Package: watina (via r-universe)

August 10, 2024

Title Querying and Processing Data from the INBO Watina Database

Version 0.4.2

Description The R-package watina contains functions to query and process data from the Watina database at the Research Institute for Nature and Forest (INBO). This database primarily provides groundwater level and chemical data, mainly from natural areas in Flanders (Belgium).

License GPL-3

URL https://inbo.github.io/watina, https://github.com/inbo/watina

BugReports https://github.com/inbo/watina/issues

Depends R (>= 3.5.0)

Imports assertthat, dplyr, inbodb, lubridate, rlang, stringr, stats, tidyr

Suggests DBI, knitr, KSgeneral, purrr, rmarkdown, sf, tibble, tidyselect

Remotes inbo/inbodb

LazyData true

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

Repository https://inbo.r-universe.dev

RemoteUrl https://github.com/inbo/watina

RemoteRef HEAD

RemoteSha fe50ec280b38f2fce91fd5e3721ae78e6cdf0607

Contents

`as_points` *Convert a dataframe with X and Y coordinates to a geospatial points object*

Description

`as_points` is a convenience function which accepts as input a dataframe with X/Y coordinates (in meters), assumed to come from the coordinate reference system (CRS) 'Belge 1972 / Belgian Lambert 72' (EPSG 31370). It converts the dataframe into an `sf` points object in the same CRS.

Usage

```
as_points(df, xvar = "x", yvar = "y", remove = FALSE, warn_dupl = TRUE)
```

Arguments

`df`	A dataframe with X and Y coordinates in meters, assumed to be in the Belgian Lambert 72 CRS (EPSG-code 31370).
`xvar`	String. The X coordinate variable name. Defaults to `"x"`.
`yvar`	String. The Y coordinate variable name. Defaults to `"y"`.
`remove`	Logical. Should the X and Y coordinates be removed from the dataframe after conversion to a spatial object?
`warn_dupl`	Logical. Defaults to `TRUE`. Should the user be warned when duplicated coordinates are present in the result?

Details

As locations in Watina are typically defined by their X/Y coordinates, this function eases the conversion to spatial data. To later remove all spatial information from the result, you can use `sf::st_drop_geometry()`.

Value

An `sf` object of geometry type `POINT` with EPSG-code 31370 as coordinate reference system.

Examples

```
library(tibble)
mydata <-
  tibble(
    a = runif(5),
    x = rnorm(5, 155763, 5),
    y = rnorm(5, 132693, 5)
  )
as_points(mydata)
```

`cluster_locs` *Detect (spatial) groundwater well clusters*

Description

`cluster_locs()` accepts as input a dataframe with X/Y coordinates, or an `sf` object of geometry type `POINT`. The function adds an integer variable that defines cluster membership. The intention is to detect spatial groundwater well clusters; hence it uses a sensible method of spatial clustering and default euclidean distance to cut the cluster tree.

Usage

```
cluster_locs(
  input,
  max_dist = 2,
  output_var = "cluster_id",
  xvar = "x",
  yvar = "y"
)
```

Arguments

`input`	A dataframe with X/Y coordinates, or an `sf` object of geometry type `POINT`. A typical input dataframe is the collected output of `get_locs`.
`max_dist`	The maximum geospatial distance between two points to make them belong to the same cluster. The default value is sensible for many usecases, supposing *meter* is the unit of the coordinate reference system, as is the case for the 'Belge 1972 / Belgian Lambert 72' CRS (EPSG 31370).
`output_var`	Name of the new variable to be added to `input`.
`xvar`	String. The X coordinate variable name; only considered when `input` is a dataframe. Defaults to `"x"`.
`yvar`	String. The Y coordinate variable name; only considered when `input` is a dataframe. Defaults to `"y"`.

Details

The function performs agglomerative clustering with the *complete linkage* method. This way, the application of a tree cutoff (`max_dist`) means that each cluster is a collection of locations with a maximum distance - between any two locations of the cluster - not larger than the cutoff value. All locations that can be clustered under this condition, will be. Locations that can not be clustered receive a unique cluster value.

The function's code was partly inspired by unpublished code from Ivy Jansen.

Value

The original object with an extra variable added (by default: `cluster_id`) to define cluster membership.

Examples

```
library(dplyr)
set.seed(123456789)
mydata <-
  tibble(
    a = runif(10),
    x = rnorm(10, 155763, 2),
    y = rnorm(10, 132693, 2)
  )
cluster_locs(mydata) %>%
  arrange(cluster_id)
mydata %>%
  as_points(remove = TRUE) %>%
  cluster_locs %>%
  arrange(cluster_id)

## Not run:
watina <- connect_watina()

clusters <-
  get_locs(watina,
           area_codes = "KBR",
           collect = TRUE) %>%
  cluster_locs

# inspect result:
clusters %>%
  select(loc_code, x, y, cluster_id) %>%
  arrange(cluster_id)

# frequency of cluster sizes:
clusters %>%
  count(cluster_id) %>%
  pull(n) %>%
  table

# Disconnect:
```

```
dbDisconnect(watina)

## End(Not run)
```

`connect_watina` *Connect to the INBO Watina database*

Description

Returns a connection to the INBO **Watina** database. The function can only be used from within the INBO network.

Usage

```
connect_watina()
```

Details

Don't forget to disconnect at the end of your R-script using `dbDisconnect`!

Value

A `DBIConnection` object.

Examples

```
## Not run:
watina <- connect_watina()
# Do your stuff.
# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

`dbDisconnect` *Disconnect a database connection*

Description

This is a re-export of `inbodb::dbDisconnect()` (url).

`eval_chem` *Evaluate hydrochemical data per location*

Description

For a dataset as returned by `get_chem`, return summary statistics (data availability and/or numeric properties) of the specified hydrochemical variables, for each location.

Usage

