# Package: kwb.pilot (via r-universe)

August 21, 2024

**Type** Package

**Title** Importing, Aggregating and Visualising Data From KWB Pilot Plants

**Version** 0.3.0

**Description** Collects, aggregates and visualises operational and analytical data from water suppliers (including a standardised reporting document).

**License** MIT + file LICENSE

**URL** https://github.com/kwb-r/kwb.pilot

**BugReports** https://github.com/kwb-r/kwb.pilot/issues

**Depends** R (>= 3.6)

**Imports** data.table (>= 1.13.4), dbplyr (>= 2.0.0), digest (>= 0.6.27), dplyr (>= 1.0.2), dygraphs (>= 1.1.1.6), fasttime (>= 1.0.2), fs (>= 1.5.0), fst (>= 0.9.4), ggforce (>= 0.3.2), ggplot2 (>= 3.3.2), htmlwidgets (>= 1.5.3), influxdbclient (>= 0.1.0), janitor (>= 2.0.1), kwb.nextcloud (>= 0.3.0), kwb.utils (>= 0.7.0), leaflet (>= 2.0.3), lubridate (>= 1.7.9.2), magrittr (>= 2.0.1), plotly (>= 4.9.2.1), plyr (>= 1.8.6), readr (>= 1.4.0), readxl (>= 1.3.1), rmarkdown (>= 2.6), rvest (>= 0.3.6), shiny (>= 1.5.0), shinythemes (>= 1.1.2), stringr (>= 1.4.0), tibble (>= 3.0.4), tidyr (>= 1.1.2), tidyselect (>= 1.1.1), withr (>= 2.3.0), xml2 (>= 1.3.2), xts (>= 0.12.1)

**Suggests** covr (>= 3.5.1), knitr (>= 1.30), remotes (>= 2.2.0), testthat (>= 3.0.1), usethis (>= 2.0.1)

**VignetteBuilder** knitr

**Remotes** github::kwb-r/kwb.nextcloud, github::kwb-r/kwb.utils

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.2

**Repository** https://kwb-r.r-universe.dev

**RemoteUrl** https://github.com/KWB-R/kwb.pilot

**RemoteRef** HEAD

**RemoteSha** e7189f774e3b675a30410a05c13bcd075c07fe92

# Contents

---

`add_label` *Helper function: add label ("SiteName_ParaName_Unit_Method")*

---

## Description

Helper function: add label ("SiteName_ParaName_Unit_Method")

## Usage

```
add_label(
  df,
  col_sitename = "SiteName",
  col_parametername = "ParameterName",
  col_parameterunit = "ParameterUnit",
```

```
  col_method = "Method_Org"
)
```

## Arguments

| | |
|---|---|
| df | data frame containing at least a columns "SiteName", "ParameterName", "ParameterUnit" and optionally "Method_Org" (if not existent no "Method_Org" will be available!) |
| col_sitename | column in df containing site name (default: "SiteName") |
| col_parametername | column in df containing parameter name (default: "ParameterName") |
| col_parameterunit | column in df containing parameter unit (default: "ParameterUnit") |
| col_method | column in df containing method code (default: "Method_Org") |

## Value

returns input data frame with added column "SiteName_ParaName_Unit_Method"

---

`add_parameter_metadata` *Helper function: add parameter metadata*

---

## Description

Helper function: add parameter metadata

## Usage

```
add_parameter_metadata(
  df,
 meta_parameter_path = package_file("shiny/basel/data/metadata/meta_parameter.csv")
)
```

## Arguments

| | |
|---|---|
| df | data frame containing at least a column "ParameterCode" |
| meta_parameter_path | Define path of "meta_parameter.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/ |

## Value

returns input data frame with joined metadata (parameter codes/ methods not included in meta_parameter file will not be imported!!!!)

## add_site_metadata *Helper function: add site metadata*

### Description

Helper function: add site metadata

### Usage

```
add_site_metadata(
  df,
  df_col_sitecode = "SiteCode",
  meta_site_path = package_file("shiny/basel/data/metadata/meta_site.csv")
)
```

### Arguments

| | |
|---|---|
| df | data frame containing at least a column "SiteCode" |
| df_col_sitecode | column in df containing site code (default: "SiteCode") |
| meta_site_path | Define path of "meta_site.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/metada |

### Value

returns input data frame with joined metadata

## aggregate_export_fst_berlin_f *Berlin-Friedrichshagen: aggregate and export to fst*

### Description

Berlin-Friedrichshagen: aggregate and export to fst

### Usage

```
aggregate_export_fst_berlin_f(
  year_month_start = "2019-11",
  year_month_end = format(Sys.Date(), "%Y-%m"),
  compression = 100
)
```

### Arguments

| | |
|---|---|
| year_month_start | start year month (default: '2019-11') |
| year_month_end | end year month (default: current month) |
| compression | (default: 100) |

### Value

exports data for each month into subfolder: /data/fst/year-month

---

## aggregate_export_fst_berlin_s

*Berlin-Schoenerlinde: aggregate and export to fst*

---

### Description

Berlin-Schoenerlinde: aggregate and export to fst

### Usage

```
aggregate_export_fst_berlin_s(
  year_month_start = "2017-04",
  year_month_end = format(Sys.Date(), "%Y-%m"),
  compression = 100
)
```

### Arguments

| | |
|---|---|
| year_month_start | start year month (default: '2017-04') |
| year_month_end | end year month (default: current month) |
| compression | (default: 100) |

### Value

exports data for each month into subfolder: /data/fst/year-month

aggregate_export_fst_berlin_t *Berlin-Tiefwerder: aggregate and export to fst*

## Description

Berlin-Tiefwerder: aggregate and export to fst

## Usage

```
aggregate_export_fst_berlin_t(
  year_month_start = "2017-06",
  year_month_end = format(Sys.Date(), "%Y-%m"),
  compression = 100
)
```

## Arguments

| | |
|---|---|
| year_month_start | start year month (default: '2017-06') |
| year_month_end | end year month (default: current month) |
| compression | (default: 100) |

## Value

exports data for each month into subfolder: /data/fst/year-month

aggregate_export_fst_mbr4 *MBR4.0: aggregate and export to fst*

## Description

MBR4.0: aggregate and export to fst

## Usage

```
aggregate_export_fst_mbr4(
  siteData_raw_list = tidy_mbr4_data(read_mbr4()),
  compression = 100
)
```

### Arguments

| | |
|---|---|
| siteData_raw_list | tidy MBR4 data as retrieved by tidy_mbr4_data, (default: kwb.pilot::tidy_mbr4_data(kwb.pilot::read_m |
| compression | (default: 100) |

### Value

exports data for each month into subfolder: /data/fst/year-month

---

## calculate_operational_parameters

*Calculate operational parameters*

---

### Description

Calculate operational parameters

### Usage

```
calculate_operational_parameters(
  df,
  calc_list = list(Redox_Out = "(Redox_Out1+Redox_Out2)/2", Redox_Diff =
   "Redox_Out - Redox_In", Power_pump = "Up*Ip", Power_cell = "Uz*Iz", Pump_WhPerCbm =
    "Power_pump/(Flux/1000)", Cell_WhPerCbm = "Power_cell/(Flux/1000)"),
  calc_list_name = c("Mean redox potential in tank",
    "Difference (outflow - inflow) of redox potential", "Power demand of pump",
    "Power demand of cell", "Specific energy demand of pump",
    "Specific energy demand of cell"),
  calc_list_unit = c("mV", "mV", "W", "W", "Wh/m3", "Wh/m3"),
 calc_paras = c("Redox_Out1", "Redox_Out2", "Redox_In", "Flux", "Up", "Ip", "Uz",
    "Iz")
)
```

