shit.

6.10.1.4 def pywetterturnier::utils::readconfig (*file* = `'config.conf'`, *inputs* = `None`)

Reading config file. There is a 'global' wetterturnier backend config file which is necessary to handle all the actions. This method also checks some parameters. E.g., if a required directory or file does not exist, the script stops.

file. String, file name of the config file. Default is config.conf. inputs, dict. Usually the input dict from utils::inputcheck. Default is None. If it is adict: all parameters will be added to the config dict which will be generated in this method. In case a key exists in the config dict (created in here) and is duplicated in the inputs dict the script will stop immediately.

Returns:

dict containing all necessary configs.

6.10.1.5 def pywetterturnier::utils::tdate2string (*date*)

Converts tdate into string of form YYY-MM-DD. Note: a so called tdate is nothing else than an integer value indicating the days since 1970-01-01 which is used extensively in the wetterturnier (especially to optimize the databases).

date. Integer, days since 1970-01-01

Returns:

string of format Y-m-d

6.10.1.6 def pywetterturnier::utils::usage (*what* = `None`)

Script usage.

Bug

change iputcheck, add propper usage.

7 Class Documentation

7.1 pywetterturnier::database::database Class Reference

This is the main database handler class.

Public Member Functions

- def __init__
- def __connect__
- def cursor
- def execute
- def executemany
- def commit
- def create_group
- def create_user
- def get_cities
- def get_stations_for_city
- def current_tournament
- def all_tournament_dates
- def get_cityall_bet_data
- def get_bet_data
- def upsert_bet_data
- def upsert_points_data
- def upsert_deadman_points
- def get_obs_data
- def get_parameter_names
- def get_parameter_id
- def get_user_id_and_create_if_necessary
- def get_user_id
- def get_username_by_id
- def get_active_groups
- def get_group_id
- def create_groupuser
- def get_deadman_points
- def close

Public Attributes

- config
- db
- prefix

7.1.1 Detailed Description

This is the main database handler class. As soon as an object of this class is initialized python automatically (tries) to connect to the database. The database class offers a bunch of methods to get/write data into the databae. Not all commands are in this class, some SQL statements are defined within the other classes and methods.

7.1.2 Member Function Documentation

7.1.2.1 def pywetterturnier::database::database::__connect__ (*self*)

```
Open database connection.
In case python is not able to open the database connection
the script will stop immediately.
@return Returns a MySQLdb object (database handler).
```

7.1.2.2 def pywetterturnier::database::database::__init__ (*self*, *config*)

```
The database class is handling the connection to the
mysql database and different calls and stats.
@params config. Dict containing all necessary information like
mysql database, user, password, and database name. Please see
@ref utils.readconfig for more details about the config object.
```

7.1.2.3 def pywetterturnier::database::database::all_tournament_dates (*self*, *cityID*)

Returns all defined tournament dates ever payed in a city. Searches for all unique tournament dates in the bets table and returns them as a list.

Parameters:

cityID. Integer, ID of the city.

Returns:

A list wil be returned containing a set of integer values where each element represents one tournament played for the city. Dates in days since 1970-01-01.

7.1.2.4 def pywetterturnier::database::database::close (*self*, *verbose* = `True`)

Simple wrapper around MySQLdb.close.

7.1.2.5 def pywetterturnier::database::database::commit (*self*)

Simple wrapper to MySQL.commit.

7.1.2.6 def pywetterturnier::database::database::create_group (*self*, *name*, *desc*)

Checks and/or creates group. Checks the Wetterturnier database table **groups** and checks if the group is already existing. If existing, the job of this method is done. Else a new group will be created. Used for the migration of the groups in the old Wetterturnier archive.

Parameters:

name. String, group name.

desc. String, group description.

7.1.2.7 def pywetterturnier::database::database::create_groupuser (*self*, *user*, *group*, *since*, *active*)

Adds an existing user to an existing group. If the user is already an actie member of the group: don't add him/her again.

Parameters:

user. String, user name.

group. String, group name.

since. datetime object (normally current time) from when on the user was a member of the group.

active. Integer, if user is an actie member of the group or not. Typically 1 as you are adding the user in that split second.

7.1.2.8 def pywetterturnier::database::database::create_user (*self*, *name*, *password* = `None`)

Check and/or create new Wetterturnier user. Creates a new user. If user already exists, the job of this method is done. Return. If not, create new user.

As we are using wordpress we have to take care how we are creating users. To be on the save side I am using the wordpress internal function to add a user. Therefore a temporary php file will be created containing the necessary lines of code calling wp_create_user (in php) to add the user. The php file then will be called using the local php interpreter. After doing that the script is using database::get_user_id to check if the user was added proppperly. If not: stop.

Parameters:

name. String containing the username.

password,optional. If set to None the user wont get a password (actually i have to set a password, in this case the user password is just its user name backwards. E.g., Reto gets paswort oteR.

7.1.2.9 def pywetterturnier::database::database::current_tournament (*self*)

Returns tdate for current tournament. The tdate is the number of days since 1970-01-01. Loading the max(tournamentdate) from the bets table. If all works as it should bets can only be placed for the upcoming tournament. And this method is used to compute e.g., the points and stuff. Therefore we just have to know which are the newest bets and loop over them.

