Wetterturnier Documentation

Release 2018-01-01

Reto

Jan 19, 2022

Contents

This is the documentation for the wetterturnier.de data repository which contains several tools to process and extract data. The code in this repository is relatively specific but might be a good starting point if you would like to setup a similar system.

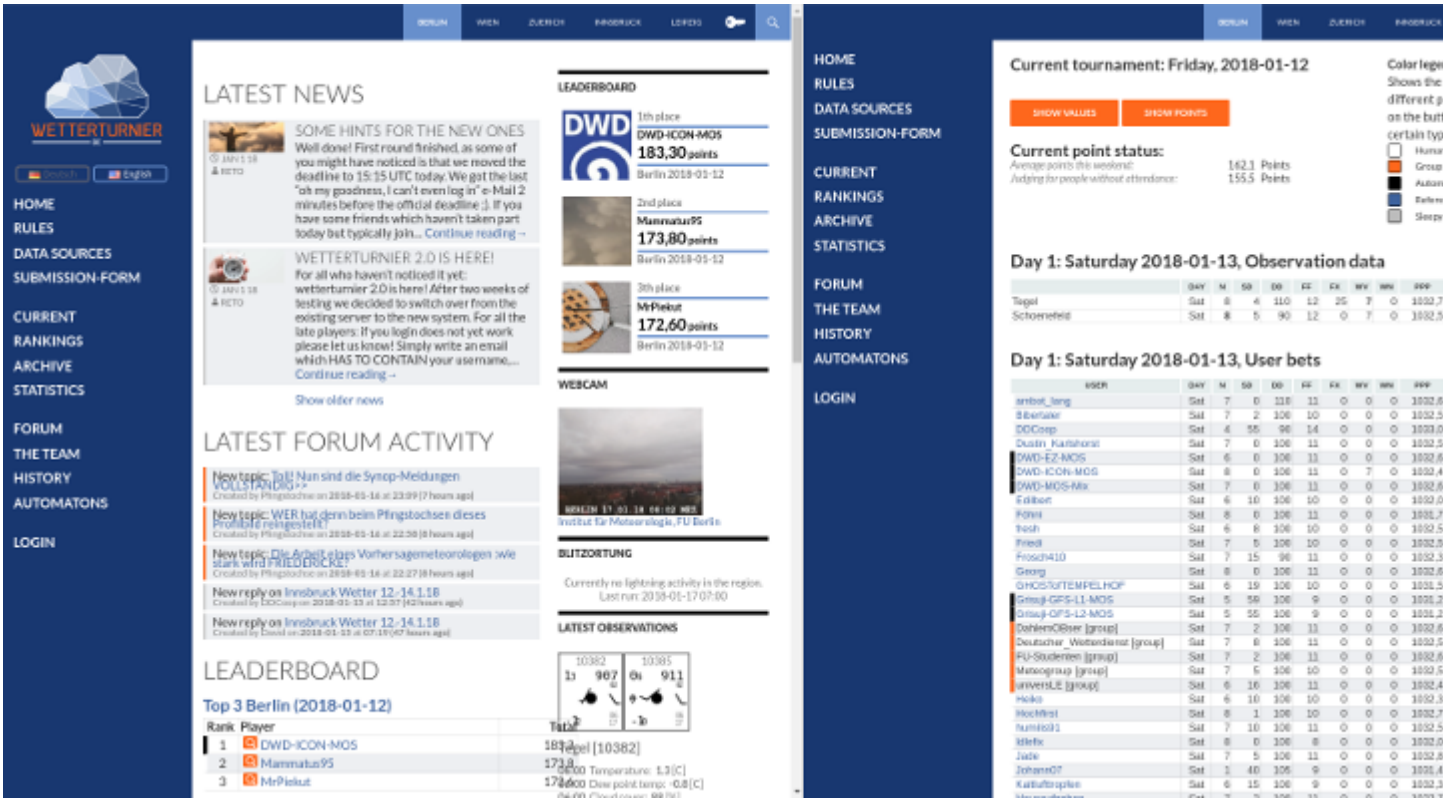
CHAPTER 1

What is “Wetterturnier”

The “Berliner Wetterturnier” as it has been known as in the beginning was launched in the year 2000 at the Institute of Meteorology at the FU Berlin. Since 2005 five cities in Central Europe are included.