### Arguments

| | |
|---|---|
| df | a data frame as retrieved by import_data_haridwar() |
| calc_list | list with calculation operations to be carried out (default: list(Redox_Out = "(Redox_Out1+Redox_Out2)/2", Redox_Diff = "Redox_Out - Redox_In", Power_pump = "Up*Ip", Power_cell = "Uz*Iz", Pump_WhPerCbm = "Power_pump/Flux/1000", Cell_WhPerCbm = "Power_cell/Flux/1000")) |
| calc_list_name | full names of parameters to be used for plotting for each calculation specified wit 'calc_list'. default: c('Tank water: Mean redox potential", 'Difference (outflow - inflow) of redox potential', 'Power demand of pump', 'Power demand of cell', 'Specific energy demand of pump', Specific energy demand of cell') |
| calc_list_unit | units of parameters to be used for plotting for each calculation specified wit 'calc_list'. default: c('mV', 'mV', 'Wh', 'Wh', 'Wh/m3', 'Wh/m3') |

calc_paras — a vector with parameter codes used for performing calculations defined in 'calc_list' (default: c('Redox_Out1', 'Redox_Out2', 'Redox_In', 'Flux', 'Up', 'Ip', 'Uz', 'Iz'))

## Value

dataframe with calculated operational parameters

## Examples

```
## Not run:
haridwar_raw_list <- import_data_haridwar()
myDat <- calculate_operational_parameters(df = haridwar_raw_list)

## End(Not run)
```

---

# calculate_operational_parameters_berlin_f

*Calculate operational parameters for Berlin-Friedrichshagen*

---

## Description

Calculate operational parameters for Berlin-Friedrichshagen

## Usage

```
calculate_operational_parameters_berlin_f(
  df,
 calc_list = list(vfrPerm = "`Durchfluss_Rohwasser` - `Durchfluss_Konzentrat`", yield
  = "100*(`Durchfluss_Rohwasser` - `Durchfluss_Konzentrat`) / `Durchfluss_Rohwasser`",
    conLoop =
  "(`Durchfluss_Rohwasser`*`LF_Rohwasser` + `Durchfluss_Rezirkulation`*`LF_Konzentrat`)/(`Durchflus
    recovery = "100*(1 - `LF_Permeat` / conLoop)", deltaPreProcConc =
  "`Druck_Arbeitsdruck` - `Druck_Konzentrat`", flux = "vfrPerm / (4 * 7.6)", cfv =
  "(`Durchfluss_Rohwasser`+ `Durchfluss_Rezirkulation`) / ((pi * 0.0095^2) * 1000 * 3600)",
      tmp = "((`Druck_Arbeitsdruck` + `Druck_Konzentrat`) / 2) - `Druck_Permeat`",
    nwpt =
  "normalised_permeate_flow(tempFeed = `Temperatur_Rohwasser`,\n                          conLoop = `
        nwpr = "- ((1 - (nwpt / vfrPerm))) * 100"),
 calc_list_name = c("Durchfluss Permeat", "Ausbeute", "Leitfähigkeit Rezirkulation",
    "Rückhalt", "Druckverlust (Feed - Konzentrat)", "Flux",
  "Überströmungsgeschwindigkeit", "Transmembrandruck", "Normalisierter Permeatstrom",
    "Relativer Permeatstrom"),
 calc_list_unit = c("l/h", "%", "\xb5S/cm", "%", "bar", "l/h/m2", "m/s", "bar",
    "l/h", "%"),
 calc_paras = c("Durchfluss_Rohwasser", "Durchfluss_Konzentrat",
    "Durchfluss_Rezirkulation", "Druck_Arbeitsdruck", "Druck_Rohwasser",
```

```
   "Druck_Konzentrat", "Druck_Permeat", "LF_Permeat", "LF_Rohwasser", "LF_Konzentrat",
    "Temperatur_Rohwasser")
)
```

## Arguments

| | |
|---|---|
| df | a data frame as retrieved by import_data_berlin_f() |
| calc_list | list with calculation operations to be carried out |
| calc_list_name | full names of parameters to be used for plotting for each calculation specified wit 'calc_list'. |
| calc_list_unit | units of parameters to be used for plotting for each calculation specified wit 'calc_list'. |
| calc_paras | a vector with parameter codes used for performing calculations defined in 'calc_list' |

## Value

dataframe with calculated operational parameters

## Examples

```
## Not run:
raw_list <- import_data_berlin_f()
myDat <- calculate_operational_parameters_berlin_f(df = raw_list)

## End(Not run)
```

---

# calculate_operational_parameters_berlin_s

*Calculate operational parameters for Berlin-Schoenerlinde*

---

## Description

Calculate operational parameters for Berlin-Schoenerlinde

## Usage

```
calculate_operational_parameters_berlin_s(
  df,
 calc_list = list(deltaSAK = "(1-SCAN_SAK_Ablauf/SCAN_SAK_Zulauf)*100", Ozoneintrag =
    "(C_O3_Zugas - C_O3_Abgas)*Q_Gas/Q_Ozonanlage"),
  calc_list_name = c("delta SAK", "Ozoneintrag"),
  calc_list_unit = c("%", "mg-O3/L"),
 calc_paras = c("SCAN_SAK_Ablauf", "SCAN_SAK_Zulauf", "C_O3_Zugas", "C_O3_Abgas",
    "Q_Gas", "Q_Ozonanlage")
)
```

### Arguments

| | |
|---|---|
| df | a data frame as retrieved by read_wedeco_data() |
| calc_list | list with calculation operations to be carried out (default: list(deltaSAK = "(1-SCAN_SAK_Ablauf/SCAN_SAK_Zulauf)*100", Ozoneintrag = "(C_O3_Zugas - C_O3_Abgas)*Q_Gas/Q_Ozonanlage")) |
| calc_list_name | full names of parameters to be used for plotting for each calculation specified wit 'calc_list'. default: c('delta SAK', 'Ozoneintrag') |
| calc_list_unit | units of parameters to be used for plotting for each calculation specified wit 'calc_list'. default: c("percent", "mg-O3/L") |
| calc_paras | a vector with parameter codes used for performing calculations defined in 'calc_list' (default: c("SCAN_SAK_Ablauf", "SCAN_SAK_Zulauf", "C_O3_Zugas", "C_O3_Abgas", "Q_Gas", "Q_Ozonanlage")) |

### Value

dataframe with calculated operational parameters

### Examples

```
## Not run:
raw_list <- read_wedeco_data()
myDat <- calculate_operational_parameters_berlin_s(df = raw_list)

## End(Not run)
```

---

## calculate_operational_parameters_berlin_t

*Calculate operational parameters for Berlin-Tiefwerder*

---

### Description

Calculate operational parameters for Berlin-Tiefwerder

### Usage

```
calculate_operational_parameters_berlin_t(
  df,
  calc_list = list(recovery = "100*`FY-20-01`/`FT-10-01`"),
  calc_list_name = c("recovery"),
  calc_list_unit = c("%"),
  calc_paras = c("FY-20-01", "FT-10-01")
)
```

### Arguments

| | |
|---|---|
| df | a data frame as retrieved by read_pentair_data() |
| calc_list | list with calculation operations to be carried out (default: list(recovery = "100*‘FY-20-01‘/‘FT-10-01‘")) |
| calc_list_name | full names of parameters to be used for plotting for each calculation specified wit 'calc_list'. default: c('recovery') |
| calc_list_unit | units of parameters to be used for plotting for each calculation specified wit 'calc_list'. default: c("percent") |
| calc_paras | a vector with parameter codes used for performing calculations defined in 'calc_list' (default: c("FY-20-01", "FT-10-01") |