Returns:

Integer date (days since 1970-01-01)

Todo

Reto just take care of the idea that we cold start two tournaments in a row. Can this method then handle the requests?

7.1.2.10 def pywetterturnier::database::database::cursor (*self*)

Simple wrapper to MySQL.cursor.

7.1.2.11 def pywetterturnier::database::database::execute (*self*, *sql*)

Simple wrapper to MySQL.execute.

7.1.2.12 def pywetterturnier::database::database::executemany (*self*, *sql*, *data*)

Simple wrapper to MySQL.executemany.

7.1.2.13 def pywetterturnier::database::database::get_active_groups (*self*)

Returns all active group names from database.

Returns:

list containing string group names of all active groups.

7.1.2.14 def pywetterturnier::database::database::get_bet_data (*self*, *typ*, *ID*, *cityID*, *paramID*, *tdate*, *bdate*)

Returns bet data used to compute e.g., Petrus. Returns a set of bets for a given city, parameter, and bet date for a specified tournament date. Note: has different modes.

- If typ == 'all': all users (human forecasters, automated forecasters and group bets) **EXCLUDING** the Deadman will be returned.
- If type == 'user' the bet of a specific user will be returned.
- If type == 'group' the bet of a specific group will be returned. User or group (if type == 'user' or 'group') are defined by the input argument ID which is the userID or the groupID.

Parameters:

typ. String, one of type 'all', 'user', or 'group'.

ID. Ignored when type = 'all'. Else the input has to be of type ineger defining the userID (for type = 'user') or groupID (for type = 'gruop').

cityID. Integer, ID of the city.

paramID. Integer, parameter ID.

tdate. Integer, tournament date (if tournament starts on Friday, this is the date of this friday). Days since 1970-01-01.

betdate,: Integer. If value is lower equal 5: assume that the real betdate is tdate + betdate (1: next day, 2: two days ahead, ...). If betdate is bigger than 5 betdate is just taken as set.

Returns:

Returns a list containing all the bets.

Bug

Reto you are including all bets, even if a user has only filled in a subset of parameters. Therefore I had the active' flag somewhen. Maybe would be better to croscheck the betstat table as the betstattable should only include full bets. Moreover betstat table is used to display the data on the front end and can easily be screened online.

Todo

Reto the deadman does ont get bets - he just gets points. Maybe I can disable/remove the 'all' function if I am not using it anymore.

7.1.2.15 def pywetterturnier::database::database::get_cities (*self*)

Loading city information from the database. Loads all active cities from the database which can then be used to loop over or whatever you'll do with it.

Returns:

list. A list containing one dict per city where each dict consists of the keys 'name' and 'ID'.

Todo

Would be nice to return cityclass objects or something. However, needs some effort as I have to chnage a few lines of code.

7.1.2.16 def pywetterturnier::database::database::get_cityall_bet_data (*self*, *cityID*, *paramID*, *tdate*, *day*, *nodeadman* = `True`, *nullonly* = `False`)

Returns bets for a given city/parameter/date/day. Returns all bets for a given city, parameter for a given bet day of a specified tournament. Note that there is no userID. This method returns the bets for all users.

Parameters:

cityID. Integer, ID of the city.

paramID. Integer, parameter ID.

tdate. Integer, tournament date (if tournament starts on Friday, this is the date of this friday). Days since 1970-01-01.

day,: Integer. [1/2] where 1 means tdate+1 (leading to a Saturday if tdate is a Friday). 2 means Sunday. Value between 0 and 5, however, only 1 and 2 are useful as we do not have bets for the other days.

Returns:

Returns two lists, the first one is a list containing the database ID's from the bets table, the second one the values (bets itself).

7.1.2.17 def pywetterturnier::database::database::get_deadman_points (*self*, *cityID*, *tdate*, *ignore*)

Loading the points from the Deadman user. Returns tuple including betdate, parameterID, deadman points for a given city/tournament date. Points are computed as follows:

- Deadman Points = Average(Points) - Standarddeviation(Points) ... where 'Points' are all Group/User points gained in the tournament **EXCLUDING** the Deadman. If the Deadman would not be excluded the points would drift towards -Inf as the average decreases iteratively and the standard deviation increases iteratively.

  Returns:

  a tuple tuple as fetched from database. Typically including two tuples, first for Saturday, second for Sunday.

7.1.2.18 def pywetterturnier::database::database::get_group_id (*self*, *group*)

Returns group ID given a group name.

Parameters:

group. String, group name.

Returns:

False if the group cannot be found, else the integer group ID

7.1.2.19 def pywetterturnier::database::database::get_obs_data (*self*, *cityID*, *paramID*, *tdate*, *bdate*, *wmo* = `None`)

Loading observation data from the obs database (the obs which are already in the format as they are used for the judging).

- if input wmo == None: return all obs for all stations for a given city/parameter/tdate/bdate.
- if input wmo is an integer value, only the observation for this specific station will be returned.

Parameters:

userID. Integer, user ID of the **Deadman** user.

cityID. Integer, city ID.

paramID. Integer, parameter ID.

tdate. Integer, tournament date. Days since 1970-01-01.

betdate. Integer, bet (forecast) date. Days since 1970-01-01.

wmo. Integer, optional. Either None (which is default) or WMO station number.