Wetterturnier is a platform where *hobby meteorologists*, *experts* and *statistical forecast model developer* battle against each other. The goal is to predict a set of meteorological variables, such as sunshine duration, wind speed, or temperature as good as possible for the consecutive two days.

This plugin is the frontend core of the whole system providing full wordpress integration (user management, messaging services, forums) and the platform where our users can *submit their forecasts/bets*. Furthermore this plugin provides live ranking tables, a leader-board, a data archive, and access to a set of important data sets such as observations and forecast maps.



Please note that this is only one part of the system. To get the whole system running the Wetterturnier Wordpress Plugin. For more information please visit the *documentation on readthedocs <http://wetterturnier-backend.readthedocs.io/en/latest/overview.html>_*.

CHAPTER 2

Wetterturnier Data Tools

This repository is part of the Wetterturnier.de system. The documentation for this repository can be found on readthedocs

CHAPTER 3

GISCobservations

```
virtualenv --no-site-packages venv
source venv/bin/activate   # activate virtualenv
pip install mysqlclient    # database access
pip install matplotlib     # For the synop symbols


export BUFR_TABLES=/path/to/your/bufr/tables
cd GISCobservations
python bufr.py
```

CHAPTER 4

License Information

The software in this repository is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. The full `LICENSE` file is included in the repository and/or can be found on gnu.org.

4.1 Overview

This is an overvie over the whole Wetterturnier.de system wich consists of a set of different tools split into separate repositories.

Note: You are currently looking at the documentation of the **wetterturnier-data** repository documentation (blueish box bottom right on the image below).

All parts (except data sources and logins, of course) are made publicly available trough <github. Some links to the repositories:

- Wetterturnier Wordpress Plugin: github repository, documentation on readthedocs. As the name says: contains the wordpress plugin. Depends on the "Wetterturnier Backend" to get the points and rankings.
- Wetterturnier Wordpress Theme: github repository, contains the theme used on Wetterturnier.de, a Wordpress child theme based on the Wordpress twentyfourteen theme.
- Wetterturnier Backend: github repository, documentation on readthedocs, contains the python code to compute automatons, points, and rankings.
- Wetterturnier Data Backend (the one you are currently looking at): github repository, this documentation on readthedocs. Contains a set of tools to process/prepare data. Relatively specific for Wetterturnier.de but might be a starting point.

CHAPTER 5

Parts and Pieces

5.1 CreateSynopSymbols

This will be outdated soon when no weather type reports are delivered anymore. This is very quick'n'dirty code to procude synop style images using python. For Marcus Bayer this was always the most important element of Wetterturnier wherefore I've implemented it using this code.

The script reads trought he observation database (see table-live) to get the latest observations for all stations configured in the `config.conf` file. For each station and observed time a png figure will be produced once (won't be re-created if output figure exists).

Uses the *synopsymbol.synopsymbol*, see below. Requires the python matplotlib package to be installed.

To get the script run:

```
## Make a copy of the config template file and adjust
## the settings, namely mysql database access information
## and input/output directories in the [essentials] and [additionals]
## section.
cp config.conf.template config.conf

## Execute script (keep care using the virtualenv if you do so)
python CrateSynopSymbols.py
```

5.1.1 Class: synopsymbol

class synopsymbol.**synopsymbol**(*config*)

Synopsymbol class extending the drawbarb class drawing the vector wind barbs onto the figure.

__init__(*config*)

Initialize a synopsymbol class. Inputs needed: config from @see readconfig.readconfig. It contains the fonts and other stuff we need later. @return Returns the initialized class itself.

_open_figure_()
Helper function opening the new matplotlib.pyplot.figure object, setting axis properly. @param No input parameters, all needed is on self.config. @return No return, stores axis and figure handler onto self.ax and self.fig.