### Value

dataframe with calculated operational parameters

### Examples

```
## Not run:
raw_list <- read_pentair_data()
myDat <- calculate_operational_parameters_berlin_t(df = raw_list)

## End(Not run)
```

---

## calenderweek_from_dates

*Helper function: get calender weeks for time period*

---

### Description

Helper function: get calender weeks for time period

### Usage

```
calenderweek_from_dates(start = "2017-04-24", end = Sys.Date())
```

### Arguments

| | |
|---|---|
| start | start of period (default: '2017-04-24') |
| end | end of period (default: .Date()) |

### Value

data.frame with daily date sequence for and corresponding calendar week

`change_timezone` *Timezone change: changes time zone to user defined time zone*

## Description

Timezone change: changes time zone to user defined time zone

## Usage

```
change_timezone(df, tz = "UTC", col_datetime = "DateTime", debug = TRUE)
```

## Arguments

| | |
|---|---|
| `df` | a dataframe containing a datetime column |
| `tz` | timezone (default: "UTC") |
| `col_datetime` | name of the datetime column (default: "DateTime") |
| `debug` | print debug messages (default: TRUE) |

## Value

returns data frame with changed time zone

## References

Check possible "tz" arguments in column "TZ*" of table https://en.wikipedia.org/wiki/List_of_tz_database_time_zones for more details.

`check_env_influxdb_ultimate` *Helper Function: check if all environment variables for ULTIMATE InfluxDB are defined*

## Description

Helper Function: check if all environment variables for ULTIMATE InfluxDB are defined

## Usage

```
check_env_influxdb_ultimate()
```

## Value

TRUE if all defined, FALSE otherwise

## check_env_nextcloud *Helper Function: check if all environment variables for Nextcloud are defined*

### Description

Helper Function: check if all environment variables for Nextcloud are defined

### Usage

```
check_env_nextcloud()
```

### Value

TRUE if all defined, FALSE otherwise

## check_thresholds *Check thresholds*

### Description

Check thresholds

### Usage

```
check_thresholds(df, thresholds = get_thresholds())
```

### Arguments

| | |
|---|---|
| df | a dataframe as retrieved by import_data_haridwar() |
| thresholds | thresholds dataframe as retrieved by get_thresholds() (default: "raw") |

### Value

dataframe with thresholds check results for selected time period (i.e. whether Parameters are below/above min/max thresholds defined in dataframe 'thresholds')

## create_monthly_selection *Create monthly selection*

### Description

Create monthly selection

### Usage

```
create_monthly_selection(startDate = "2016-09-01", endDate = Sys.Date())
```

### Arguments

| | |
|---|---|
| startDate | (default: '2016-09-01') |
| endDate | (default: Sys.Date()) (default: "raw") |

### Value

dataframe with first/last day for each month between 'startDate' and 'endDate' month including a column 'label' (used in shiny app for month selection)

## create_report_batch *Report batch: creates batch file for report*

### Description

Report batch: creates batch file for report

### Usage

```
create_report_batch(
  batchDir = file.path(tempdir(), "batch_report"),
  batchName = "create_report.bat",
  report_path = NULL,
  report_config_path = NULL,
  open_in_explorer = TRUE
)
```

### Arguments

| | |
|---|---|
| batchDir | path to report batch directory (default: tempdir()) |
| batchName | name for report batch file(default: "create_report.bat") |
| report_path | (default: NULL) |
| report_config_path | (default: NULL) |
| open_in_explorer | open batchDir in Windows explorer (default: TRUE). Only working on a Windows system! |

---

## create_wedeco_metafile *Create WEDECO metafile data*

---

### Description

Create WEDECO metafile data

### Usage

```
create_wedeco_metafile(raw_data_file)
```

### Arguments

| | |
|---|---|
| raw_data_file | file path to raw data which should be used for as template for meta file creation |

### Value

data.frame with meta data file structure

---

## download_nextcloud_files *Helper Function: Download Nextcloud Files from a Directory*

---

### Description

Helper Function: Download Nextcloud Files from a Directory

### Usage

```
download_nextcloud_files(
  dir_cloud,
  dir_local,
  file_pattern = "Project\\.xls$"
)
```

### Arguments

| | |
|---|---|
| dir_cloud | directory on Nextcloud |
| dir_local | directory on local computer. If not existing it will be created |
| file_pattern | file pattern to be used as download filter (default: "Project\.xls$") |

### Value

downloads all files from cloud into local folder fullfilling file_pattern and returns the `file`, i.e. filename

### Examples

```
## Not run:
#1 Open RStudio and run usethis::edit_r_environ()
#2 In the opened window add the required environment variables
### NEXTCLOUD_URL = "https://<replace-with-nextcloud-cloud-url>"
### NEXTCLOUD_USER = "<your-nextcloud-username>" # your username
### NEXTCLOUD_PASSWORD = "your-nextcloud-app-password" ### see details below
#3 For creating <your-nextcloud-app-password>:
#3.1 go to: https://replace-with-nextcloud-url/index.php/settings/user/security
#3.2 scroll down to create new app password
#3.3 select a name e.g. r-script and copy the token and replace your-nextcloud-app-password
#4 Finally you need to restart Rstudio and proceed with the code below:
paths_list <- list(site_code = "Pilot_A",
common_path = "ultimate/raw_data_pilots/<site_code>/data",
dir_cloud = "projects/<common_path>",
dir_local = "C:/kwb/projects/<common_path>")

paths <- kwb.utils::resolve(paths_list)
download_nextcloud_files(dir_cloud = paths$dir_cloud,
dir_local = paths$dir_local,
file_pattern = "Project\\.xls$"
)

## End(Not run)
```

---

## dygraph_add_limits *Dygraph: add (multiple) horizontal lines to plot*

---

### Description

Dygraph: add (multiple) horizontal lines to plot

### Usage

```
dygraph_add_limits(
  dygraph,
  limits_df,
```

```
  label_loc = "left",
  col_limits = "ParameterThreshold",
  col_label = "label",
  ...
)
```

## Arguments

| | |
|---|---|
| dygraph | a dygraph object where (possibly) multiple horizontal lines should be added |
| limits_df | dataframe containing the limits information to be added to the dygraph (e.g. as retrieved by function get_thresholds()) |
| label_loc | Location for horizontal dygraph labels (left or right). (default: "left) |
| col_limits | column in limits_df containing the limits values (default: "ParameterThreshold") |
| col_label | column in limits_df containing the label values (default: "label") |
| ... | further arguments passed to dygraphs::dyLimit() |

## Value

add limits to existing dygraph object

---

# export_data *CSV data export in "wide" format*

---

## Description

CSV data export in "wide" format

## Usage

```
export_data(df_long, export_dir, dbg = TRUE)
```

## Arguments

| | |
|---|---|
| df_long | data frame in long format (as retrieved by kwb.pilot::group_datetime) |
| export_dir | path to export directory |
| dbg | debug messages (default: TRUE) |

## Value

tranforms data in "long" into "wide" format and writes into CSV file

## get_env_influxdb_ultimate *Helper Function: get influxdb config for Ultimate if defined defined*

### Description

Helper Function: get influxdb config for Ultimate if defined defined

### Usage

```
get_env_influxdb_ultimate()
```

### Value

list with influxdb config

## get_monthly_data_from_calendarweeks *Helper function for Berlin-S: get all calendar week files for monthy*

### Description

Helper function for Berlin-S: get all calendar week files for monthy

### Usage

```
get_monthly_data_from_calendarweeks(year_month)
```

### Arguments

| | |
|---|---|
| year_month | month to be imported (e.g. 2017-04') |

### Value

character vector with operational filenames with all calendar weeks that need to be imported for Berlin Schoenerlinde