Returns:

: Returns either a list containing numeric values (if wmo == None) or a single numeric value. If there are no data at all, a boolean False will be returned.

7.1.2.20 def pywetterturnier::database::database::get_parameter_id (*self*, *param*)

Returns parameter ID given a parameter Name.

Parameters:

param. String, parameter short name (e.g., TTm)

Returns:

False if the parameter cannot be found in the database or the corresponding integer parameter ID.

7.1.2.21 def pywetterturnier::database::database::get_parameter_names (*self*, *active* = `False`)

Returns all parameter names. If input active is set, only active parametres will be returned.

Parameters:

active. Boolean, optional. Default is False.

Returns:

False if no parameters can be found. Else a list will be returned containing the parameter shortnames as strings.

7.1.2.22 def pywetterturnier::database::database::get_stations_for_city (*self*, *cityID*)

Loading all stations mached to a certain city.

Parameters:

cityID. Integer, ID of the city in the database.

Returns:

List object containing **N** stationclass::stationclass objects.

7.1.2.23 def pywetterturnier::database::database::get_user_id (*self*, *user*)

Returns user ID given a username. If the user cannot be found, the method returns False. There is a vice versa function called database::get_username_by_id .

Parameters:

user. String, user name.

Returns:

False if user does not exist, else the integer user ID.

7.1.2.24 def pywetterturnier::database::database::get_user_id_and_create_if_necessary (*self*, *name*)

Returns user ID and create user if necessary. All-in-one wonder method. Searching for the user ID of a given user. If the user does not exist: create the user first and then return the user ID.

See also:

database.create_user
database.get_user_id

Parameters:

name. String, user name.

Returns:

Integer value with userID.

7.1.2.25 def pywetterturnier::database::database::get_username_by_id (*self*, *userID*)

Returns username given a user ID. Returns the username for a given user ID. If the user cannot be found, the return value will be False. There is a vice versa function called database::get_user_id.

Parameters:

userID. Integer, user ID.

Returns:

False if user cannot be identified, else string user name.

7.1.2.26 def pywetterturnier::database::database::upsert_bet_data (*self*, *userID*, *cityID*, *paramID*, *tdate*, *bdate*, *value*)

Helper function to update the bets database. Upserts the bets database setting a new 'value' for a given player.

Parameters:

userID. Integer, user ID.

cityID. Integer, city ID.

paramID. Integer, parameter ID.

tdate. Integer, tournament date. Days since 1970-01-01.

betdate. Integer, bet (forecast) date. Days since 1970-01-01.

value. Integer bet/forecast value. Note that the value has to be scaled already as we store e.g. temperature in 1/10th of degrees celsius.

7.1.2.27 def pywetterturnier::database::database::upsert_deadman_points (*self*, *userID*, *cityID*, *tdate*, *points*)

Helper function to update the points for the Deadman player. Upserts the betstat database updating the points for the Deadman. Actually for any player. But the userID should be the one from the Deadman here. The Deadman just gets points for the full weekend, not points for specific parameters.

Parameters:

userID. Integer, user ID of the **Deadman** user.

cityID. Integer, city ID.

paramID. Integer, parameter ID.

tdate. Integer, tournament date. Days since 1970-01-01.

betdate. Integer, bet (forecast) date. Days since 1970-01-01.

points. Numeric, points the player got for that specific entry.

7.1.2.28 def pywetterturnier::database::database::upsert_points_data (*self*, *userID*, *cityID*, *paramID*, *tdate*, *bdate*, *points*)

Helper function to update the bets database. Upserts the bets database updating the points for a given player. This is for the payers. They get points for each of the parameters.

Parameters:

userID. Integer, user ID.

cityID. Integer, city ID.

paramID. Integer, parameter ID.

tdate. Integer, tournament date. Days since 1970-01-01.

betdate. Integer, bet (forecast) date. Days since 1970-01-01.

points. Numeric, points the player got for that specific entry.

7.1.3 Member Data Documentation

7.1.3.1 pywetterturnier::database::database::config

7.1.3.2 pywetterturnier::database::database::db

7.1.3.3 pywetterturnier::database::database::prefix

The documentation for this class was generated from the following file:

- pywetterturnier/database.py

7.2 pywetterturnier::getobs::getobs Class Reference

Public Member Functions

- def __init__
- def get_maximum_Sd
- def get_columns

- def load_obs
- def check_record
- def prepare
- def show
- def write_to_db

Public Attributes

- db
- data
- stations

Private Member Functions

- def _add_obs_value_
- def _prepare_fun_TTm_
- def _prepare_fun_TTn_
- def _prepare_fun_TTd_
- def _prepare_fun_PPP_
- def _prepare_fun_dd_
- def _prepare_fun_ff_
- def _prepare_fun_fx_
- def _prepare_fun_N_
- def _prepare_fun_Wv_
- def _prepare_fun_Wn_
- def _prepare_fun_RR_
- def _prepare_fun_Sd_

Private Attributes

- _config_
- _table_
- _city_
- _date_
- _columns_
- _maxSd_

7.2.1 Member Function Documentation

7.2.1.1 def pywetterturnier::getobs::getobs::__init__ (*self*, *config*, *db*, *city*, *date*)

Initialization of the getobs::getobs class.