_save_figure_(*file*)
Saves the figure self.fig into the output file specified. @param file. Required, string. Name of the output file. @return No return.

5.2 ForecastProducts

Todo: Documentation has to be added (Reto).

5.3 GISCobservations

This thing is named **GISC observations** as we get some data from the DWD GISC interface. Ideally all data would come from this one system, however, that was not possible (did not got access, furthermore station 11320 Universitaet Innsbruck is not included in the GISC at all).

Note: There are is a version `extractBUFReccodes.py` which has the very same structure as `extractBUFRperl.py` and makes use of the ecmwf eccodes python library to extract the BUFR files. This is non-finished code! However, in case one has to switch over to the eccodes library one might use this draft. All you would have to do at the end is to change the include in the `bufr.py` file.

However, the documentation only provides information about the currently used `extractBUFRperl.py` script.

5.3.1 The worker script

The main script to be executed is `bufr.py`. If started without any input arguments the default input folders will be checked for new incoming bufr files. There are two incoming folders specified via `config.conf`. Depending on the folder where the files are stored the data get different labels in the database, either *essential* (means open data, can be used and downloaded by everyone) or *additional* (closed data, access will only be given to logged in users when using the Wetterturnier Wordpress Plugin. For both data types (essential and additional) an incoming directory (`indir`) and an outgoing directory (`outdir`) is specified in the `config.conf` file.

The script `bufr.py` automatically checks the incoming folders for new files. If there are new files the files are processed using `extractBUFRperl::extractBUFR` and moved into the output directory. The will be stored either in a subfolder `error` if the BUFR file could not have been extracted/processed or in a subfolder `processed` if successfully processed.

To run the script please note that the corresponding BUFR tables have to be available. They can either be located in the system wide default folder or specified via environment variable `BUFR TABLES`. Note that some BUFR files require custom BUFR TABLE files (e.g., for a specific subcentre using custom BUFR entries). WMO style BUFR TABLES can for example be downloaded on the ECMWF website. WARNING: the BUFR tables in this archive have the suffix `.txt` while `bufrread.pl` is looking for `.TXT` files. Simple solution: link all your files `.txt` to `.TXT` and try.

To get this script to run:

```
## Make a copy of the config template file and adjust
## the settings, namely mysql database access information
## and input/output directories in the [essentials] and [additionals]
## section.
cp config.conf.template config.conf

## If required: set BUFR TABLES environment variable
export BUFR_TABLES=/path/to/your/bufrtables

## Execute script (keep care using the virtualenv if you do so)
python bufr.py
```

For testing a specific file can be specified using the `-f/--file` flag. In this case this file will be read and *not moved* after execution.

```
## Processing af specific bufr file (keep care using the virtualenv if you do so)
python bufr.py --file <path/to/buf/file>
```

5.3.2 The cleanup script

To keep the databaes small only a subset of data will be archived while the live table is a rolling table containing the last N days of data only. Furthermore, old unused BUFR files should be removed from the disc. The `CleanUp.py` script does this job using the configuration from the `config.conf` file (mysql access config and the `[cleanup]` section).

To get the script running:

```
## Make a copy of the config template file if you havn't done this
## yet and adjsut the settings, namely mysql database access information
## and input/output directories in the [essentials] and [additionals]
## section. For the archive table: check the list of stations in the
## [cleanup] section which should be moved from the live table (``srctable``)
## to the archive table (``dsttable``).
cp config.conf.template config.conf

## Run the script
python cleanup.py
```

The script ...

- Reads the `config.conf` file
- Creates an object of class `cleanup` * Deletes old raw (BUFR) files from the disc * Moves a subset of observations from the live table into the archive table * Removes old observations from the live table

5.3.3 Class: cleanup

This is the class used by the `CleanUp.py`.

class cleanup.**cleanup**(*config*)

Setting up the class to clean files and databases used for processing incoming observations.