## get_monthly_periods *Get monthly periods*

### Description

Get monthly periods

### Usage

```
get_monthly_periods(
  year_month_start = "2017-06",
  year_month_end = format(Sys.Date(), "%Y-%m"),
  tz = "CET"
)
```

### Arguments

| | |
|---|---|
| year_month_start | start year month (default: '2017-06') |
| year_month_end | end year month (default: current month) |
| tz | (default: 'CET') |

### Value

dataframe with monthly periods

## get_pivot_data *InfluxDB: Get Pivot Data from ultimate_mean_ bucket*

### Description

InfluxDB: Get Pivot Data from ultimate_mean_ bucket

### Usage

```
get_pivot_data(
  agg_interval = "1d",
  date_start = "2021-07-05",
  date_stop = Sys.Date()
)
```

### Arguments

| | |
|---|---|
| agg_interval | aggregation interval (default: 1h) |
| date_start | default: "2021-07-05" |
| date_stop | default: Sys.Date() |

### Value

pivot data for desired period and aggregation

---

`get_rawfilespaths_for_month` *Berlin-Tiefwerder: get rawfilepaths for months*

---

### Description

Berlin-Tiefwerder: get rawfilepaths for months

### Usage

```
get_rawfilespaths_for_month(
  monthly_period = get_monthly_periods()[1, ],
  raw_data_dir = package_file("shiny/berlin_t/data/operation"),
  max_offset_days = 7
)
```

### Arguments

| | |
|---|---|
| `monthly_period` | one row of data.frame as retrieved by function first row of get_monthly_periods(), i.e. year month is (default: '2017-06') |
| `raw_data_dir` | directory with operational raw data files for Berlin Tiefwerder (default: `kwb.pilot:::package_file("sh` |
| `max_offset_days` | number of days in previous/next month to look for beginning/ ending of month (default: 7) |

### Value

dataframe with monthly periods

---

`get_thresholds` *Get thresholds for analytics/operational parameters*

---

### Description

Get thresholds for analytics/operational parameters

### Usage

```
get_thresholds(file = package_file("shiny/haridwar/data/thresholds.csv"))
```

### Arguments

| | |
|---|---|
| `file` | path to csv file with thresholds for Haridwar site (default: `kwb.pilot:::package_file("shiny/haridwa` |

### Value

returns data frame thresholds for operational/analytical parameters

---

`get_valid_timezones` *Timezone: get valid time zones from Wikipedia*

---

### Description

Timezone: get valid time zones from Wikipedia

### Usage

```
get_valid_timezones()
```

### Value

returns data frame valid time zones (column: TZ.) from Wikipedia

### References

Check possible "tz" arguments in column "TZ*" of table https://en.wikipedia.org/wiki/List_of_tz_database_time_zones for more details.

---

`group_datetime` *Group DateTime by user defined period (year, month, day, hour, minute)*

---

### Description

Group DateTime by user defined period (year, month, day, hour, minute)

### Usage

```
group_datetime(
  df,
  by = 600,
  fun = "stats::median",
  col_datetime = "DateTime",
  col_datatype = "DataType",
  dbg = TRUE
)
```

### Arguments

| | |
|---|---|
| df | a data frame as retrieved by import_data_haridwar() |
| by | an aggregation time step in seconds (default: 600 seconds) for intra- day aggregation or "day", "month" or "year" for longer time spans |
| fun | function to be used for grouping measurement data of column ParameterValue (default: stats::median) (default: kwb.pilot:::package_file("shiny/haridwar/.my.cnf")) |
| col_datetime | column name of datetime column (default: DateTime) |
| col_datatype | column name of data type column (default: DataType) |
| dbg | print debug information |

### Value

returns data frame with data aggregated according to user defined aggregation time step

---

## import_analytics_basel

*Imports analytical data for Basel (without metadata)*

---

### Description

Imports analytical data for Basel (without metadata)

### Usage

```
import_analytics_basel(csv_dir = package_file("shiny/basel/data/analytics"))
```

### Arguments

| | |
|---|---|
| csv_dir | Define directory with raw analytical data in CSV (.csv) format to be imported (default: sema.pilot:::package_file("shiny/basel/data/analytics")) |

### Value

returns data frame with imported raw analytics data

## import_analytics_meta_basel

*Imports analytical data for Basel (with metadata for both sites at once, i.e. "rhein" and "wiese")*

### Description

Imports analytical data for Basel (with metadata for both sites at once, i.e. "rhein" and "wiese")

### Usage

```
import_analytics_meta_basel(
  analytics_dir = package_file("shiny/basel/data/analytics"),
  meta_site_path = package_file("shiny/basel/data/metadata/meta_site.csv"),
 meta_parameter_path = package_file("shiny/basel/data/metadata/meta_parameter.csv")
)
```

### Arguments

| | |
|---|---|
| analytics_dir | Define directory with raw analytical data in CSV (.csv) format to be imported (default: sema.pilot:::package_file("shiny/basel/data/analytics")) |
| meta_site_path | Define path of "meta_site.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/metada |
| meta_parameter_path | Define path of "meta_parameter.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/ |

### Value

data.frame with analytics data for Basel sites including metadata

## import_data_basel

*Imports operational & analytical data for Basel (with metadata for both sites at once, i.e. "rhein" and "wiese")*

### Description

Imports operational & analytical data for Basel (with metadata for both sites at once, i.e. "rhein" and "wiese")

## Usage

```
import_data_basel(
  analytics_dir = package_file("shiny/basel/data/analytics"),
  raw_dir_rhein = package_file("shiny/basel/data/operation/rhein"),
  raw_dir_wiese = package_file("shiny/basel/data/operation/wiese"),
  meta_online_path = package_file("shiny/basel/data/metadata/meta_online.csv"),
 meta_parameter_path = package_file("shiny/basel/data/metadata/meta_parameter.csv"),
  meta_site_path = package_file("shiny/basel/data/metadata/meta_site.csv")
)
```

## Arguments

| | |
|---|---|
| analytics_dir | Define directory with raw analytical data in CSV (.csv) format to be imported (default: sema.pilot:::package_file("shiny/basel/data/analytics")) |
| raw_dir_rhein | Define directory for site "rhein" with raw data in EXCEL spreadsheet format (.xlsx) to be imported (default: sema.pilot:::package_file("shiny/basel/data/operation/rhein")) |
| raw_dir_wiese | Define directory for site "rhein" with raw data in EXCEL spreadsheet format (.xlsx) to be imported (default: sema.pilot:::package_file("shiny/basel/data/operation/wiese")) |
| meta_online_path | path to file containing metadata for online data (default: sema.pilot:::package_file("shiny/basel/data/metac |
| meta_parameter_path | Define path of "meta_parameter.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/ |
| meta_site_path | Define path of "meta_site.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/metada |

## Value

data.frame with analytical & operational data for Basel

---

# import_data_berlin_f *Import data for Berlin Friedrichshagen*

---

## Description

Import data for Berlin Friedrichshagen

## Usage

```
import_data_berlin_f(
 raw_data_files = fs::dir_ls(package_file("shiny/berlin_f/data/raw/online_data"),
    recurse = TRUE, regexp = "^[^~].*\\.xlsx$"),

  meta_file_path = package_file("shiny/berlin_f/data/raw/online_data/parameter_unit_metadata.csv")
)
```

### Arguments

| | |
|---|---|
| raw_data_files | vector with full path to operational raw data files that allows to limit import to specific files (default: fs::dir_ls(package_file("shiny/berlin_f/data/raw/online_data""), recurse = TRUE, regexp = "^[^~].*\.xlsx$")). |
| meta_file_path | path to metadata file (default: kwb.pilot:::package_file(file.path("shiny/berlin_f/data/raw/online_data", "parameter_site_metadata.csv")) |

### Value

data.frame with imported operational data (analytics´data to be added as soon as available)