Parameters:

config. List, contains all necessary configs for the pywetterturnier package. Please have a look into utils::readconfig for more details.

Returns:

Returns nothing. Just contains several methods.

7.2.1.2 def pywetterturnier::getobs::getobs::_add_obs_value_ (*self*, *parameter*, *wmo*, *value*) `[private]`

Adds a loaded observation to the final data object stored on the parent getobs::getobs object. Appends all data to the public attribute getobs.getobs.data .

Parameters:

parameter. String, wetterturnier parameter shortname.

wmo. Integer, WMO station number.

value. Either a numeric value or None.

7.2.1.3 def pywetterturnier::getobs::getobs::_prepare_fun_dd_ (*self*, *station*) `[private]`

Helper function for the wind direction at 12 UTC from database column dd. Values will be returned in 1/10 degrees but rounded to full 10 degrees. E.g., observed '138' degrees will be converted into '1400' (1/10 degrees, rounded to full 10 degrees). Special case: also depends on database column ff. The following cases will be used:

- 1) if dd not observed/received: **return** None
- 2) if dd==0 and ff==0: **return** 0.0
- 3) if dd==0 and ff>0: **return** None (variable wind direction)
- 4) else: **return** value

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.4 def pywetterturnier::getobs::getobs::_prepare_fun_ff_ (*self*, *station*) `[private]`

Helper function for the wind speed at 12 UTC. Based on database column ff. Values will be in 1/10 knots but rounded to full knots. E.g., if 3.2m/s observed -> 6.22kt -> Return value will be 60. Args: parameter (string): string with parameter short name (e.g., TTm, N, RR) station (class): stationclass object. Returns: numeric: observed value if loading data was successful None: if observation not available or nor recorded

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.5 def pywetterturnier::getobs::getobs::_prepare_fun_fx_ (*self*, *station*) `[private]`

Helper function for the maximum gust speed (fx > 25kt). Based on database column fx1. Return value will be in 1/10 knots but rounded to full knots. E.g., if 21.2m/s observed -> 41.21kt -> Return value will be 410. Special cases:

- 1) no observation available but +30h observation (row) is in database: Assume that there were no gusts at all: **return** 0
- 2) no observations available and +30h observation (row) not yet in database: **return** None
- 3) observation available, below 25 knots: **return** 0
- 4) observation available, >= 25 knots: **return** value

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.6 def pywetterturnier::getobs::getobs::_prepare_fun_N_ (*self*, *station*) `[private]`

Helper function for cloud cover at 12 UTC. Return value will be in 1/10 octas, rounded to full octas [0,10,20,30,...,80]. Observations based on database column cc.

- 1) if observation is available: **return** value
- 2) if observation not recorded but 12 UTC database entry exists we assume that there were no clouds: **return** 0
- 3) else: **return** None

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.7 def pywetterturnier::getobs::getobs::_prepare_fun_PPP_ (*self*, *station*) `[private]`

Helper function for mean sea level pressure at 12 UTC. Based on database column pmsl. Return value will be in 1/10 hPa.

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.8 def pywetterturnier::getobs::getobs::_prepare_fun_RR_ (*self*, *station*) `[private]`

Helper function for 24h sum of precipitation based on database column 'rrr12' at +18 and +30h (as the reported observations are 12h sums this means from 06 UTC today to 06 UTC tomorrow). Returns precipitation in 1/10 mm OR -30 if there was no precipitation at all.

- First: if database entry for 18 UTC is here but there is no recorded amount of precipitation we have to assume that there was no precipitation. The same for +30h (06 UTC next day).
- Second: If observed precipitation amount is negative (some stations send -0.1mm/12h for no precipitation) we

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.9 def pywetterturnier::getobs::getobs::_prepare_fun_Sd_ (*self*, *station*) `[private]`

Helper function for relative sun shine duration for the full day. Will be returned in 1/10 percent rounded to full percent (34% will result in 340.).

- 1) if 24h sum is available on 'sunday' on 06 UTC we will take this value.
- 2) if 06UTC sunday is not available but we have a sunday value at 00 UTC: take this one.
- 3) else sum up the 'sun' 1h-rly obs. WARNING: seems that 'none' is either 'no sun' or 'not reported'. I can't decide which one is which one at the moment. I just take none = 0 and sum up.

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.10 def pywetterturnier::getobs::getobs::_prepare_fun_TTd_ (*self*, *station*) `[private]`

Helper function for dew point temperature. Returns 12 UTC observed dew point temperature from database column td in 1/10 degrees Celsius.

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.11 def pywetterturnier::getobs::getobs::_prepare_fun_TTm_ (*self*, *station*) `[private]`

Helper function for TTM, maximum temperature. Returns 18 UTC maximum temperature, either from column tmax12 or - if tmax12 not available but tmax24 exists - maximum temperature from tmax24. Temperature will be in 1/10 degrees Celsius.

Parameters:

station. Object of class stationclass::stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.12 def pywetterturnier::getobs::getobs::_prepare_fun_TTn_ (*self*, *station*) `[private]`

Helper function for TTN, minimum temperature. Returns 06 UTC minimum temperature. Simply the tmin12 column at 06 UTC in 1/10 degrees Celsius.