Parameters **config** (str) – Name of the config file to read.

cleanup_live_table()

We have a live and an archive table. These two tables are defined in the config.conf file. Here we are

deleting all observations from the live table ('srctable') which are older than about 'db_days' days (as well defined in the config.conf file).

closeDB()
Closing database.

delete_old_raw_files()
Method deleting files from disc in the directory 'outdir' as defined in the config.conf file. We do NOT decide between synop/bufr or processed/error here. Just kill them if they are older than 'file_days' as specified in config.conf.

getOldFiles(*dirPath*, *maxage*, *postfix*)
List old files on disc.

Parameters

- **dirPath** (str) – Path to the directory which should be checked.
- **maxage** (int) – Timestamp, files older than this will be considered to be old and marked for deletion.
- **postfix** (str) – File postfix. Only files where the postfix matches (not case sensitive) will be considered.

Returns A list of all files under dirPath older than days.

Return type list

live_database_to_archive()
I would like to store some observation data longer than just a few days - however - we wont create a copy of the WMO observation data archive or simething. Therefore we are just archiving some stations as defined in 'cleanup:stations' in the config.conf file. Move them from 'cleanup:srctable' to 'cleanup:dsttable' (see config.conf file).

5.3.4 Class: extractBUFR

Main class, extracting observations from BUFR data files using the Geo::BUFR `bufrread.pl` script. `bufrread.pl` converts the BUFR files into ASCII whcih will be parsed by `extractBUFRperl::extractBUFR` and stored into the database.

class `extractBUFRperl.`**extractBUFR**(*file*, *config*, *stint*, *verbose*, *filterfile=None*)
Main class, extracting data from the BUFR file.

This object uses *subprocess.Popen* to call the Geo::BUFR bufrread.pl file (see http://search.cpan.org/dist/Geo-BUFR/lib/Geo/BUFR.pm, https://wiki.met.no/bufr.pm/start). If not installed None will be returned. To install Geo::BUFR check the readme of the package. It is as simple as:

```
cpan Geo::BUFR
```

Please note that you will also have to have the BUFRTABLES installed on your system at either one of the default locations or by setting the environment variable `BUFR_TABLES=<path>` corresponding to the location of the bufr files.

BUFR Tables can e.g. be downloaded here: <https://software.ecmwf.int/wiki/display/BUFR/BUFRDC+Home>'_. The files in this archive are named `.txt` while `.TXT` files are expected. bufrread.pl will drop a corresponding message. Simply link the `.txt` files to a corresponding `.TXT` version in your BUFR_TABLES folder to get around this.

Parameters

- **config** (str) – Name of the config file.

- **stint** (str) – Used to store a flag into the database from which source the messages come. In this case "bufr". Keep in mind that the database column type is "ENUM" and only allows a distinct set of strings.
- **verbose** (bool) – Boolean True/False whether the object should be verbose or not.
- **filterfile** (str) – Default is None, a filter file can be specified forwarded to Geo::BUFR bufrread.pl.

__check_bufrdesc_and_add_if_necessary__(*rec*, *param*)

Adding bufr entry to database table *bufrdesc* if necessary. Input rec is a bufrentry object. Input param has to be of class paramclass. Checks if entry is already in the bufrdesc database. If not, we have to add a row.

Parameters

- **rec** (*bufrentry*) – Object to be added.
- **param** (*bufrdesc*) – Bufr description object.

__check_displacement__(*rec*)

Check if current record is a time displacement specification. If so the value of the time displacement value will be returned as int in seconds. If not bool False is returned.

Parameters **rec** (*bufrentry*) – Object to check.

Returns Returns bool FALSE or int.

__check_sensorheight__(*rec*)

Check if current record is a sensorheight specification. If so the value of the sensorheight value will be returned (float). If not a bool False is returned.

Parameters **rec** (*bufrentry*) – Object to check.

Returns Returns bool FALSE or float.

__check_verticalsign__(*rec*)

Check if current record is a vertical significance specification. If so the value of the vertical significance value will be returned (absolute value as integer). If not a bool False is returned.

Parameters **rec** (*bufrentry*) – Object to check.