---

## import_data_berlin_s *Import data for Berlin Schoenerlinde*

---

### Description

Import data for Berlin Schoenerlinde

### Usage

```
import_data_berlin_s(
  raw_data_dir = package_file("shiny/berlin_s/data/operation"),
  raw_data_files = NULL,
 meta_file_path = package_file("shiny/berlin_s/data/parameter_site_metadata.csv")
)
```

### Arguments

| | |
|---|---|
| raw_data_dir | path of directory containing WEDECO CSV files (default: kwb.pilot:::package_file("shiny/berlin_s/data/c |
| raw_data_files | vector with full path to operational raw data files that allows to limit import to specific files (default: NULL). If specified parameter "raw_data_dir" will not be used |
| meta_file_path | path to metadata file (default: kwb.pilot:::package_file("shiny/berlin_s/data/parameter_site_metadata.csv' |

### Value

list with "df": data.frame with imported operational data (analytics data to be added as soon as available) and "added_data_points": number of added data points in case of existing fst file was updated with new operational data

## import_data_berlin_t *Import data for Berlin Tiefwerder*

### Description

Import data for Berlin Tiefwerder

### Usage

```
import_data_berlin_t(
  raw_data_dir = package_file("shiny/berlin_t/data/operation"),
  raw_data_files = NULL,
  analytics_path = package_file("shiny/berlin_t/data/analytics.xlsx"),
 meta_file_path = package_file("shiny/berlin_t/data/parameter_site_metadata.csv")
)
```

### Arguments

| | |
|---|---|
| raw_data_dir | path of directory containing PENTAIR xls files (default: kwb.pilot:::package_file("shiny/berlin_t/data/ope |
| raw_data_files | vector with full path to operational raw data files that allows to limit import to specific files (default: NULL). If specified parameter "raw_data_dir" will not be used |
| analytics_path | full path to lab data EXCEL file in xlsx format (default: kwb.pilot:::package_file("shiny/berlin_t/data/anal |
| meta_file_path | path to metadata file (default: kwb.pilot:::package_file("shiny/berlin_t/data/parameter_site_metadata.csv" |

### Value

data.frame with imported operational data (analytics´data to be added as soon as available)

## import_data_haridwar *Imports Haridwar data*

### Description

Imports Haridwar data

### Usage

```
import_data_haridwar(
  analytics_path = package_file("shiny/haridwar/data/analytics.xlsx"),
  operation_mySQL_conf = package_file("shiny/haridwar/.my.cnf"),
 operation_meta_path = package_file("shiny/haridwar/data/operation_parameters.csv"),
  excludedSheets = c("Parameters", "Location", "Sites", "#Summary",
   "Site_and_Parameter", "Observations", "dP", "ORP", "Flow", "Current_Voltage",
    "As_total_Arsenator"),
```

```
  skip = 69,
  debug = TRUE
)
```

## Arguments

| | |
|---|---|
| analytics_path | Define path of analytics EXCEL spreadsheet to be imported (default: kwb.pilot:::package_file("shiny/hari |
| operation_mySQL_conf | column name pattern for identifying raw data (default: kwb.pilot:::package_file("shiny/haridwar/.my.cnf" |
| operation_meta_path | path to table with meta data for operational parameters (default: kwb.pilot:::package_file("shiny/haridwar/ |
| excludedSheets | all sheets, which are not listed here will be imported as lab data sheets (default: c("Parameters", "Location", "Sites", "#Summary", "Site_and_Parameter", "Observations", "dP", "ORP", "Flow", "Current_Voltage", "As_total_Arsenator")) |
| skip | number of rows to skip for each lab data sheet (default: 69), i.e. for all sheets which are not explictly excluded with parameter "excludedSheets" |
| debug | if TRUE print debug messages (default: TRUE) |

## Value

returns data frame with Haridwar raw data (operation & analytics)

---

# import_lab_data_berlin_t

*BerlinTiefwerder: import lab data*

---

## Description

BerlinTiefwerder: import lab data

## Usage

```
import_lab_data_berlin_t(
  xlsx_path = package_file("shiny/berlin_t/data/analytics.xlsx")
)
```

## Arguments

| | |
|---|---|
| xlsx_path | full path to lab data EXCEL file in xlsx format (default: kwb.pilot:::package_file("shiny/berlin_t/data/anal |

## Value

a list of imported lab data for Berlin-Tiefwerder

`import_operation` *Imports operational data*

## Description

Imports operational data

## Usage

```
import_operation(mysql_conf = file.path(getwd(), ".my.cnf"))
```

## Arguments

| | |
|---|---|
| `mysql_conf` | path to the MySQL configuration file |

## Value

returns data frame operational data from MySQL db

`import_operation_basel` *Imports operational data for Basel (without metadata and only for one site at once, e.g. "rhein" or "wiese")*

## Description

Imports operational data for Basel (without metadata and only for one site at once, e.g. "rhein" or "wiese")

## Usage

```
import_operation_basel(
  xlsx_dir = package_file("shiny/basel/data/operation/wiese")
)
```

## Arguments

| | |
|---|---|
| `xlsx_dir` | Define directory with raw data in EXCEL spreadsheet (.xlsx) to be imported (default: sema.pilot:::package_file("shiny/basel/data/operation/wiese")) |

## Value

returns data frame with imported raw operational data

`import_operation_meta_basel` *Imports operational data for Basel (with metadata for both sites at once, i.e. "rhein" and "wiese")*

## Description

Imports operational data for Basel (with metadata for both sites at once, i.e. "rhein" and "wiese")

## Usage

```
import_operation_meta_basel(
  raw_dir_rhein = package_file("shiny/basel/data/operation/rhein"),
  raw_dir_wiese = package_file("shiny/basel/data/operation/wiese"),
  meta_online_path = package_file("shiny/basel/data/metadata/meta_online.csv"),
  meta_site_path = package_file("shiny/basel/data/metadata/meta_site.csv"),
 meta_parameter_path = package_file("shiny/basel/data/metadata/meta_parameter.csv")
)
```

## Arguments

| | |
|---|---|
| `raw_dir_rhein` | Define directory for site "rhein" with raw data in EXCEL spreadsheet format (.xlsx) to be imported (default: sema.pilot:::package_file("shiny/basel/data/operation/rhein")) |
| `raw_dir_wiese` | Define directory for site "rhein" with raw data in EXCEL spreadsheet format (.xlsx) to be imported (default: sema.pilot:::package_file("shiny/basel/data/operation/wiese")) |
| `meta_online_path` | path to file containing metadata for online data (default: sema.pilot:::package_file("shiny/basel/data/metac |
| `meta_site_path` | Define path of "meta_site.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/metada |
| `meta_parameter_path` | Define path of "meta_parameter.csv" to be imported (default: sema.pilot:::package_file("shiny/basel/data/ |

## Value

returns data frame with imported raw operational data with metadata for both sites (i.e."rhein" and "wiese")

data.frame with operational data for Basel sites including metadata

`import_sheets` *Imports multiple analytics sheets from an EXCEL spreadsheet*

## Description

Imports multiple analytics sheets from an EXCEL spreadsheet

## Usage

```
import_sheets(
  xlsPath,
  sheets_analytics,
  sheet_parameters = "Parameters",
  sheet_sites = "Sites",
  sheet_location = "Location",
  col_rawData_pattern = "raw",
  col_ignore_pattern = "mean|empty|X_|RX|not_used",
  skip = 69,
  tz_org = NULL,
  tz_export = "UTC",
  dbg = TRUE
)
```