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.13 def pywetterturnier::getobs::getobs::_prepare_fun_Wn_ (*self*, *station*) `[private]`

Helper function for significant weather observatioins between 12 UTC and 18 UTC (afternoon). Based on database table w1. Value will be in 1/10 levels [0,10,20,...,90].

- 1) if observation not recorded but 18 UTC database entry exists we assume that there were no clouds: **return** 0
- 2) if observation not recorded and 18 UTC database entry not available: **return** None
- 3) observation here BUT the observed value is >= 10 (note: BUFR messages, 10+ are for automated significant weather instruments): **return** None
- 3) else: **return** value

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.14 def pywetterturnier::getobs::getobs::_prepare_fun_Wv_ (*self*, *station*) `[private]`

Helper function for significant weather observatioins between 06 UTC and 12 UTC (forenoon) based on database table w1. Value will be in 1/10 levels [0,10,20,...,90]. 1) if observation not recorded but 12 UTC database entry exists we assume that there were no clouds return 0 2) if observation not recorded and 12 UTC database entry not available return None 3) observation here BUT the observed value is $<$ 10 (note: BUFR messages, 10+ are for automated significant weather instruments) return None 3) else return value

Parameters:

station. Object of class stationclass.

Returns:

numeric or None. Returns observed value if loading data was successful, or None if observation not available or nor recorded

7.2.1.15 def pywetterturnier::getobs::getobs::check_record (*self*, *wmo*, *hour*)

Checks if database record exists. Just checks if for the current date/time/station combination a row exists in the database and not what a specific cell contains. Note: date, and table name are coming from the public attributes of getobs::getobs .

Parameters:

wmo. Integer, WMO station number.

hour. Integer, hour time shift relative to the 'date', 00:00 UTC. As an example: hour=30 means that we are checking 'tomorrow 06:00 UTC'.

Returns:

Boolean value. False if no such row exists, else True.

7.2.1.16 def pywetterturnier::getobs::getobs::get_columns (*self*, *table*)

Returns database table columns.

Parameters:

> ***table.*** String, name of the database table.

Returns:

> A list of all available table columns.

7.2.1.17 def pywetterturnier::getobs::getobs::get_maximum_Sd (*self*, *stations*, *date*)

Computes astronomic (maximum) sun shine duration for a set of stations. Note that the station has to be stored in the database table **obs.stations**. If not, we dont know the position of the station and therefore we can't compute astronomical sunshine duration resulting in a None value. Uses external python package .

Parameters:

> ***stations.*** List of stationclass::stationclass objects.
>
> ***datetime*** object. Day of the observations.

Returns:

> A dict consisting of WMO station number and the length of the astronomic day in seconds.

7.2.1.18 def pywetterturnier::getobs::getobs::load_obs (*self*, *wmo*, *hour*, *parameter*)

Loading a specific observation from the database. The date for which the observation should be valid and the name of the database are coming from the public attributes of the class (getobs::getobs).

Parameters:

> ***wmo.*** Numeric, WMO station number.
>
> ***hour.*** Integer, hour for which the observation should be valid [0,...,24].

Returns:

> Either a numeric value or None if the cell in the database was empty (NULL).

7.2.1.19 def pywetterturnier::getobs::getobs::prepare (*self*, *parameter*)

Prepares the different observed parameters like TTn, TTm, N, and so on. Note that the script will ignore a wrong specified or unknown parameter as the internal method then does not exist. The date will be taken from the public arguments of the parent class getobs::getobs.

Parameters:

parameter. String, shortname of the Wetterturnier parameters.

Returns:

No return. Appends data to internal object.

7.2.1.20 def pywetterturnier::getobs::getobs::show (*self*)

Shows a summary of the current getobs::getobs class. Was mainly used for development purposes. Shows data and stuff stored on class attributes.

7.2.1.21 def pywetterturnier::getobs::getobs::write_to_db (*self*)

Write prepared observations to database. Writes all prepared observations which are stored in the parent getobs::getobs class to the Wetterturnier database.

7.2.2 Member Data Documentation

7.2.2.1 pywetterturnier::getobs::getobs::_city_ `[private]`

City ID for which we are loading the observations.

7.2.2.2 pywetterturnier::getobs::getobs::_columns_ `[private]`

Columns available in in self._table_.

7.2.2.3 pywetterturnier::getobs::getobs::_config_ `[private]`

Copy of the input config list.

7.2.2.4 pywetterturnier::getobs::getobs::_date_ `[private]`

Date of the observations.

7.2.2.5 pywetterturnier::getobs::getobs::_maxSd_ `[private]`

Dict with astronomic day length per station.

7.2.2.6 pywetterturnier::getobs::getobs::_table_ [private]

Database table name.

7.2.2.7 pywetterturnier::getobs::getobs::data

Will contain data. Intitialized as None, will be a 2-level deep dict at the end of the form: data[wmo station number][parameter] = observation value.

7.2.2.8 pywetterturnier::getobs::getobs::db

database::database object handling the db I/O.

7.2.2.9 pywetterturnier::getobs::getobs::stations

List of stationclass objects.

The documentation for this class was generated from the following file:

- pywetterturnier/getobs.py

7.3 pywetterturnier::importbets::importbets Class Reference

USED TO IMPORT OLD ARCHIVE DATA FROM THE OLD WETTERTURNIER.