```
eval_chem(
  data,
 chem_var = c("P-PO4", "N-NO3", "N-NO2", "N-NH4", "HCO3", "SO4", "Cl", "Na", "K", "Ca",
    "Mg", "Fe", "Mn", "Si", "Al", "CondF", "CondL", "pHF", "pHL"),
  type = c("avail", "num", "both"),
  uniformity_test = FALSE
)
```

Arguments

`data`	An object returned by `get_chem`.
`chem_var`	A character vector to select chemical variables for which statistics will be computed. To specify chemical variables, use the codes from the column `chem_variable` in `data`.
`type`	A string defining the requested type of summary statistics. See section 'Value'. Either: • `"avail"`: availability statistics (the default); • `"num"`: numeric summary statistics; • `"both"`: both types will be returned.
`uniformity_test`	Should the availability statistic `pval_uniform_totalspan` be added (see section 'Value')? Defaults to `FALSE` as this takes much more time to calculate than everything else.

Details

For the availability statistics, an extra level `"combined"` is added in the column `chem_variable` whenever the arguments `data` and `chem_var` imply more than one chemical variable to be investigated. This 'combined' level defines data availability for a water sample as the availability of data for **all** corresponding chemical variables.

Value

A tibble with variables `loc_code` (see `get_locs`), `chem_variable` (character; see Details) and summary statistics (*at the level of* `loc_code` *x* `chem_variable`) depending on the state of `type` (`type = "both"` combines both cases):

With `type = "avail"` (the default):

- nryears: number of calendar years for which the chemical variable is available (on one date at least)
- nrdates: number of dates at which the chemical variable is available
- firstdate: earliest date at which the chemical variable is available
- lastdate: latest date at which the chemical variable is available
- timespan_years: duration as years from the first calendar year (year of firstdate) to the last calendar year (year of lastdate) with data available
- timespan_totalspan_ratio: the ratio of timespan_years to the 'total span', which is the duration as years from the first to the last calendar year *for the whole of* data.
- nryears_totalspan_ratio: the ratio of nryears to the 'total span'
- pval_uniform_totalspan: Only returned with uniformity_test = TRUE. p-value of an exact, one-sample two-sided Kolmogorov-Smirnov test for the discrete uniform distribution of the calendar years *with data of the chemical variable available* within the series of consecutive calendar years defined by the 'total span' (see above). The smaller the p-value, the less uniform the years are spread within the total span. A perfectly uniform spread results in a p-value of 1.

With type = "num": summary statistics based on the chemical values, i.e. excluding the 'combined' level:

- val_min: minimum value
- val_pct10, val_pct25, val_pct50, val_pct75, val_pct90: percentiles
- val_max: maximum value
- val_range: range
- val_mean: mean
- val_geometric_mean: geometric mean. Only calculated when all values are strictly positive.
- unit
- prop_below_loq: the proportion of measurements below the limit of quantification (as derived from below_loq == TRUE), i.e. relative to the total number of measurements *for which* below_loq *is not* NA. This statistic neglects measurements with value NA for below_loq! If all measurements have value NA for below_loq, this statistic is set to NA.

See Also

Other functions to evaluate locations: eval_xg3_avail(), eval_xg3_series()

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "ZWA")
mydata <-
 mylocs %>%
 get_chem(watina, "1/1/2010")
mydata %>% arrange(loc_code, date, chem_variable)
```

```
mydata %>%
  pull(date) %>%
  lubridate::year(.) %>%
  (function(x) c(firstyear = min(x), lastyear = max(x)))
mydata %>%
  eval_chem(chem_var = c("P-PO4", "N-NO3", "N-NO2", "N-NH4")) %>%
  arrange(desc(loc_code))
mydata %>%
  eval_chem(chem_var = c("P-PO4", "N-NO3", "N-NO2", "N-NH4"),
            type = "both") %>%
  arrange(desc(loc_code)) %>%
  as.data.frame() %>%
  head(10)
mydata %>%
  eval_chem(chem_var = c("P-PO4", "N-NO3", "N-NO2", "N-NH4"),
            uniformity_test = TRUE) %>%
  arrange(desc(loc_code)) %>%
  select(loc_code, chem_variable, pval_uniform_totalspan)
# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

`eval_xg3_avail` *Evaluate the availability of XG3 values per location*

Description

For a dataset as returned by `get_xg3`, return for each location the first and last (hydro)year and the number of (hydro)years with available XG3 values of the specified type(s) (LG3, HG3 and/or VG3).

Usage

```
eval_xg3_avail(data, xg3_type = c("L", "H", "V"))
```

Arguments

`data`	An object returned by `get_xg3`.
`xg3_type`	Character vector of length 1, 2 or 3. Defines the types of XG3 which are taken from `data`. Specifies the 'X' in 'XG3': either `"L"`, `"H"` and/or `"V"`. Defaults to `"L"`.

Details

The column xg3_variable in the resulting tibble stands for the XG3 type + the vertical CRS (see get_xg3). xg3_variable is restricted to the requested XG3 types (LG3, HG3 and/or VG3) via the xg3_type argument, but adds an extra level "combined" whenever the combination of data (which may have both vertical CRSes) and xg3_type implies more than one requested variable. The "combined" level evaluates the combined presence of all selected XG3 variables at each location.

Value

A tibble with variables loc_code (see get_locs), xg3_variable (character; see Details), nryears, firstyear and lastyear.

See Also

Other functions to evaluate locations: eval_chem(), eval_xg3_series()

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "KAL")
mydata <-
 mylocs %>%
 get_xg3(watina, 2014)
mydata %>% arrange(loc_code, hydroyear)
eval_xg3_avail(mydata,
               xg3_type = c("L", "V"))
# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

---

eval_xg3_series *Identify and evaluate XG3 series per location*

Description

For a dataset as returned by get_xg3, determine for each location the available multi-year XG3 series and calculate summary statistics for each series. Note that 'years' in this context always refers to hydroyears.

Usage

```
eval_xg3_series(data, xg3_type = c("L", "H", "V"), max_gap, min_dur)
```

Arguments

data	An object returned by get_xg3.
xg3_type	Character vector of length 1, 2 or 3. Defines the types of XG3 which are taken from data. Specifies the 'X' in 'XG3': either "L", "H" and/or "V". Defaults to "L".
max_gap	A positive integer (can be zero). It is part of what the user defines as 'an XG3 series': the maximum allowed time gap between two consecutive XG3 values in a series, expressed as the number of years without XG3 value.
min_dur	A strictly positive integer. It is part of what the user defines as 'an XG3 series': the minimum required duration of an XG3 series, i.e. the time (expressed as years) from the first to the last year of the XG3 series.

Details

An XG3 series is a location-specific, multi-year series of LG3, HG3 and/or VG3 variables and is user-restricted by max_gap (maximum allowed number of empty years between 'series member' years) and min_dur (minimum required length of the series). Further, given a dataset of XG3 values per year and location, XG3 series are always constructed *as long as possible* given the aforementioned restrictions. For one location and XG3 variable, more than one such XG3 series may be available, which implies that those XG3 series are separated by more years than the value of max_gap.

The function returns summary statistics for the XG3 series that are available in the dataset. The XG3 series of each site are numbered as 'prefix_series1', 'prefix_series2' with the prefix being the value of xg3_variable.

The column xg3_variable in the resulting tibble stands for the XG3 type + the vertical CRS (see get_xg3) to which a series belongs. xg3_variable is restricted to the requested XG3 types (LG3, HG3 and/or VG3) via the xg3_type argument, but adds an extra level "combined" whenever the combination of data (which may have both vertical CRSes) and xg3_type implies more than one requested variable. This 'combined' level defines an XG3 series as an XG3 series where each 'member' year has **all** selected XG3 variables available.

Value

A tibble with variables:

- loc_code: see get_locs
- xg3_variable: character; see Details
- series: series ID, unique within loc_code
- ser_length: series duration (as years), i.e. from first to last year
- ser_nryears: number of years in the series for which the XG3 variable is available
- ser_rel_nryears: the fraction ser_nryears / ser_length,
- ser_firstyear: first year in the series with XG3 variable
- ser_lastyear: last year in the series with XG3 variable

- ser_pval_uniform: p-value of an exact, one-sample two-sided Kolmogorov-Smirnov test for the discrete uniform distribution of the member years within the XG3 series. The smaller the p-value, the less uniform the member years are spread within a series. A perfectly uniform spread results in a p-value of 1. Only with larger values of max_gap this p-value can get low.
- Summary statistics based on the XG3 values, i.e. excluding the 'combined' series:
  - ser_mean: mean XG3 value of the series, as meters.
  - ser_sd: standard deviation of the XG3 values of the series (an estimate of the superpopulation's standard deviation). As meters.
  - ser_se_6y: estimated standard error of the mean XG3 for a six-year period, applying finite population correction (i.e. for design-based estimation of this mean). Hence, ser_se_6y is zero when a series has no missing years. As meters. For series shorter than six years, the estimation is still regarding a six-year period, assuming the same sampling variance as in the short series.
  - ser_rel_sd_lcl: relative standard deviation of XG3 values, i.e. ser_sd / ser_mean. **This value is only calculated for** vert_crs = "local"**.**
  - ser_rel_se_6y_lcl: relative standard error of the mean XG3 for a six-year period, i.e. ser_se_6y / ser_mean. **This value is only calculated for** vert_crs = "local"**.**

See Also

extract_xg3_series

Other functions to evaluate locations: eval_chem(), eval_xg3_avail()

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "KAL")
mydata <-
 mylocs %>%
 get_xg3(watina, 1900)
mydata %>% arrange(loc_code, hydroyear)
mydata %>%
  eval_xg3_series(xg3_type = c("L", "V"),
                  max_gap = 2,
                  min_dur = 5)
# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

`extract_xg3_series` *Identify XG3 series per location*

Description

For a dataset as returned by `get_xg3`, determine for each location the available multi-year XG3 series. The function is called by `eval_xg3_series`. Contrary to `eval_xg3_series`, it lists the member years of each series as separate lines. Note that 'years' in this context always refers to hydroyears.

Usage