## Arguments

| | |
|---|---|
| `xlsPath` | path to xls file with analytics data |
| `sheets_analytics` | a character vector with the names of the sheets with analytics data (check with: readxl::excel_sheets(xlsPath)) |
| `sheet_parameters` | sheet name containing parameter metadata (default: "Parameters") |
| `sheet_sites` | sheet name containing sites metadata (default: "Sites") |
| `sheet_location` | sheet name containing location metadata (default: "Location") |
| `col_rawData_pattern` | specify pattenr of columns containing raw data (default: "raw") |
| `col_ignore_pattern` | specify pattern of columns that should be ignored of importing (default: "mean\|empty\|X_\|RX\|not_used") |
| `skip` | number of rows in sheet to skip (default: 69), |
| `tz_org` | specify timezone of samples (default: "UTC") |
| `tz_export` | specify timezone for data export (default: "UTC") |
| `dbg` | print debug messages (default: TRUE) |

## Value

returns data frame with normalised analytics data in list form

## load_fst_data *Load fst data for shiny app*

### Description

Load fst data for shiny app

### Usage

```
load_fst_data(fst_dir)
```

### Arguments

| | |
|---|---|
| fst_dir | directory of fst files to be loaded |

## long_to_wide *Helper function: transform "long" to "wide"*

### Description

Helper function: transform "long" to "wide"

### Usage

```
long_to_wide(df)
```

### Arguments

| | |
|---|---|
| df | data frame in long format (as retrieved by kwb.pilot::group_datetime) |

## merge_and_export_fst *Helper function: merge and export fst files into main shiny data folder*

### Description

Helper function: merge and export fst files into main shiny data folder

### Usage

```
merge_and_export_fst(
  time_pattern = NULL,
  compression = 100,
  import_dir = package_file("shiny/berlin_t/data/fst"),
  export_dir = package_file("shiny/berlin_t/data")
)
```

### Arguments

| | |
|---|---|
| time_pattern | optional pattern to filter months to be imported (default: NULL), for using it do e.g. "2017-06\|2017-07" or c("2017-06", "2017-07") |
| compression | compression for fst export (default: 100) |
| import_dir | directory with fst files or subdirs to be imported (default: kwb.pilot:::package_file("shiny/berlin_t/data/fst' |
| export_dir | directory with fst directory for export (default: kwb.pilot:::package_file("shiny/berlin_t/data")) |

### Value

imports multiple fst files and exports them to be used for app

---

## move_nextcloud_files *Move Nextcloud Files*

---

### Description

Move Nextcloud Files

### Usage

```
move_nextcloud_files(paths_df, overwrite = FALSE, dbg = TRUE)
```

### Arguments

| | |
|---|---|
| paths_df | dataframe with columns raw and imported with full path to source and target file |
| overwrite | overwrite files if these exist in target directory (default: TRUE) |
| dbg | debug messages (default: TRUE) |

### Value

moves file to target file

`normalised_permeate_flow` *Calculate normalised permeate flow*

## Description

Calculate normalised permeate flow

## Usage

```
normalised_permeate_flow(
  tempFeed,
  conLoop,
  vfrPerm,
  vfrLoop,
  vfrFeed,
  prePerm,
  preProc,
  preConc,
  nwp0 = 1.429162,
  vfrPerm0 = 800
)
```

## Arguments

| | |
|---|---|
| `tempFeed` | tempFeed |
| `conLoop` | conLoop |
| `vfrPerm` | vfrPerm |
| `vfrLoop` | vfrLoop |
| `vfrFeed` | vfrFeed |
| `prePerm` | prePerm |
| `preProc` | preProc |
| `preConc` | preConc |
| `nwp0` | nwp0 |
| `vfrPerm0` | vfrPerm0 |

## Value

nwpt

## plot_analytics *Plot analytics data (in PDF)*

### Description

Plot analytics data (in PDF)

### Usage

```
plot_analytics(df)
```

### Arguments

| | |
|---|---|
| df | dataframe as retrieved by import_sheets() |

### Value

creates new subdirectory "/report" in current working directory and stores pdf plots there

## plot_calculated_operational_timeseries *Plot calculate operational time series*

### Description

Plot calculate operational time series

### Usage

```
plot_calculated_operational_timeseries(df)
```

### Arguments

| | |
|---|---|
| df | a data frame as retrieved by calculate_operational_parameters() |

### Value

plots time series for calculated operational parameters

### Examples

```
## Not run:
haridwar_raw_list <- import_data_haridwar()
myDat <- calculate_operational_parameters(df = haridwar_raw_list)
plot_calculated_operational_timeseries(myDat)

## End(Not run)
```

## plot_data *Export interactive HTML plot with "plotly"*

### Description

Export interactive HTML plot with "plotly"

### Usage

```
plot_data(df_long, export_dir, dbg = TRUE)
```

### Arguments

| | |
|---|---|
| df_long | data frame in long format (as retrieved by kwb.pilot::group_datetime) |
| export_dir | path to export directory |
| dbg | debug messages (default: TRUE) |

### Value

interactive HTML plots in subdirectory "<export_dir>/plots/"

## read_fst *Wrapper for fst::read.fst to read DateTime column in POSIXct format*

### Description

Wrapper for fst::read.fst to read DateTime column in POSIXct format

### Usage

```
read_fst(path, tz = "CET", col_datetime = "DateTime", ...)
```

### Arguments

| | |
|---|---|
| path | path to fst file |
| tz | timezone of DateTime to be imported (default: "CET") |
| col_datetime | column name containing numeric values in nanoseconds since 1970-01-01 (default: "DateTime") |
| ... | further arguments passed to fst::read.fst |

### Value

data.frame with formatting of DateTime column POSIXct

read_mbr4 *Read MBR4.0 data combining latest and archived data*

## Description

Download latest data as 'tsv' from Martin Systems Webportal and combine with archived ('tsv') on Nextcloud'

## Usage

```
read_mbr4(
  latest_url = Sys.getenv("MBR40_URL"),
  archived_file = "MBR_export_",
  archived_dir = "projects/MBR4.0/Exchange/Rohdaten/Online_export",
  archived_url = Sys.getenv("NEXTCLOUD_URL"),
  archived_user = Sys.getenv("NEXTCLOUD_USER"),
  archived_pw = Sys.getenv("NEXTCLOUD_USER"),
  target_dir = tempdir(),
  dbg = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| latest_url | url to download latest .tsv file (default: Sys.getenv("MBR40_URL"), please use run edit_r_environ and define a key value pair "MBR40_URL" = "download-url-martin-systems") so that this function works automatically |
| archived_file | file name/pattern of XLSX file (default: "MBR_export_") |
| archived_dir | directory on Nextcloud containing file (default: "projects/MBR4.0/Exchange/Rohdaten/Online_export") |
| archived_url | url of Nextcloud (default: Sys.getenv("NEXTCLOUD_URL")) |
| archived_user | username of Nextcloud (default: Sys.getenv("NEXTCLOUD_USER")) |
| archived_pw | password of Nextcloud (default: Sys.getenv("NEXTCLOUD_USER")) |
| target_dir | directory to download data (default: tempdir()) |
| dbg | print debug messages (default: FALSE) |
| ... | additional arguments passed to read_tsv |

## Value

tibble with imported MBR4.0 xlsx data (archived on Nextcloud since start of operation)

## Examples

```
## Not run:
mbr4_data <- read_mbr4()
str(mbr4_data)

## End(Not run)
```

`read_mbr4_archived` *Read MBR4.0 archived data from Nextcloud*

## Description

Read MBR4.0 archived data from Nextcloud

## Usage