Public Member Functions

- def __init__
- def loadfile
- def loadfilecontent
- def takedata
- def identify_city
- def extract_bettimes
- def extract_parameter_points
- def extract_sum_points
- def extract_obs
- def __insert_obs_to_db__
- def __get_wmo_number__
- def extract_bets
- def __insert_bet_to_db__
- def __insert_betstat_to_db__
- def wp_create_user
- def wp_user_exists
- def wp_get_param_id
- def close

Public Attributes

- config
- db
- raw_lines
- tournamentdate
- obs1
- obs2
- points
- bettimes

Static Public Attributes

- data1 = None
- data2 = None
- cityID = None
- string prefix = 'wp_'
- string db_users = 'users'
- string db_param = 'wetterturnier_param'
- string db_bets = 'wetterturnier_bets'
- string db_betstat = 'wetterturnier_betstat'
- string db_cities = 'wetterturnier_cities'
- string db_obs = 'wetterturnier_obs'
- unames = None

7.3.1 Detailed Description

USED TO IMPORT OLD ARCHIVE DATA FROM THE OLD WETTERTURNIER. AS I WILL NEVER USE THIS ROUTINE AGAIN I DONT DO THE DOCUMENTATION FOR ALL THE METHODS IN HERE.

7.3.2 Member Function Documentation

7.3.2.1 def pywetterturnier::importbets::importbets::__get_wmo_number__ (*self*, *string*)

7.3.2.2 def pywetterturnier::importbets::importbets::__init__ (*self*, *config*)

7.3.2.3 def pywetterturnier::importbets::importbets::__insert_bet_to_db__ (*self*, *userID*, *paramID*, *tdate*, *bdate*, *value*)

7.3.2.4 def pywetterturnier::importbets::importbets::__insert_betstat_to_db__ (*self*, *userID*, *tdate*)

7.3.2.5 def pywetterturnier::importbets::importbets::__insert_obs_to_db__ (*self*, *station*, *paramID*, *bdate*, *value*)

7.3.2.6 def pywetterturnier::importbets::importbets::close (*self*)

7.3.2.7 def pywetterturnier::importbets::importbets::extract_bets (*self*, *block*)

7.3.2.8 def pywetterturnier::importbets::importbets::extract_bettimes (*self*)

7.3.2.9 def pywetterturnier::importbets::importbets::extract_obs (*self*, *block*)

7.3.2.10 def pywetterturnier::importbets::importbets::extract_parameter_points (*self*)

7.3.2.11 def pywetterturnier::importbets::importbets::extract_sum_points (*self*)

7.3.2.12 def pywetterturnier::importbets::importbets::identify_city (*self*)

7.3.2.13 def pywetterturnier::importbets::importbets::loadfile (*self*, *uri*, *file* = `False`)

7.3.2.14 def pywetterturnier::importbets::importbets::loadfilecontent (*self*, *file*)

7.3.2.15 def pywetterturnier::importbets::importbets::takedata (*self*, *lines*, *file* = `False`)

7.3.2.16 def pywetterturnier::importbets::importbets::wp_create_user (*self*, *username*)

7.3.2.17 def pywetterturnier::importbets::importbets::wp_get_param_id (*self*, *paramname*)

7.3.2.18 def pywetterturnier::importbets::importbets::wp_user_exists (*self*, *username*)

7.3.3 Member Data Documentation

7.3.3.1 pywetterturnier::importbets::importbets::bettimes

7.3.3.2 pywetterturnier::importbets::importbets::cityID = None `[static]`

7.3.3.3 pywetterturnier::importbets::importbets::config

7.3.3.4 pywetterturnier::importbets::importbets::data1 = None `[static]`

7.3.3.5 pywetterturnier::importbets::importbets::data2 = None `[static]`

7.3.3.6 pywetterturnier::importbets::importbets::db

7.3.3.7 string pywetterturnier::importbets::importbets::db_bets = 'wetterturnier_bets' `[static]`

7.3.3.8 string pywetterturnier::importbets::importbets::db_betstat = ’wetterturnier_betstat’ `[static]`

7.3.3.9 string pywetterturnier::importbets::importbets::db_cities = ’wetterturnier_cities’ `[static]`

7.3.3.10 string pywetterturnier::importbets::importbets::db_obs = ’wetterturnier_obs’ `[static]`

7.3.3.11 string pywetterturnier::importbets::importbets::db_param = ’wetterturnier_param’ `[static]`

7.3.3.12 string pywetterturnier::importbets::importbets::db_users = ’users’ `[static]`

7.3.3.13 pywetterturnier::importbets::importbets::obs1

7.3.3.14 pywetterturnier::importbets::importbets::obs2

7.3.3.15 pywetterturnier::importbets::importbets::points

7.3.3.16 string pywetterturnier::importbets::importbets::prefix = ’wp_’ `[static]`

7.3.3.17 pywetterturnier::importbets::importbets::raw_lines

7.3.3.18 pywetterturnier::importbets::importbets::tournamentdate

7.3.3.19 pywetterturnier::importbets::importbets::unames = None `[static]`

The documentation for this class was generated from the following file:

- pywetterturnier/importbets.py

7.4 pywetterturnier::judgingclass20021206::judging Class Reference

This is a judgingclass - a class used to compute the points a user gets on a specific weekend.