```
extract_xg3_series(data, xg3_type = c("L", "H", "V"), max_gap, min_dur)
```

Arguments

`data`	An object returned by `get_xg3`.
`xg3_type`	Character vector of length 1, 2 or 3. Defines the types of XG3 which are taken from `data`. Specifies the 'X' in 'XG3': either `"L"`, `"H"` and/or `"V"`. Defaults to `"L"`.
`max_gap`	A positive integer (can be zero). It is part of what the user defines as 'an XG3 series': the maximum allowed time gap between two consecutive XG3 values in a series, expressed as the number of years without XG3 value.
`min_dur`	A strictly positive integer. It is part of what the user defines as 'an XG3 series': the minimum required duration of an XG3 series, i.e. the time (expressed as years) from the first to the last year of the XG3 series.

Details

An XG3 series is a location-specific, multi-year series of LG3, HG3 and/or VG3 variables and is user-restricted by `max_gap` (maximum allowed number of empty years between 'series member' years) and `min_dur` (minimum required length of the series). Further, given a dataset of XG3 values per year and location, XG3 series are always constructed *as long as possible* given the aforementioned restrictions. For one location and XG3 variable, more than one such XG3 series may be available, which implies that those XG3 series are separated by more years than the value of `max_gap`.

The function returns the available XG3 series in the dataset for each site and XG3 variable, and numbers them for each site as 'prefix_series1', 'prefix_series2' with the prefix being the value of `xg3_variable`.

The column `xg3_variable` in the resulting tibble stands for the XG3 type + the vertical CRS (see `get_xg3`) to which a series belongs. `xg3_variable` is restricted to the requested XG3 types (LG3, HG3 and/or VG3) via the `xg3_type` argument, but adds an extra level `"combined"` whenever the combination of `data` (which may have both vertical CRSes) and `xg3_type` implies more than one requested variable. This 'combined' level defines an XG3 series as an XG3 series where each 'member' year has **all** selected XG3 variables available.

Value

A tibble with variables:

- loc_code: see get_locs
- xg3_variable: character; see Details
- hydroyear: each member year of a series is listed separately
- series: series ID, unique within loc_code
- series_length: series duration, i.e. from first to last year

See Also

eval_xg3_series

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "KAL")
mydata <-
 mylocs %>%
 get_xg3(watina, 1900)
mydata %>% arrange(loc_code, hydroyear)
mydata %>%
  extract_xg3_series(xg3_type = c("L", "V"),
                     max_gap = 2,
                     min_dur = 5)
# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

---

get_chem *Get hydrochemical data from the database*

Description

Returns hydrochemical data from the *Watina* database, either as a lazy object or as a local tibble. The values must belong to selected locations and to a specified timeframe.

Usage