```
read_mbr4_archived(
  file = "MBR_export_",
  dir = "projects/MBR4.0/Exchange/Rohdaten/Online_export",
  target_dir = tempdir(),
  locale = readr::locale(tz = "CET", decimal_mark = ",", grouping_mark = "."),
  col_types = readr::cols(.default = readr::col_double(), zustand =
    readr::col_character(), meldungen = readr::col_character(), Zeitstempel =
    readr::col_datetime(format = "%d.%m.%Y %H:%M")),
  url = Sys.getenv("NEXTCLOUD_URL"),
  user = Sys.getenv("NEXTCLOUD_USER"),
  pw = Sys.getenv("NEXTCLOUD_USER"),
  dbg = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `file` | file name/pattern of tsv file (default: "MBR_export_") |
| `dir` | directory on Nextcloud containing file (default: "projects/MBR4.0/Exchange/Rohdaten/Online_export") |
| `target_dir` | directory to download data (default: tempdir()) |
| `locale` | locale (default: `locale`(tz = "CET", decimal_mark = ",", grouping_mark = ".")) |
| `col_types` | col_types (default: `cols`( .default = `col_double`, zustand = `col_character`, meldungen = `col_character`, Zeitstempel = `col_datetime`(format = " ) |
| `url` | url of Nextcloud (default: Sys.getenv("NEXTCLOUD_URL")) |
| `user` | username of Nextcloud (default: Sys.getenv("NEXTCLOUD_USER")) |
| `pw` | password of Nextcloud (default: Sys.getenv("NEXTCLOUD_USER")) |
| `dbg` | print debug messages (default: FALSE) |
| `...` | additional arguments passed to read_tsv |

## Value

tibble with imported archived MBR4.0 xlsx data from Nextcloud

## Examples

```
if(check_env_nextcloud()) {
mbr4_data_archived <- read_mbr4_archived()
str(mbr4_data_archived)
}
```

---

`read_mbr4_latest` *Read MBR4.0 data from Martin Systems Webportal (As "tsv")*

---

## Description

Read MBR4.0 data from Martin Systems Webportal (As "tsv")

## Usage

```
read_mbr4_latest(
  url = Sys.getenv("MBR40_URL"),
  target_dir = tempdir(),
  locale = readr::locale(tz = "CET", decimal_mark = ".", grouping_mark = ","),
  col_types = readr::cols(.default = readr::col_double(), zustand =
    readr::col_character(), meldungen = readr::col_character(), Zeitstempel =
    readr::col_datetime(format = "%Y-%m-%d %H:%M:%S")),
  dbg = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `url` | url to download site (default: Sys.getenv("MBR4.0_URL"), please use run edit_r_environ and define a key value pair "MBR40_URL" = "download-url-martin-systems") so that this function works automatically |
| `target_dir` | directory to download data (default: tempdir()) |
| `locale` | locale (default: locale(tz = "CET", decimal_mark = ".", grouping_mark = ",")) |
| `col_types` | col_types (default: cols( .default = col_double, zustand = col_character, meldungen = col_character, Zeitstempel = col_datetime(format = " ) |
| `dbg` | print debug messages (default: FALSE) |
| `...` | additional arguments passed to read_tsv |

## Value

tibble with imported MBR4.0 tsv data (~ last four weeks)

## Examples

```
url_mbr40 <- Sys.getenv("MBR40_URL")
if(url_mbr40 != "") {
mbr4_data_latest <- read_mbr4_latest(url = url_mbr40)
str(mbr4_data_latest)
}
```

---

`read_mbr4_tsv` *Read MBR4.0 tsv data*

---

## Description

Read MBR4.0 tsv data

## Usage

```
read_mbr4_tsv(
  path,
  locale = readr::locale(tz = "CET", decimal_mark = ",", grouping_mark = "."),
  col_types = readr::cols(.default = readr::col_double(), zustand =
    readr::col_character(), meldungen = readr::col_character(), Zeitstempel =
    readr::col_datetime(format = "%d.%m.%Y %H:%M")),
  dbg = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `path` | path to tsv file to be imported |
| `locale` | locale (default: `locale`(tz = "CET", decimal_mark = ",", grouping_mark = ".")) |
| `col_types` | col_types (default: `cols`( .default = `col_double`, zustand = `col_character`, meldungen = `col_character`, Zeitstempel = `col_datetime`(format = " ) |
| `dbg` | print debug messages (default: FALSE) |
| `...` | additional arguments passed to read_tsv |

## Value

Reads MBR4.0 tsv data

## read_pentair_data *Read PENTAIR operational data*

### Description

Read PENTAIR operational data

### Usage

```
read_pentair_data(
  raw_data_dir = package_file("shiny/berlin_t/data/operation"),
  raw_data_files = NULL,
 meta_file_path = package_file("shiny/berlin_t/data/parameter_site_metadata.csv"),
  locale = readr::locale(tz = "CET"),
  col_types = readr::cols()
)
```

### Arguments

| | |
|---|---|
| raw_data_dir | path of directory containing PENTAIR xls files (default: kwb.pilot:::package_file("shiny/berlin_t/data/ope |
| raw_data_files | vector with full path to operational raw data files that allows to limit import to specific files (default: NULL). If specified parameter "raw_data_dir" will not be used |
| meta_file_path | path to metadata file (default: kwb.pilot:::package_file("shiny/berlin_t/data/parameter_site_metadata.csv" |
| locale | locale (default: locale(tz = "CET")) |
| col_types | col_types (default: cols) |

### Value

data.frame with imported PENTAIR operational data

## read_wedeco_data *Import WEDECO raw data*

### Description

Import WEDECO raw data

### Usage

```
read_wedeco_data(
  raw_data_dir = package_file("shiny/berlin_s/data/operation"),
  raw_data_files = NULL,
 meta_file_path = package_file("shiny/berlin_s/data/parameter_site_metadata.csv")
)
```

### Arguments

| | |
|---|---|
| raw_data_dir | path to raw data directory |
| raw_data_files | vector with full path to operational raw data files that allows to limit import to specific files (default: NULL). If specified parameter "raw_data_dir" will not be used |
| meta_file_path | path to meta data file |

---

## read_weintek *Read Weintek data from single file*

---

### Description

Read Weintek data from single file

### Usage

```
read_weintek(path, tz = "CET", dbg = TRUE)
```

### Arguments

| | |
|---|---|
| path | path to Weintek file |
| tz | time zone (default: CET) the measurements are taken (passed to function kwb.pilot:::set_timezone()) |
| dbg | debug (default: TRUE) |

### Value

data frame with Weintek raw data

---

## read_weintek_batch *Read Weintek data from multiple files*

---

### Description

Read Weintek data from multiple files

### Usage

```
read_weintek_batch(files, tz = "CET", dbg = TRUE)
```

### Arguments

| | |
|---|---|
| files | path to Weintek files |
| tz | time zone (default: CET) the measurements are taken (passed to function kwb.pilot:::set_timezone()) |
| dbg | debug (default: TRUE) |

### Value

data frame with Weintek raw data

---

`remove_duplicates` *Remove duplicates in data.frame*

---

### Description

Remove duplicates in data.frame

### Usage

```
remove_duplicates(df, col_names = names(df))
```

### Arguments

| | |
|---|---|
| `df` | data.frame to be checked for duplicates |
| `col_names` | column names to be used for duplicate checking (default: names(df)). can be defined by providing: c("col_name1", "col_name2") |

### Value

data.frame without duplicates

---

`report_config_template`
*Report config: generate template*

---

### Description

Report config: generate template

### Usage

```
report_config_template(
  df = NULL,
  temporal_aggregation = "raw",
  output_timezone = "UTC"
)
```

### Arguments

| | |
|---|---|
| df | a dataframe as retrieved by import_data_haridwar() |
| temporal_aggregation | Set the following values if data should be summarised to e.g. 10 minutes (600) or hourly (3600), daily ("day") or monthly ("month") median values (default: "raw") |
| output_timezone | into which timezone should the data be converted for the report? (default: "UTC") |