Public Member Functions

- def __init__
- def points_to_database
- def get_points
- def __residuals__
- def __points_TTm__
- def __points_TTn__
- def __points_TTd__
- def __points_TTmTTnTTd__
- def __points_N__
- def __points_Sd__
- def __points_dd__
- def __points_ff__
- def __points_fx__
- def __points_Wv__
- def __points_Wn__
- def __points_WvWn__
- def __points_PPP__
- def __points_RR__

Public Attributes

- quiet
- tdate_min

Static Public Attributes

- int tdate_min = 12027
- tdate_max = None

Private Member Functions

- def _prepare_for_database_

7.4.1 Detailed Description

This is a judgingclass - a class used to compute the points a user gets on a specific weekend. Please note that it is possible that the rules change somewhen and that there is a second judgingclass.

The class contains public attributes tdate_min and tdate_max as a safety-instrument. As soon as you would like to compute points for a specific tournament which falls outside this limits the script will stop in the operational mode not to re-compute old bets with a wrong judgingclass.

7.4.2 Member Function Documentation

7.4.2.1 def pywetterturnier::judgingclass20021206::judging::__init__ (*self*, *quiet* = `False`)

Initialize new juding object.

Parameters:

quiet. Boolean, default is False. Redudes the output to a minimum if set to True. Used for some applications.

Note:

actually I would like to connect this script to a php scrpt to give the users the oportunity to 'test' the judgingclass. And therefore the script has to be quiet and just prints what I need. Not done yet.

7.4.2.2 def pywetterturnier::judgingclass20021206::judging::__points_dd__ (*self*, *obs*, *data*, *special*)

7.4.2.3 def pywetterturnier::judgingclass20021206::judging::__points_ff__ (*self*, *obs*, *data*, *special*)

7.4.2.4 def pywetterturnier::judgingclass20021206::judging::__points_fx__ (*self*, *obs*, *data*, *special*)

7.4.2.5 def pywetterturnier::judgingclass20021206::judging::__points_N__ (*self*, *obs*, *data*, *special*)

7.4.2.6 def pywetterturnier::judgingclass20021206::judging::__points_PPP__ (*self*, *obs*, *data*, *special*)

7.4.2.7 def pywetterturnier::judgingclass20021206::judging::__points_RR__ (*self*, *obs*, *data*, *special*)

7.4.2.8 def pywetterturnier::judgingclass20021206::judging::__points_Sd__ (*self*, *obs*, *data*, *special*)

7.4.2.9 def pywetterturnier::judgingclass20021206::judging::__points_TTd__ (*self*, *obs*, *data*, *special*)

7.4.2.10 def pywetterturnier::judgingclass20021206::judging::__points_TTm__ (*self*, *obs*, *data*, *special*)

7.4.2.11 def pywetterturnier::judgingclass20021206::judging::__points_TTmTTnTTd__ (*self*, *obs*, *data*, *special*, *factor*)

7.4.2.12 def pywetterturnier::judgingclass20021206::judging::__points_TTn__ (*self*, *obs*, *data*, *special*)

7.4.2.13 def pywetterturnier::judgingclass20021206::judging::__points_Wn__ (*self*, *obs*, *data*, *special*)

7.4.2.14 def pywetterturnier::judgingclass20021206::judging::__points_Wv__ (*self*, *obs*, *data*, *special*)

7.4.2.15 def pywetterturnier::judgingclass20021206::judging::__points_WvWn__ (*self*, *obs*, *data*, *special*)

7.4.2.16 def pywetterturnier::judgingclass20021206::judging::__residuals__ (*self*, *obs*, *data*)

7.4.2.17 def pywetterturnier::judgingclass20021206::judging::_prepare_for_database_ (*self*, *IDs*, *values*) `[private]`

Prepares ID/value pairs for database. Creates a tuple list which can be used with **database.database.executemany**. Used later with the method 'points_to_database' within this class to update the bets table.

Parameters:

IDs. List containing integers, ID's of the rows in the database we have to update.

values. List containing numeric values connected to the ID's.

7.4.2.18 def pywetterturnier::judgingclass20021206::judging::get_points (*self*, *obs*, *what*, *data*, *special* = `None`, *tdate* = `None`)

7.4.2.19 def pywetterturnier::judgingclass20021206::judging::points_to_database (*self*, *db*, *IDs*, *values*)

Updates the bets database.

Parameters:

IDs. List containing integers, ID's of the rows in the database we have to update.

values. List containing numeric values connected to the ID's.

7.4.3 Member Data Documentation

7.4.3.1 pywetterturnier::judgingclass20021206::judging::quiet

7.4.3.2 pywetterturnier::judgingclass20021206::judging::tdate_max = None `[static]`

If set - do not use this judgingclass in the operational mode for tournaments > this date (days sicne 1970-01-01).

7.4.3.3 pywetterturnier::judgingclass20021206::judging::tdate_min

7.4.3.4 int pywetterturnier::judgingclass20021206::judging::tdate_min = 12027 `[static]`

Do not use this judgingclass in the operational mode for tournaments smaller or equal to this date (days since 1970-01-01).