```
get_chem(
  locs,
  con,
  startdate,
```

```
  enddate = paste(day(today()), month(today()), year(today())),
  conc_type = c("mass", "eq"),
  en_range = c(-0.1, 0.1),
  en_exclude_na = FALSE,
  en_fecond_threshold = 0.0023,
  collect = FALSE
)
```

Arguments

locs	A tbl_lazy object or a dataframe, with at least a column loc_code that defines the locations for which values are to be returned. Typically, this will be the object returned by get_locs.
con	A DBIConnection object to Watina. See connect_watina to generate one.
startdate	First date of the timeframe, as a string. The string must use a formatting of the order 'day month year', i.e. a format which can be interpreted by dmy. Examples: "16-1-2005", "16-01-2005", "1-01-2005", "16/1/2005", "16/1/05", "16/1/88" (years 69 and higher are regarded as 19xy), "16/1-2005", "23 Oct 99", "23 Okt 99" (supposing this notation follows your system locale), "16 1-!!-2005", ...
enddate	Last date of the timeframe, as a string. The same formatting rule must be applied as in startdate. Defaults to a string representation of the current system date.
conc_type	A string defining the type of concentration in *ionic concentration variables*. Either: • "mass": mass concentration (the default); • "eq": equivalent concentration (= normality), referring to the electrical charge of the dissolved ion's main natural form. Note that the argument has no effect on the value of non-ion-variables.
en_range	Numeric vector of length 2. Specifies the allowed range of water sample electroneutrality for ion-variable measurements (see Details). Both vector elements must be within the range c(-1, 1), with the second element not being smaller than the first. Note that this argument only affects the selection of water samples for ionic concentration variables, not for non-ion variables such as pH and electrical conductivity. Measurements of non-ion variables are always returned.
en_exclude_na	Logical. Should ion-variable measurements of water samples with missing electroneutrality value be omitted? Defaults to FALSE. A missing electroneutrality value is the consequence of one or more missing values of ionic concentration variables that are needed for electroneutrality calculation of the water sample. Note that this argument has no effect on the selection of non-ion variable measurements, which are always returned.
en_fecond_threshold	A number (with a sensible default). May be set to NA or NULL by the user. • If en_fecond_threshold is a number (numeric scalar), all measurements from water samples with an iron (meq/l) / conductivity (µS/cm) ratio (Fe/CondL) equal to or larger than en_fecond_threshold are returned, regardless of the en_range and en_exclude_na arguments.

- If en_fecond_threshold is set to NA or NULL, the iron / conductivity ratio is ignored. Hence, no exceptions are made to the conditions imposed by en_range and en_exclude_na (except for measurements of non-ion variables, which are always returned).

collect — Should the data be retrieved as a local tibble? If FALSE (the default), a tbl_lazy object is returned (lazy query). Hence the result can be further built upon before retrieving data with collect().

Details

The timeframe is a selection interval between a given startdate and enddate.

The water samples must meet a specified electroneutrality condition, set by en_range.

- This condition is however ignored when the sample's iron (meq/l) / conductivity (µS/cm) ratio exceeds en_fecond_threshold (use en_fecond_threshold = NA if you don't want this to happen).
- Further, water samples are included by default if their electroneutrality is NA (this is controlled by the en_exclude_na argument).
- Finally, please note that measurements of non-ion variables are *always* returned!

To retrieve all data from all water samples, use en_range = c(-1, 1).

TO BE ADDED: What is electroneutrality and why is it used as a criterion?

Value

By default, a tbl_lazy object. With collect = TRUE, a local tibble is returned.

(TO BE ADDED: Explanation on the variable names of the returned object)

(TO BE ADDED: Explanation on the different abbreviations in the column 'chem_variable')

Note

Up to and including watina 0.3.0, the result was sorted according to loc_code, date and chem_variable, both for the lazy query and the collected result. Later versions avoid sorting in case of a lazy result, because otherwise, when using the result inside another lazy query, this led to 'ORDER BY' constructs in SQL subqueries, which must be avoided. If you like to print the lazy object in a sorted manner, you must add %>% arrange(...) yourself.

See Also

Other functions to query the database: get_locs(), get_xg3()

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "ZWA")
mylocs %>%
    get_chem(watina, "1/1/2017") %>%
```

```
    arrange(loc_code, date, chem_variable)
mylocs %>%
    get_chem(watina, "1/1/2017", collect = TRUE)
mylocs %>%
    get_chem(watina, "1/1/2017", conc_type = "eq") %>%
    arrange(loc_code, date, chem_variable)

# compare the number of returned rows:
mylocs %>% get_chem(watina, "1/1/2017") %>% count
mylocs %>% get_chem(watina, "1/1/2017",
                    en_fecond_threshold = NA) %>% count
mylocs %>% get_chem(watina, "1/1/2017",
                    en_exclude_na = TRUE) %>% count
mylocs %>% get_chem(watina, "1/1/2017",
                    en_exclude_na = TRUE,
                    en_fecond_threshold = NA) %>% count
mylocs %>% get_chem(watina, "1/1/2017",
                    en_range = c(-1, 1)) %>% count

# joining results to mylocs:
mylocs %>%
get_chem(watina, "1/1/2017") %>%
    left_join(mylocs %>%
                  select(-loc_wid),
              .) %>%
    collect %>%
    arrange(loc_code, date, chem_variable)

# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

`get_locs` *Get locations from the database*

Description

Returns locations (and optionally, observation wells) from the *Watina* database that meet several criteria, either as a lazy object or as a local tibble. Criteria refer to spatial or non-spatial physical attributes of the location or the location's observation wells. Essential metadata are included in the result.

Usage

```
get_locs(
  con,
  filterdepth_range = c(0, 3),
  filterdepth_guess = FALSE,
```