### Value

default list for report configuration template

---

## report_config_to_txt *Report config: saves config to text file*

---

### Description

Report config: saves config to text file

### Usage

```
report_config_to_txt(config_list, output_file = "report_config.txt")
```

### Arguments

| | |
|---|---|
| config_list | a report configuration list e.g. as retrieved by report_config_template() |
| output_file | absolute or relative path of where to save output file (default: "report_config.txt") |

### Value

saves report configuration list as text file

### Examples

```
## Not run:
### Creates a configuration template
config <- report_config_template()
### Saves list config in text
report_config_to_txt(
  config_list = config,
  output_file = "report_config.txt"
)

## End(Not run)
```

## report_txt_to_config *Report config: imports text file to list*

### Description

Report config: imports text file to list

### Usage

```
report_txt_to_config(config_txt = "report_config.txt")
```

### Arguments

| | |
|---|---|
| config_txt | path to report configuration text file created by a report configuration list e.g. as retrieved by function report_config_to_txt() |

### Value

saves report configuration list as text file

### Examples

```
## Not run:
### Creates a configuration template
config <- report_config_template()
### Saves list config in text
report_config_to_txt(config_list = config, output_file = "report_config.txt")
### Reads config list from text file to
config_imported <- report_txt_to_config(config_txt = "report_config.txt")
### Check whether both are identical
identical(x = config, y = config_imported)

## End(Not run)
```

## run_app *Runs Shiny app for an AQUANES site*

### Description

Runs Shiny app for an AQUANES site

## Usage

```
run_app(
  siteName = "haridwar",
  use_live_data = FALSE,
  mySQL_conf = NULL,
  launch.browser = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| siteName | site name for shiny app (default: "haridwar") |
| use_live_data | should live data be used (default: FALSE) |
| mySQL_conf | file path to mySQL config file (.my.cnf). Only used if parameter use_live_data is TRUE and there is no .my.cnf in the app folder for the selected site (default: NULL) |
| launch.browser | If true, the system's default web browser will be launched automatically after the app is started (default: TRUE) |
| ... | further arguments passed to shiny::runApp() |

---

# set_timezone *Timezone set: sets a user defined time zone*

---

## Description

Timezone set: sets a user defined time zone

## Usage

```
set_timezone(df, tz = "UTC", col_datetime = "DateTime")
```

## Arguments

| | |
|---|---|
| df | a dataframe containing a datetime column |
| tz | timezone (default: "UTC") |
| col_datetime | name of the datetime column (default: "DateTime") |

## Value

returns data frame with specified time zone

## References

Check possible "tz" arguments in column "TZ*" of table https://en.wikipedia.org/wiki/List_of_tz_database_time_zones for more details.

## shiny_file *Path to Shiny File in Package*

### Description

Path to Shiny File in Package

### Usage

```
shiny_file(...)
```

### Arguments

| | |
|---|---|
| ... | relative path to file in "shiny" folder of R package "kwb.pilot" |

### Value

path to file (if existing)

### Examples

```
shiny_file()
```

## tidy_mbr4_data *MBR 4.0 Data Tidy*

### Description

MBR 4.0 Data Tidy

### Usage

```
tidy_mbr4_data(
  mbr4_data,
  path_metadata = kwb.pilot:::shiny_file("mbr4.0/data/metadata.csv")
)
```

### Arguments

| | |
|---|---|
| mbr4_data | tibble as retrieved by read_mbr4 |
| path_metadata | path to metadata file (default: kwb.pilot:::shiny_file("mbr4.0/data/metadata.csv")) |

### Value

tidy MBR 4 data in long format

## Examples

```
## Not run:
mbr4_data <- read_mbr4()
mbr4_data_tidy <- kwb.pilot::tidy_mbr4_data(mbr4_data)

## End(Not run)
```

---

`write_aggr_to_influxdb` *InfluxDB: write aggregated time series to Ultimate target bucket*

---

## Description

InfluxDB: write aggregated time series to Ultimate target bucket

## Usage

```
write_aggr_to_influxdb(
  start,
  end,
  agg_interval = "1h",
  agg_function = "mean",
  bucket_source = "ultimate",
  bucket_target = sprintf("%s_%s_%s", bucket_source, agg_function, agg_interval),
  bucket_org = "kwb"
)
```

## Arguments

| | |
|---|---|
| `start` | date start |
| `end` | date end |
| `agg_interval` | aggregation interval (default: "1h") |
| `agg_function` | aggregation function (default: "mean") |
| `bucket_source` | bucket source (default: "ultimate") |
| `bucket_target` | bucket target (default: <bucket_source_<agg_function>_<agg_interval>)) |
| `bucket_org` | bucket organisation (default: "kwb") |

## Value

writes aggregated time series to InfluxDB target bucket in loop

`write_aggr_to_influxdb_loop` *InfluxDB: write aggregated time series to Ultimate target bucket in loop*

## Description

wrapper for `write_aggr_to_influxdb`

## Usage

```
write_aggr_to_influxdb_loop(
  agg_interval = "1h",
  agg_function = "mean",
  bucket_source = "ultimate",
  bucket_target = sprintf("%s_%s_%s", bucket_source, agg_function, agg_interval),
  bucket_org = "kwb",
  date_start = "2021-07-05",
  date_end = Sys.Date(),
  hour_start = 0,
  hour_end = 12,
  max_days = 5
)
```

## Arguments

| | |
|---|---|
| `agg_interval` | aggregation interval (default: "1h") |
| `agg_function` | aggregation function (default: "mean") |
| `bucket_source` | bucket source (default: "ultimate") |
| `bucket_target` | bucket target (default: <bucket_source_<agg_function>_<agg_interval>)) |
| `bucket_org` | bucket organisation (default: "kwb") |
| `date_start` | date start (default: "2021-07-05") |
| `date_end` | date end (default: Sys.Date()) |
| `hour_start` | (default: 0) |
| `hour_end` | (default: 12) |
| `max_days` | maximum time period in days that should be sent within one query to influxdb (default: 5) |

## Value

writes aggregated time series to InfluxDB target bucket in loop

`write_to_influxdb` *InfluxDB: write to InfluxDB*

## Description

InfluxDB: write to InfluxDB

## Usage

```
write_to_influxdb(tsv_paths, paths, changed_only = TRUE, batch_size = 5000)
```

## Arguments

| | |
|---|---|
| `tsv_paths` | vector with tsv_paths with files to be imported by a modification of `read_pentair_data` |
| `paths` | paths list with elements `raw_data_dir` and `site_code` |
| `changed_only` | TRUE if only columns with changing data points within time series of provided tsv_path should be written to InfluxDB, otherwise FALSE (default: TRUE) |
| `batch_size` | number of data points that are written in one query (default: 5000) |

## Value

writes imported data to InfluxDB

`write_to_influxdb_loop`
*InfluxDB: write to InfluxDB in Loop*

## Description

wrapper function for `write_to_influxdb`

## Usage

```
write_to_influxdb_loop(
  tsv_paths,
  paths,
  changed_only = TRUE,
  max_tsv_files = 5,
  batch_size = 5000
)
```

## Arguments

| | |
|---|---|
| tsv_paths | vector with tsv_paths with files to be imported by write_to_influxdb which relies on read_pentair_data |
| paths | paths list with elements raw_data_dir and site_code |
| changed_only | TRUE if only columns with changing data points within time series of provided tsv_paths (limited by parameter max_tsv_files, i.e. changes between different tsv splits are not detected!) should be written to InfluxDB, otherwise FALSE (default: TRUE) |
| max_tsv_files | maximum number of tsv files to read at once (should be limited due to high RAM demand), default: 5 |
| batch_size | number of data points that are written in one query (default: 5000) |

## Value

writes imported data to InfluxDB in Loop

# Index