The documentation for this class was generated from the following file:

- pywetterturnier/judgingclass20021206.py

7.5 pywetterturnier::migrategroups::migrategroups Class Reference

Public Member Functions

- def __init__
- def __download_file__
- def __read_rawfile__
- def create_groups_and_users

Public Attributes

- config
- db
- rawfile
- data

7.5.1 Member Function Documentation

7.5.1.1 def pywetterturnier::migrategroups::migrategroups::__download_file__ (*self*)

```
Downloading the html file and store locally
```

7.5.1.2 def pywetterturnier::migrategroups::migrategroups::__init__ (*self*, *config*)

```
Downloading the groups list file from wetterturnier.de,
Parse the content and create groups and its users if they
are not allready existing.
```

7.5.1.3 def pywetterturnier::migrategroups::migrategroups::__read_rawfile__ (*self*)

```
Parsing raw file and import groups and users.
```

7.5.1.4 def pywetterturnier::migrategroups::migrategroups::create_groups_and_users (*self*)

7.5.2 Member Data Documentation

7.5.2.1 pywetterturnier::migrategroups::migrategroups::config

7.5.2.2 pywetterturnier::migrategroups::migrategroups::data

7.5.2.3 pywetterturnier::migrategroups::migrategroups::db

7.5.2.4 pywetterturnier::migrategroups::migrategroups::rawfile

The documentation for this class was generated from the following file:

- pywetterturnier/migrategroups.py

7.6 pywetterturnier::stationclass::stationclass Class Reference

A small class holding all infors for a specific WMO station.

Public Member Functions

- def __init__
- def show

Public Attributes

- ID
- wmo
- cityID
- name
- nullconfig
- changed

7.6.1 Detailed Description

A small class holding all infors for a specific WMO station.

7.6.2 Member Function Documentation

7.6.2.1 def pywetterturnier::stationclass::stationclass::__init__ (*self*, *desc*, *data*)

Initializing a new stationclss object.

Parameters:

- ***desc.*** List/tuple of strings, value description of the values in input data.
- ***data.*** List/tuple of values corresponding to input desc.

7.6.2.2 def pywetterturnier::stationclass::stationclass::show (*self*)

Small summary function which prints the content of a stationclass object in a nice way.

7.6.3 Member Data Documentation

7.6.3.1 pywetterturnier::stationclass::stationclass::changed

Date the station was changed the last time.

7.6.3.2 pywetterturnier::stationclass::stationclass::cityID

To which city the station maches, city ID from database.

7.6.3.3 pywetterturnier::stationclass::stationclass::ID

Station ID in the database.

7.6.3.4 pywetterturnier::stationclass::stationclass::name

String, name of the station.

7.6.3.5 pywetterturnier::stationclass::stationclass::nullconfig

The nullconfig is a list of ID's where each ID corresponds to a parameter in the database (parameterID). All parameters included in the nullconfig are labeled as 'this station does NOT report this value at all'.

7.6.3.6 pywetterturnier::stationclass::stationclass::wmo

Station WMO number.

The documentation for this class was generated from the following file:

- pywetterturnier/stationclass.py

8 File Documentation

8.1 pywetterturnier/__init__.py File Reference

Namespaces

- namespace pywetterturnier

8.2 pywetterturnier/database.py File Reference

Classes

- class pywetterturnier::database::database

  This is the main database handler class.

Namespaces

- namespace pywetterturnier::database

8.3 pywetterturnier/getobs.py File Reference

Classes

- class pywetterturnier::getobs::getobs

Namespaces

- namespace pywetterturnier::getobs

8.4 pywetterturnier/groupbets.py File Reference

Namespaces

- namespace pywetterturnier::groupbets

8.5 pywetterturnier/importbets.py File Reference

Classes

- class pywetterturnier::importbets::importbets
  USED TO IMPORT OLD ARCHIVE DATA FROM THE OLD WETTERTURNIER.

Namespaces

- namespace pywetterturnier::importbets

8.6 pywetterturnier/judgingclass20021206.py File Reference

Classes

- class pywetterturnier::judgingclass20021206::judging
  This is a judgingclass - a class used to compute the points a user gets on a specific weekend.

Namespaces

- namespace pywetterturnier::judgingclass20021206

8.7 pywetterturnier/migrategroups.py File Reference

Classes

- class pywetterturnier::migrategroups::migrategroups

Namespaces

- namespace pywetterturnier::migrategroups

8.8 pywetterturnier/mitteltip.py File Reference

Namespaces

- namespace pywetterturnier::mitteltip

Functions

- def pywetterturnier::mitteltip::mitteltip

8.9 pywetterturnier/stationclass.py File Reference

Classes

- class pywetterturnier::stationclass::stationclass
  A small class holding all infors for a specific WMO station.

Namespaces

- namespace pywetterturnier::stationclass

8.10 pywetterturnier/utils.py File Reference

Namespaces

- namespace pywetterturnier::utils

Functions

- def pywetterturnier::utils::inputcheck
- def pywetterturnier::utils::usage
- def pywetterturnier::utils::readconfig
- def pywetterturnier::utils::tdate2string
- def pywetterturnier::utils::nicename
- def pywetterturnier::utils::exit

Index