```
  filterdepth_na = FALSE,
  obswells = FALSE,
  obswell_aggr = c("latest", "latest_fd", "latest_sso", "mean"),
  mask = NULL,
  join_mask = FALSE,
  buffer = 10,
  bbox = NULL,
  area_codes = NULL,
  loc_type = c("P", "S", "R", "N", "W", "D", "L", "B"),
  loc_validity = c("VLD", "ENT"),
  loc_vec = NULL,
  collect = FALSE
)
```

Arguments

con	A DBIConnection object to Watina. See connect_watina to generate one.
filterdepth_range	Numeric vector of length 2. Specifies the allowed range of the depth of the filter below soil surface, as meters (minimum and maximum allowed filterdepth, respectively). This condition is only applied to groundwater piezometers. The second vector element cannot be smaller than the first. Note that 'filterdepth' takes into account *half* the length of the filter. It is always assumed that filters are at the bottom of the tube. Hence filterdepth = tubelength - filterlength / 2 -[tubelength part above soil surface]. If filterlength is missing, it is assumed to be 0.3 m. With obswells = FALSE, a location is kept whenever one observation well fulfills the condition.
filterdepth_guess	Logical. Only relevant for groundwater piezometers. Defaults to FALSE. For observation wells of which tubelength is known, but not the part of the tubelength above soil surface (height of measuring point), filterdepth cannot be calculated and is missing. However, filterdepth will never be larger than tubelength minus half the filterlength; hence a maximum possible (i.e. conservative) value for filterdepth is given by tubelength - filterlength / 2. With filterdepth_guess = TRUE, filterdepth is replaced by this value when it cannot be calculated and tubelength is available. This is done before applying the filterdepth_range condition. To mark these cases, a logical variable filterdepth_guessed is added to the result: TRUE for wells where filterdepth was replaced; FALSE in all other rows.
filterdepth_na	Logical. Are observation wells with missing filterdepth value to be included? Defaults to FALSE. With filterdepth_guess = TRUE, this has only effect on the *remaining* observation wells with missing filterdepth value.
obswells	Logical. If TRUE, the returned object distinguishes all observation wells (see *Details*) that meet the filterdepth_range condition (or have missing filterdepth, if filterdepth_na = TRUE). If FALSE (the default), the returned object just distinguishes locations. In the latter case, the variables obswell_installdate and obswell_stopdate are not returned.

`obswell_aggr` String. Defines how the attributes of multiple observation wells per location that fulfill the `filterdepth_range` and `filterdepth_na` criteria (after filterdepth adjustment if `filterdepth_guess = TRUE`), are aggregated into one record **per location**:

- `"latest"`: return attributes of the most recent observation well that fulfills the `filterdepth_range` and `filterdepth_na` criteria;
- `"latest_fd"`: return attributes of the most recent observation well that fulfills the `filterdepth_range` condition, i.e. filterdepth will not be missing unless *all* retained wells have missing filterdepth *and* `filterdepth_na = TRUE`;
- `"latest_sso"`: return attributes of the most recent observation well that fulfills the `filterdepth_range` and `filterdepth_na` criteria *and* for which `soilsurf_ost` (soil surface level in the Ostend height CRS (EPSG 5710) is not missing (unless *all* retained wells have missing `soilsurf_ost`);
- `"mean"`: aggregation not by selecting an individual observation well, but by averaging the values of the associated variables `soilsurf_ost`, `measuringref_ost`, `tubelength`, `filterlength`, `filterdepth` for the observation wells with non-missing values (different wells may be involved for each variable, depending on the distribution of missing values). With `filterdepth_guess = TRUE`, the extra variabele `filterdepth_guessed` is summarised as `TRUE` for a location if at least one of the location's observation wells has `filterdepth_guessed = TRUE`.

In all cases the returned value of `obswell_statecode` and `obswell_state` corresponds to the `"latest"` approach. The `obswell_aggr` argument has no effect on locations with a single retained observation well. It is ignored if `obswells = TRUE`.

`mask` An optional geospatial filter of class `sf`. If provided, only locations that intersect with `mask` will be returned, with the value of `buffer` taken into account. The CRS must be Belgian Lambert 72 (EPSG-code 31370).

`join_mask` Logical. Do you want to spatially join the attribute columns of `mask` to the resulting tibble? The spatial join is executed with `st_intersects()` as the topological operator. Beware: if the same location intersects with more than one element of `mask` (taking into account the value of `buffer`), that location will occur multiple times in the result. `join_mask` is ignored if `mask` is not provided.

`buffer` Number of meters taken as a buffer to enlarge `mask` (or shrink it, if `buffer < 0`) if `mask` is provided.

`bbox` Optional geospatial fiter (rectangle). A bounding box (class `bbox`), or a vector of four named elements `xmin`, `xmax`, `ymin`, `ymax` defining the boundary coordinates of a bounding box. If provided, only locations within this rectangular area will be returned. The CRS must be Belgian Lambert 72 (EPSG-code 31370).

`area_codes` An optional vector with area codes. If provided, only locations within the areas will be returned.

`loc_type` Type of the location (mainly: the type of measurement device). Defaults to `"P"`, i.e. only groundwater piezometers are returned by default. Can be a vector with multiple selected values.

`loc_validity` Validation status of the location. Can be a vector with multiple selected values, which must belong to `"VLD"`, `"ENT"`, `"DEL"` or `"CLD"`. Defaults to `c("VLD", "ENT")`.

`loc_vec` An optional vector with location codes. If provided, only locations are returned that are present in this vector.

`collect` Should the data be retrieved as a local tibble? If `FALSE` (the default), a `tbl_lazy` object is returned (lazy query). Hence the result can be further built upon before retrieving data with `collect()`.

Details

(TO BE ADDED: Explanation on the different available values of loc_type and loc_validity)

The lazy object returns a `loc_wid` variable, for further use in *remote* queries. However, don't use it in local objects: `loc_wid` is not to be regarded as stable. Therefore, `collect = TRUE` does not return `loc_wid`.

The result also provides metadata at the level of the observation well, even when `obswells = FALSE`. In the latter case, this refers to the variables `soilsurf_ost`, `measuringref_ost`, `tubelength`, `filterlength`, `filterdepth`. See the argument `obswell_aggr` for options of how to aggregate this information at the location level; by default the latest observation well is used (per location) that meets the criteria on filterdepth. Mind that `obswells = FALSE` and `filterdepth_na = TRUE` may lead to missing filterdepth values at locations which do have a value for an older observation well, but not for the most recent one.

Please note the meaning of observation well in Watina: if there are multiple observation wells attached to one location, these belong to *other timeframes*! So one location always coincides with exactly one observation well at one moment in time. Multiple observation wells can succeed one another because of physical alterations (e.g. damage of a piezometer). Here, the term 'observation well' is used to refer to a fixed installed device in the field (groundwater piezometer, surface water level measurement device).

Value

By default, a `tbl_lazy` object. With `collect = TRUE` or with a specified `mask`, a local `tibble` is returned.

(TO BE ADDED: Explanation on the variable names of the returned object)

Note

Up to and including `watina 0.3.0`, the result was sorted according to `area_code` and `loc_code`, both for the lazy query and the collected result. Later versions avoid sorting in case of a lazy result, because otherwise, when using the result inside another lazy query, this led to 'ORDER BY' constructs in SQL subqueries, which must be avoided. If you like to print the lazy object in a sorted manner, you must add `%>% arrange(...)` yourself.

See Also

Other functions to query the database: `get_chem()`, `get_xg3()`

Examples

```
## Not run:
watina <- connect_watina()

library(dplyr)

get_locs(watina,
         bbox = c(xmin = 1.4e+5,
                  xmax = 1.7e+5,
                  ymin = 1.6e+5,
                  ymax = 1.9e+5)) %>%
    arrange(area_code, loc_code)

get_locs(watina,
         area_codes = c("KAL", "KBR"),
         collect = TRUE)

get_locs(watina,
         area_codes = c("KAL", "KBR"),
         loc_type = c("P", "S"),
         collect = TRUE)

get_locs(watina,
         area_codes = "WES") %>%
    count()

get_locs(watina,
         area_codes = "WES",
         filterdepth_guess = TRUE) %>%
    count()

get_locs(watina,
         area_codes = c("KAL", "KBR"),
         loc_type = c("P", "S"),
         filterdepth_na = TRUE,
         collect = TRUE)

# Mark the different output of:
  get_locs(watina,
           loc_vec = c("KBRP081", "KBRP090", "KBRP095", "KBRS001"),
           loc_type = c("P", "S"),
           collect = TRUE)
  # versus:
  get_locs(watina,
           loc_vec = c("KBRP081", "KBRP090", "KBRP095", "KBRS001"),
           collect = TRUE)

# Returning all individual observation wells:
get_locs(watina,
         obswells = TRUE,
         area_codes = c("KAL", "KBR"),
         loc_type = c("P", "S"),
```

```
         collect = TRUE)

# Different examples of aggregating observation wells at location level:
get_locs(watina,
         area_codes = "WES",
         filterdepth_na = TRUE,
         filterdepth_guess = TRUE,
         obswell_aggr = "latest",
         collect = TRUE) %>%
    select(loc_code, contains("ost"), contains("filterdepth")) %>%
    head(12)

get_locs(watina,
         area_codes = "WES",
         filterdepth_na = TRUE,
         filterdepth_guess = TRUE,
         obswell_aggr = "mean",
         collect = TRUE) %>%
    select(loc_code, contains("ost"), contains("filterdepth")) %>%
    head(12)

# Selecting all piezometers with status VLD of the
# province "West-Vlaanderen" (current polygon taken
# from the official WFS service):
library(sf)
library(purrr)
library(httr)
mymask <-
    "https://geoservices.informatievlaanderen.be/overdrachtdiensten/VRBG/wfs" %>%
    parse_url() %>%
    list_merge(query = list(request = "GetFeature",
                            typeName = "VRBG:Refprv",
                            cql_filter="NAAM='West-Vlaanderen'",
                            srsName = "EPSG:31370",
                            outputFormat = "text/xml; subtype=gml/3.1.1")) %>%
    build_url() %>%
    read_sf(crs = 31370) %>%
    st_cast("GEOMETRYCOLLECTION")
get_locs(watina,
         loc_validity = "VLD",
         mask = mymask,
         buffer = 0)

# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

---

get_xg3 *Get XG3 values from the database*

Description

Returns XG3 values from the *Watina* database, either as a lazy object or as a local tibble. The values must belong to selected locations and to a specified timeframe.