Returns Returns bool FALSE or int.

__get_param_obj__(*search*, *displacement*, *verticalsign*, *sensorheight*)

The config file bufr_config.conf contains a set of parameter definitions. This method is used to finde the appropriate parameter description given the inputs which directly come from the BUFR entry extracted from the BUFR file using Geo::BUFR buffread.pl.

We are therefore matching each data line from the BUFR file with one of our specified parameter configs from the bufr_config.conf and use them to further process the data.

Parameters

- **search** (burentry) – Bufrentry object.
- **displacement** (int) – Lates time displacement value, seconds.
- **verticalsign** (int) – Latest vertical significance value.
- **sensorheight** (float) – Latest sensor height value.

Returns Returns two values, the first one is a bool whether to drop the message or not. If no parameter entry can be matched to the current bufrentry this value is True (drop message, unknown). Else False will be returned (don't drop). The second argument is bool False if we cannot find the parameter entry, or a parameter entry of class *bufrdesc* else.

__getval__(*x*)
Get value: if the value is a string: simply return. Else convert value to `float`. If the value is extremely large or extremely small: return `MISSING_VALUE`.

Returns Properly prepare the value.

__init__(*file*, *config*, *stint*, *verbose*, *filterfile=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

__read_bufr_file__(*file*, *filterfile=None*)
Function reading the BUFR file. Actually calling the perl Geo::BUFR library to convert the binary files into ASCII table and pase the output to extract the necessary information.

Parameters

- **file** (`str`) – Path/Name of the BUFR file (binary file).
- **filterfile** (`str`) – Default None, dan be set and will be forwarded to Geo::BUFR bufrread.pl to set specific filters. If set only this subset of the bufr file will be extracted/processed.

Returns Returns a list of lists, each containing a set of *bufrentry* objects. The length of the most outer list corresponds to the number of messages in the BUFR file. The first nested lists are the messages each consisting of a set of *bufrentry* entries with the data.

Return type list

__showdata_sort_order__(*force=None*)
Takes care of the order of the columns in the output.

__weakref__
list of weak references to the object (if defined)

commit()
Alias for MySQLdb.commit.

cursor()
Alias for MySQLdb.close.

Returns Returns a MySQL.cursor object.

dbClose()
Alias for MySQLdb.close.

dbConnect()
Method to open the database connection. Uses the settings on self.config. No return, saves the database handler on the object itself.

extractdata()
Looping trough self.raw (raw information returned by __read_bufr_file__ and prepares the data.

load_bufr_description(*table*)
Loading data from 'table' and returns a list object containing one 'bufrdesc' object for each of the rows in the database.

Parameters **table** (`str`) – Name of the database table containing the bufr descriptions.

Returns Returns a list of *bufrdesc* objects containing the definition/description.

Return type list

manipulatedata()
Manipulate data. Is looking for some meta information such as `wmoblock`, `statnr`, `year`, `month`,

`hour`, and `minute` and creates the columns `datumsec` (unix time stamp), `stdmin` (hour/minute integer, e.g., 7:00 UTC is 700), and `statnr` (a combination of the wmoblock and station number information from the bufr file).

prepare_data()
Prepares the data. Puts the data we found bevore in the single messages into a matrix style variable called "res". Stores parameter (column description of the matrix) and the data matrix into self.PREPARED.

showdata()
Helper function to print the data to stdout.

showdropped()
If a bufrentry cannot be attributed (is not defined by bufr_config.conf) we will ignore these lines. To see what has been dropped and whether there is important information being dropped the dropped lines will be kept.

This method allows to print the dropped lines to stdout.

update_stations()
Update station database. Update the station database with the information from the bufr message. Plase note that we do simply update the database row and do not take care of history (e.g., if a station would be renamed or moved the latest name/location will be stored and the old information is simply overwritten).

write_to_db()
Write data to database.