Usage

```
get_xg3(
  locs,
  con,
  startyear,
  endyear = year(now()) - 1,
  vert_crs = c("local", "ostend", "both"),
  truncated = TRUE,
  with_estimated = TRUE,
  collect = FALSE
)
```

Arguments

`locs`	A `tbl_lazy` object or a dataframe, with at least a column `loc_code` that defines the locations for which values are to be returned. Typically, this will be the object returned by `get_locs`.
`con`	A `DBIConnection` object to Watina. See `connect_watina` to generate one.
`startyear`	First hydroyear of the timeframe.
`endyear`	Last hydroyear of the timeframe.
`vert_crs`	A string, defining the 1-dimensional vertical coordinate reference system (CRS) of the XG3 water levels. Either `"local"` (the default, i.e. returned values are relative to soil surface level, with positive values = above soil surface), or `"ostend"` (values are from the CRS Ostend height (EPSG 5710), also known as 'TAW' or 'DNG'), or `"both"`, where the values for both CRS options are returned. The units are always meters.
`truncated`	Logical. If `TRUE` (the default), the XG3 values are calculated after having set the underlying water level measurements that are above soil surface level to the soil surface level itself (which is zero in the case of the local CRS).
`with_estimated`	Logical. If `TRUE` (the default), the XG3 values calculations also use estimated (i.e. non-measured) water level data that are available in the database.
`collect`	Should the data be retrieved as a local tibble? If `FALSE` (the default), a `tbl_lazy` object is returned (lazy query). Hence the result can be further built upon before retrieving data with `collect()`.

Details

The timeframe is a selection interval between a given first and last hydroyear.

Note: the arguments `truncated` and `with_estimated` are currently not used. Currently, non-truncated values are returned, with usage of estimated values.

(TO BE ADDED: What are XG3 values? What is a hydroyear? Why truncate, and why truncate by default? When to choose which `vert_crs`?)

Value

By default, a tbl_lazy object. With collect = TRUE, a local tibble is returned.

(TO BE ADDED: Explanation on the variable names of the returned object)

The suffix of the XG3 variables is either "_lcl" for vert_crs = "local" or "_ost" for vert_crs = "ostend".

Note

Up to and including watina 0.3.0, the result was sorted according to loc_code and hydroyear, both for the lazy query and the collected result. Later versions avoid sorting in case of a lazy result, because otherwise, when using the result inside another lazy query, this led to 'ORDER BY' constructs in SQL subqueries, which must be avoided. If you like to print the lazy object in a sorted manner, you must add %>% arrange(...) yourself.

See Also

Other functions to query the database: get_chem(), get_locs()

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "KAL")
mylocs %>%
    get_xg3(watina, 2010) %>%
    arrange(loc_code, hydroyear)
mylocs %>% get_xg3(watina, 2010, collect = TRUE)
mylocs %>%
    get_xg3(watina, 2010, vert_crs = "ostend") %>%
    arrange(loc_code, hydroyear)

# joining results to mylocs:
mylocs %>%
    get_xg3(watina, 2010) %>%
    left_join(mylocs %>%
              select(-loc_wid),
              .) %>%
    collect %>%
    arrange(loc_code, hydroyear)

# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

selectlocs_chem *Select locations based on hydrochemical data properties*

Description

Select locations that comply with user-specified conditions, from a dataset as returned by either get_chem or eval_chem. Conditions can be specified for each of the summary statistics returned by eval_chem.

Usage

```
selectlocs_chem(
  data,
  data_type = c("data", "summary"),
 chem_var = c("P-PO4", "N-NO3", "N-NO2", "N-NH4", "HCO3", "SO4", "Cl", "Na", "K", "Ca",
    "Mg", "Fe", "Mn", "Si", "Al", "CondF", "CondL", "pHF", "pHL"),
  conditions,
  verbose = TRUE,
  list = FALSE
)
```

Arguments

data	An object as returned by either get_chem (the object corresponds to real 'data') or eval_chem (the object contains summary values).
data_type	A string. Either "data" (the default) or "summary", in correspondence with the choice made for data.
chem_var	Only relevant when data is an object formatted as returned by get_chem. Is a character vector to select chemical variables for which statistics will be computed. To specify chemical variables, use the codes from the column chem_variable in data. Together with the available variables in data, chem_var determines the meaning of the variable "combined".
conditions	A dataframe. See the devoted section below.
verbose	Logical. If TRUE, give feedback on dropped locations because of (specific) unused conditions and other 'mismatch' reasons.
list	Logical. If FALSE (the default), the function only returns the end-result (a tibble with selected location codes). If TRUE, the function returns a list with the end-result plus useful intermediate results (see Value).

Details

selectlocs_chem() separately runs eval_chem on the input (data) if data_type = "data". See the documentation of eval_chem to learn more about the available summary statistics. Each condition for evaluation + selection of locations is specific to a chemical *variable*, which can also be the level 'combined'. Hence, the result will depend both on the *chemical variables* for which statistics

have been computed (specified by chem_var), and on the conditions, specified by conditions. See the devoted section on the conditions dataframe.

Only locations are returned:

- which have **all** chemical variables, implied by chem_var and present in conditions, available in data. (In other words, all conditions must be testable.)
- for which **all** conditions are met;

As the conditions imposed by the conditions dataframe are always evaluated as a required combination of conditions ('and'), the user must make different calls to selectlocs_chem() if different sets of conditions are to be allowed ('or').

If data_type = "data", selectlocs_chem() calls eval_chem. Its type and uniformity_test arguments are derived from the user-specified conditions dataframe.

selectlocs_chem() joins the long-formatted results of eval_chem with the conditions dataframe in order to evaluate the conditions. Often, this join in itself already leads to dropping specific combinations of loc_code and chem_variable. At least the locations that are completely dropped in this step are reported when verbose = TRUE.

The user may want to repeatedly try different sets of conditions until a satisfying selection of locations is returned. However the output of eval_chem will not change as long as the data are not altered. For that reason, the user can also feed the result of eval_chem() to the data argument, with data_type = "summary". In that case the argument chem_var is ignored.

Value

If list = FALSE: a tibble with one column loc_code that provides the locations selected by the conditions.

If list = TRUE: a list of tibbles that extends the previous end-result with intermediate results. All list elements are named:

1. combined_result_filtered: the end-result, same as given by list = FALSE.
2. result: the test result of each computed and tested statistic for each location and chemical variable: 'condition met' (cond_met) is TRUE or FALSE.
3. combined_result: aggregation of result **per location**. Specific columns: all_cond_met is TRUE if *all* conditions for that location were TRUE, and is FALSE in all other cases. pct_cond_met is the percentage of 'met' availability conditions per location.

Specification of the conditions dataframe

Conditions can be specified for each of the summary statistics returned by eval_chem.

The conditions parameter takes a dataframe that must have the following columns:

chem_variable Can be any chemical variable code, including "combined".

statistic Name of the statistic to be evaluated.

criterion Numeric. Defines the value of the statistic on which the condition will be based.

For condition testing on statistics of type 'date', provide the numeric date representation, i.e. the number of days since 1 Jan 1970 (older dates are negative). This can be easily calculated for a given 'datestring' (e.g. "18-5-2020") with: as.numeric(lubridate::dmy(datestring)).

`direction` One of: `"min"`, `"max"`, `"equal"`. Together with `criterion`, this completes the condition which will be evaluated with respect to the specific `chem_variable`: for `direction = "min"`, the statistic must be the criterion value or larger; for `direction = "max"`, the statistic must be the criterion value or lower; for `direction = "equal"`, the statistic must be equal to the criterion value.

Each condition is one row of the dataframe. The dataframe should have at least one, and may have many. Each combination of `chem_variable` and `statistic` must be unique. Conditions on chemical variables, absent from `data` or not implied by `chem_var`, will be dropped without warning. Hence, it is up to the user to do sensible things.

The possible statistics for conditions on chemical variables are documented by `eval_chem`.

See Also

`eval_chem`

Other functions to select locations: `selectlocs_xg3()`

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina, area_codes = "ZWA")
mydata <-
    mylocs %>%
    get_chem(watina, "1/1/2010")
mydata %>% arrange(loc_code, date, chem_variable)
mydata %>%
    pull(date) %>%
    lubridate::year(.) %>%
    (function(x) c(firstyear = min(x), lastyear = max(x)))

## EXAMPLE 1
# to prepare a condition on 'firstdate', we need its numerical value:
as.numeric(lubridate::dmy("1/1/2014"))
conditions_df <-
    tribble(
        ~chem_variable, ~statistic, ~criterion, ~direction,
        "N-NO3", "nrdates", 2, "min",
        "P-PO4", "nrdates", 2, "min",
        "P-PO4", "firstdate", 16071, "max",
        "P-PO4", "timespan_years", 5, "min"
    )
conditions_df
myresult <-
    mydata %>%
    selectlocs_chem(data_type = "data",
                    chem_var = c("N-NO3", "P-PO4"),
                    conditions = conditions_df,
                    list = TRUE)
myresult
```

```
# or:
# mystats <- eval_chem(mydata, chem_var = c("N-NO3", "P-PO4"))
# myresult <-
#   mystats %>%
#   selectlocs_chem(data_type = "summary",
#                   conditions = conditions_df,
#                   list = TRUE)
myresult$combined_result_filtered

## EXAMPLE 2
# An example based on numeric statistics:
conditions_df <-
    tribble(
        ~chem_variable, ~statistic, ~criterion, ~direction,
        "pHF", "val_mean", 5, "max",
        "CondF", "val_pct50", 100, "min"
    )
conditions_df
mydata %>%
    selectlocs_chem(data_type = "data",
                    chem_var = c("pHF", "CondF"),
                    conditions = conditions_df)

# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

---

selectlocs_xg3 *Select locations based on XG3 availability and XG3 series' properties*

Description

Select locations that comply with user-specified conditions, from a dataset as returned by get_xg3, *or* from a list with the outputs of eval_xg3_avail and eval_xg3_series. Conditions can be specified for each of the summary statistics returned by eval_xg3_avail and eval_xg3_series.

Usage

```
selectlocs_xg3(
  data,
  xg3_type = NULL,
  max_gap = NULL,
  min_dur = NULL,
  conditions,
  verbose = TRUE,
  list = FALSE
)
```

Arguments

data	Either an object returned by get_xg3, or a named list of two tibbles: "avail" and "ser". In the latter case, "avail" must be the output of eval_xg3_avail and "ser" must be the output of eval_xg3_series, whereby each function was applied to the same dataset and used the same setting for the xg3_type argument. See Details.
xg3_type	Only relevant when data is an object formatted as returned by get_xg3. In that case, must be a character vector of length 1, 2 or 3, which will default to "L" if not specified. Defines the types of XG3 which are taken from data for the eval_xg3_xxx() functions. Specifies the 'X' in 'XG3': either "L", "H" and/or "V". Together with the available variables in data, xg3_type determines the meaning of the variable "combined".
max_gap	A positive integer (can be zero). It is part of what the user defines as 'an XG3 series': the maximum allowed time gap between two consecutive XG3 values in a series, expressed as the number of years without XG3 value.
min_dur	A strictly positive integer. It is part of what the user defines as 'an XG3 series': the minimum required duration of an XG3 series, i.e. the time (expressed as years) from the first to the last year of the XG3 series.
conditions	A dataframe. See the devoted section below.
verbose	Logical. If TRUE, give feedback on dropped locations because of (specific) unused conditions and other 'mismatch' reasons.
list	Logical. If FALSE (the default), the function only returns the end-result (a tibble with selected location codes). If TRUE, the function returns a list with the end-result plus useful intermediate results (see Value).

Details

selectlocs_xg3() separately runs eval_xg3_avail() and eval_xg3_series() on the input (data) if the latter conforms to the output of get_xg3. See the documentation of eval_xg3_avail and eval_xg3_series to learn more about how an 'XG3 variable' and an 'XG3 series' are defined, and about the available summary statistics. Each condition for evaluation + selection of locations is specific to an XG3 *variable*, which can also be the level 'combined'. Hence, the result will depend both on the XG3 *types* (HG3, LG3 and/or VG3) for which statistics have been computed (specified by xg3_type), and on the conditions, specified by conditions. See the devoted section on the conditions dataframe.

Only locations are returned:

- which have **all** XG3 variables, implied by xg3_type and present in conditions, available in data. (In other words, all conditions must be testable.)
- for which **all** conditions are met;

As the conditions imposed by the conditions dataframe are always evaluated as a required combination of conditions ('and'), the user must make different calls to selectlocs_xg3() if different sets of conditions are to be allowed ('or').

Regarding conditions that evaluate XG3 *series*, it is taken into account that one location can have multiple series for the same XG3 variable. When the user provides one or more conditions for the

series of a specific XG3 variable, the condition(s) are regarded as fulfilled ('condition met') when **at least one** series is present of that XG3 variable for which **all** those conditions are met.

`selectlocs_xg3()` joins the long-formatted results of `eval_xg3_avail` and `eval_xg3_series` with the `conditions` dataframe in order to evaluate the conditions. Often, this join in itself already leads to dropping specific combinations of `loc_code` and `xg3_variable`. At least the locations that are completely dropped in this step are reported when `verbose = TRUE`.