5.3.5 Class: bufrentry

`extractBUFRperl::extractBUFR` uses the perl library Geo::BUFR `bufrread.pl` to extract the binary BUFR files (called internally via `subprocess.Popen`)

The script `bufrread.pl` returns the content of the BUFR file in ASCII where each line in the data section corresponds to one BUFR entry. `extractBUFRperl::extractBUFR` stores each line in a `extractBUFRperl::bufrentry` object which are easy to iterate over.

class `extractBUFRperl.`**bufrentry**(*string*, *width*)
This is a small helper class. I store all entries from the bufr file in such bufrentry classes. A bufrenry class contains the specification of one single message. E.g., bufrid, value, description.

Parameters

- **string** (`str`) – A bufrentry is a line as extracted by the Geo::BUFR bufrread.pl perl script.
- **width** (`int`) – bufrread.pl allows to set a width for the description column. This width has to be known by *`bufrentry`* to be able to properly extract the information from this line.

__init__(*string*, *width*)
x.__init__(...) initializes x; see help(type(x)) for signature

__weakref__
list of weak references to the object (if defined)

show()
Allows to print the content of this object, mainly for development.

Returns No return, creates output on stdout.

string()
Helper method to output the content of this object to console.

Returns Returns the information from the object in a string format.

5.3.6 Class: bufrdesc

The class `extractBUFRperl::extractBUFR` uses `extractBUFRperl::bufrdesc` classes to handle the bufr parameter configuration read from the `bufr_config.conf` file. Each entry (`bufrentry`) read from the BUFR file has to match a parameter configured in `bufr_config.conf` and will be dropped else.

For ease of use the configuration of `bufr_config.conf` is read piece-wise and each config is stored as a `extractBUFRperl::bufrdesc` object.

class `extractBUFRperl.`**bufrdesc**(*rec*, *cols*)

This is a small helper class. I am loading the bufrdesc database as a list ob such bufrdesc classes which are easily iteratable. Used to store each record (each row of the *bufrdesc database table* as an object which is easy to iterate over.

Parameters

- **rec** (`tuple`) – A record from the bufrdesc database table. The elements of the `tuple` are described by the second input argument `cols`.
- **cols** (`list`) – List of `str` describing the elements in the first argument (rec `tuple`).

__init__(*rec*, *cols*)

x.__init__(...) initializes x; see help(type(x)) for signature

__weakref__

list of weak references to the object (if defined)

get(*what*)

Returns element corresponding to input string 'what'. If we cant find it in the columns from the database: stop!

Parameters **what** (`str`) – Element to be returned.

Returns Returns the corresponding element if available, else stop.

show()

Shows content of the object

5.4 blitzortung

I dont want to spend too much time to explain this mini thing. The Institute for Atmospheric and Cryospheric Sciences (ACINN) is a member of the Blitzortung.org network and allowd to redistribute the raw data. At the ACINN there is a script running to procude live lightning data plots. During this process a small sqlite3 file is created which is copied via `ssh` to the prognose server (into the `blitzortung` folder of this repository).

The `blitzortug.R` script is run every X minutes via cron and checks the current sqlite file to draw a small map for each of our Wetterturnier cities (specified via `stations.txt`) and places a figure and a small file containing information about the last run (to check whether the data are outdated) in the `blitzortung` folder.

This folder is linked to the webserver to grant access on the frontend, namely via Wetterturnier Wordpress Plugin Lightning Activity Widget .

Note: If there is someting wrong with the data one might ask Reto Stauffer or Georg J. Mayr (from the ACINN) to see wheter there is someting wrong or has been changed.

Todo: If we could replace this script with a python script we might be able to somewhen remove the R installation from the server (except the data handling from the not yet published and not yet finished R package wetterturnier, ask Reto Stauffer).

To get the code run (requires the sqlite3 file from the sqlite folder):

```
## Simply to this:
Rscript blitzortung.R
```

5.5 Database Tables

5.5.1 live

The `live` table is used to store incoming observations. Please note that only a subset of all columns is shown in the table below. The script processing the observations and saving them into this database table automatically creates additional columns if there are data. `...` in the table indicate the data columns (e.g,. temperature observations, cloud cover observations, ...).