For larger datasets `eval_xg3_series()` can take quite some time, whereas the user may want to repeatedly try different sets of conditions until a satisfying selection of locations is returned. However the output of both `eval_xg3_avail()` and `eval_xg3_series()` will not change as long as the data and the chosen values of `max_gap` and `min_dur` are not altered. For that reason, the user can also prepare a list object with the respective results of `eval_xg3_avail()` and `eval_xg3_series()`, which must be named as `"avail"` and `"ser"`, respectively. This list can instead be used as data-input, and in that case `xg3_type`, `max_gap` and `min_dur` are not needed (they will be ignored).

Value

If `list = FALSE`: a tibble with one column `loc_code` that provides the locations selected by the conditions.

If `list = TRUE`: a list of tibbles that extends the previous end-result with intermediate results. The below elements nrs. 2 and 3 are only given when at least one XG3 availability condition was given, nrs. 4, 5 and 6 only when at least one XG3 series condition was given, and nr. 7 is only returned when both types of condition were given. All list elements are named:

1. `combined_result_filtered`: the end-result, same as given by `list = FALSE`.
2. `result_avail`: the test result of each computed and tested *availability* statistic for each location and XG3 variable: 'condition met' (`cond_met`) is TRUE or FALSE.
3. `combined_result_avail`: aggregation of `result_avail` **per location**. Specific columns: `cond_met_avail` is `TRUE` if *all* availability conditions for that location were `TRUE`, and is `FALSE` in all other cases. `pct_cond_met_avail` is the percentage of 'met' availability conditions per location.
4. `result_series`: the test result of each computed and tested *series* statistic for each location and XG3 series: 'condition met' (`cond_met`) is TRUE or FALSE.
5. `combined_result_series_xg3var`: aggregation of `result_series` **per location and XG3 variable**. Two consecutive aggregation steps are involved here:
   (a) per XG3 series: are *all* series conditions met?
   (b) per XG3 variable: is there *at least one* series where all series conditions are met?

   Specific columns: `all_ser_cond_met_xg3var` is the answer to question 2 (TRUE/FALSE). `avg_pct_cond_met_nonpassed_series` is the average percentage (for a location and XG3 variable) of 'met' series conditions in the series where *not* all conditions were met. (Note that the same percentage is 100 for series where all conditions are met, leading to `all_ser_cond_met_xg3var = TRUE` at the level of location and XG3 variable.)
6. `combined_result_series`: aggregation of `combined_result_series_xg3var` **per location**. Specific columns: `cond_met_series` is `TRUE` if *all* XG3 variables (that have series and on which series conditions were imposed) were `TRUE` in the previous aggregation (`all_ser_cond_met_xg3var = TRUE`), and is `FALSE` in all other cases. `pct_xg3vars_passed_ser` is the percentage of a location's XG3 variables (that have series and on which series conditions were imposed) that

were TRUE in the previous aggregation (all_ser_cond_met_xg3var = TRUE). avg_pct_cond_met_in_nonpassed_ser is the average of avg_pct_cond_met_nonpassed_series from the previous aggregation step, over the involved XG3 variables at the location.

7. combined_result: the inner join (on loc_code) between combined_result_avail and combined_result_series. Locations that were dropped in either evaluation because of missing information, are dropped here too, because the function is to return locations for which all conditions hold and hence could be verified.

   The last column, all_cond_met, requires both cond_met_avail = TRUE and cond_met_series = TRUE to result in TRUE for a location.

   Notes:

   - locations with all_cond_met = FALSE are not discarded in this object;
   - filtering locations with all_cond_met = TRUE will result in exactly the locations given by combined_result_filtered, see list element 1;
   - the object is only returned when both availability and series condition(s) were given (at least one of each family). In the other cases, you can directly look at combined_result_avail or combined_result_series, from which combined_result_filtered is derived.

Specification of the conditions dataframe

Conditions can be specified for each of the summary statistics returned by eval_xg3_avail and eval_xg3_series. Consequently, XG3 availability conditions and XG3 series conditions can be distinguished.

The conditions parameter takes a dataframe that must have the following columns:

xg3_variable One of: "combined","lg3_lcl","lg3_ost","vg3_lcl","vg3_ost","hg3_lcl","hg3_ost".

statistic Name of the statistic to be evaluated.

criterion Numeric. Defines the value of the statistic on which the condition will be based.

direction One of: "min","max","equal". Together with criterion, this completes the condition which will be evaluated with respect to the specific xg3_variable: for direction = "min", the statistic must be the criterion value or larger; for direction = "max", the statistic must be the criterion value or lower; for direction = "equal", the statistic must be equal to the criterion value.

Each condition is one row of the dataframe. The dataframe should have at least one, and may have many. Each combination of xg3_variable and statistic must be unique. Conditions on XG3 variables, absent from data or not implied by xg3_type, will be dropped without warning. Hence, it is up to the user to do sensible things.

The possible statistics for XG3 availability conditions are: *nryears, firstyear, lastyear*.

The possible statistics for XG3 series conditions are: *ser_length, ser_nryears, ser_rel_nryears, ser_firstyear, ser_lastyear, ser_pval_uniform, ser_mean, ser_sd, ser_se_6y, ser_rel_sd_lcl, ser_rel_se_6y_lcl*. The last six are not defined for the XG3 variable 'combined', and the last two are only defined for variables with a local vertical CRS.

See Also

eval_xg3_avail, eval_xg3_series

Other functions to select locations: selectlocs_chem()

Examples

```
## Not run:
watina <- connect_watina()
library(dplyr)
mylocs <- get_locs(watina,
                   area_codes = "TOR",
                   loc_type = c("P", "S"))
mydata <-
 mylocs %>%
 get_xg3(watina, 2000)
mydata %>% arrange(loc_code, hydroyear)
# Number of locations in mydata:
mydata %>% distinct(loc_code) %>% count
# Number of hydrological years per location and XG3 variable:
mydata %>%
  group_by(loc_code) %>%
  collect %>%
  summarise(lg3_lcl = sum(!is.na(lg3_lcl)),
            hg3_lcl = sum(!is.na(hg3_lcl)),
            vg3_lcl = sum(!is.na(vg3_lcl)))
conditions_df <-
  tribble(
  ~xg3_variable, ~statistic, ~criterion, ~direction,
  "lg3_lcl", "ser_lastyear", 2015, "min",
  "hg3_lcl", "ser_lastyear", 2015, "min"
  )
conditions_df
result <-
  mydata %>%
  selectlocs_xg3(xg3_type = c("L", "H"),
                 max_gap = 1,
                 min_dur = 5,
                 conditions = conditions_df,
                 list = TRUE)
# or:
# mystats <- list(avail = eval_xg3_avail(mydata,
#                                        xg3_type = c("L", "H")),
#                 ser =  eval_xg3_series(mydata,
#                                        xg3_type = c("L", "H"),
#                                        max_gap = 1,
#                                        min_dur = 5))
# result <-
#   mystats %>%
#   selectlocs_xg3(conditions = conditions_df,
#                  list = TRUE)
result$combined_result_filtered
result[2:4]
# Disconnect:
dbDisconnect(watina)

## End(Not run)
```

Index