The `live` table is a rolling database containing the latest observations for all incoming stations. The script *CleanUp.py* cleans the database from time to time moving the observations for some specific stations into the *archive* database table and deletes all others.

Table 1: [Autogenerated table scheme of table "live] Rolling database for (raw) incoming observations."

Field	Type	Null	Key	Default	Extra
statnr	int(11)	NO	MUL	None	
datum	int(8)	NO	MUL	None	
datumsec	int(11)	NO	MUL	None	
stdmin	smallint(4)	NO		None	
msgtyp	enum('na','bufr','synop')	YES		na	
stint	enum('na','essential','additional')	YES		na	
utime	timestamp	NO	MUL	CURRENT_TIMESTAMP	
ucount	tinyint(3) unsigned	YES		0	
...	...	...	...	...	...

- Non-unique key named *bufr_statnr* on `(statnr)`
- Non-unique key named *bufr_datumsec* on `(datumsec)`
- Non-unique key named *bufr_datum* on `(datum)`
- Non-unique key named *bufr_einspiel* on `(utime)`
- **Unique-key** named *bufr_statnr_datumsec_msgtyp* on `(statnr, datumsec, msgtyp)`

5.5.2 archive

The archive table has the same structure as the *live* database table and contains long-term archive data for a set of specified stations. We keep the data for the tournament stations and drop all others as we don't want to keep a copy of all observations (would be a huge database and an unnecessary and unused copy of everything).

Table 2: [Autogenerated table scheme of table “archive] Archive table, contains long-term observations (copy of the live table) for specified stations.”

Field	Type	Null	Key	Default	Extra
statnr	int(11)	NO	MUL	None	
datum	int(8)	NO	MUL	None	
datumsec	int(11)	NO	MUL	None	
stdmin	smallint(4)	NO		None	
msgtyp	enum(‘na’,’bufr’,’synop’)	YES		na	
stint	enum(‘na’,’essential’,’additional’)	YES		na	
utime	timestamp	NO	MUL	CURRENT_TIMESTAMP	
ucount	tinyint(3) unsigned	YES		0	
…	…	…	…	…	…

- Non-unique key named *bufr_statnr* on `(statnr)`
- Non-unique key named *bufr_datumsec* on `(datumsec)`
- Non-unique key named *bufr_datum* on `(datum)`
- Non-unique key named *bufr_einspiel* on `(utime)`
- **Unique-key** named *bufr_statnr_datumsec_msgtyp* on `(statnr, datumsec, msgtyp)`

5.5.3 stations

Station information as read from the BUFR files.

Table 3: [Autogenerated table scheme of table “stations] Station meta information as received from the BUFR messages. Rows will be updated, no historical information kept (if a station e.g., would be moved or renamed).”

Field	Type	Null	Key	Default	Extra
statnr	smallint(11) unsigned	NO	MUL	None	
nr	tinyint(3) unsigned	NO		None	
name	varchar(150)	YES		None	
lon	decimal(10,4)	NO		None	
lat	decimal(10,4)	NO		None	
hoehe	smallint(6)	NO		None	
hbaro	smallint(6)	YES		-999	
changed	timestamp	NO		CURRENT_TIMESTAMP	

- Non-unique key named *stations_statnr* on `(statnr)`

5.5.4 bufrdesc

BUFR description as read from the BUFR files.

Table 4: [Autogenerated table scheme of table "bufrdesc] Stores bufr data description handlers. Contain variable description and original BUFRID."

Field	Type	Null	Key	Default	Extra
bufrid	smallint(3) unsigned	NO		None	
param	varchar(35)	NO	PRI	None	
desc	varchar(150)	YES		None	
unit	varchar(35)	YES		None	
period	mediumint(8) unsigned	YES		0	
offset	float	YES		0	
factor	float	YES		1	
changed	timestamp	NO		CURRENT_TIMESTAMP	

- **Unique-key** named *bufrdesc_param* on `(param)`

Index

Symbols

B

C

D

E

G

L

M

P

S

U

W