Package ‘SHARK4R’

March 12, 2026

Title Accessing and Validating Marine Environmental Data from 'SHARK' and Related Databases

Version 1.1.1

Description Provides functions to retrieve, process, analyze, and quality-control marine physical, chemical, and biological data. The main focus is on Swedish monitoring data available through the 'SHARK' database <https://shark.smhi.se/en/>, with additional API support for 'Nordic Microalgae' <https://nordicmicroalgae.org/>, 'Dyntaxa' <https://artfakta.se/>, World Register of Marine Species ('WoRMS') <https://www.marinespecies.org>, 'AlgaeBase' <https://www.algaebase.org>, OBIS 'xylookup' web service <https://iobis.github.io/xylookup/> and Intergovernmental Oceanographic Commission (IOC) - UNESCO databases on harmful algae <https://www.marinespecies.org/hab/> and toxins <https://toxins.hais.ioc-unesco.org/>.

License MIT + file LICENSE

URL https://sharksmhi.github.io/SHARK4R/, https://github.com/sharksmhi/SHARK4R

BugReports https://github.com/sharksmhi/SHARK4R/issues

Depends R (>= 4.1.0)

Imports dplyr, DT, ggplot2, httr, jsonlite, leaflet, lifecycle, purrr, readr, readxl, rlang, sf, sp, stringi, terra, tidyr, vroom, worrms

Suggests htmltools, iRfcb, knitr, plotly, RColorBrewer, rmarkdown, skimr, spelling, shiny, shinythemes, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Markus Lindh [aut] (Swedish Meteorological and Hydrological Institute, ORCID: <https://orcid.org/0000-0002-7120-4145>),
Anders Torstensson [aut, cre] (Swedish Meteorological and Hydrological Institute, ORCID: <https://orcid.org/0000-0002-8283-656X>),
Mikael Hedblom [ctb] (Swedish Meteorological and Hydrological Institute, ORCID: <https://orcid.org/0009-0007-5124-9956>),
Bengt Karlson [ctb] (Swedish Meteorological and Hydrological Institute, ORCID: <https://orcid.org/0000-0002-7524-3504>),
SHARK [cph],
SBDI [fnd] (Swedish Research Council, 2019-00242)

Maintainer Anders Torstensson <anders.torstensson@smhi.se>

Repository CRAN

Date/Publication 2026-03-12 08:20:39 UTC

Contents

`add_worms_taxonomy` *Add WoRMS taxonomy hierarchy to AphiaIDs or scientific names*

Description

This function enhances a dataset of AphiaIDs (and optionally scientific names) with their complete hierarchical taxonomy from the World Register of Marine Species (WoRMS). Missing AphiaIDs can be resolved from scientific names automatically.

Usage

```
add_worms_taxonomy(
  aphia_ids,
  scientific_names = NULL,
  add_rank_to_hierarchy = FALSE,
  verbose = TRUE,
  aphia_id = deprecated(),
  scientific_name = deprecated()
)
```

Arguments

`aphia_ids`	Numeric vector of AphiaIDs.
`scientific_names`	Optional character vector of scientific names (same length as `aphia_id`).
`add_rank_to_hierarchy`	Logical (default FALSE). If TRUE, includes rank labels in the concatenated hierarchy string.
`verbose`	Logical (default TRUE). If TRUE, prints progress updates.
`aphia_id`	**[Deprecated]** Use `aphia_ids` instead.
`scientific_name`	**[Deprecated]** Use `scientific_names` instead.

Value

A `tibble` with taxonomy columns added, including:

- `aphia_id`, `scientific_name`
- `worms_kingdom`, `worms_phylum`, `worms_class`, `worms_order`, `worms_family`, `worms_genus`, `worms_species`
- `worms_scientific_name`, `worms_hierarchy`

Examples

```
# Using AphiaID only
try(add_worms_taxonomy(c(1080, 109604), verbose = FALSE))

# Using a combination of AphiaID and scientific name
try(add_worms_taxonomy(
  aphia_ids = c(NA, 109604),
  scientific_names = c("Calanus finmarchicus", "Oithona similis"),
  verbose = FALSE
))
```

assign_phytoplankton_group

Assign phytoplankton group to scientific names

Description

This function assigns default phytoplankton groups (Diatoms, Dinoflagellates, Cyanobacteria, or Other) to a list of scientific names or Aphia IDs by retrieving species information from the World Register of Marine Species (WoRMS). The function checks both Aphia IDs and scientific names, handles missing records, and assigns the appropriate plankton group based on taxonomic classification in WoRMS. Additionally, custom plankton groups can be specified using the `custom_groups` parameter, allowing users to define additional classifications based on specific taxonomic criteria.

Usage

```
assign_phytoplankton_group(
  scientific_names,
  aphia_ids = NULL,
  diatom_class = c("Bacillariophyceae", "Coscinodiscophyceae", "Mediophyceae",
    "Diatomophyceae"),
  dinoflagellate_class = "Dinophyceae",
  cyanobacteria_class = "Cyanophyceae",
  cyanobacteria_phylum = "Cyanobacteria",
  match_first_word = TRUE,
  marine_only = FALSE,
  return_class = FALSE,
  custom_groups = list(),
  verbose = TRUE
)
```

Arguments

`scientific_names`	A character vector of scientific names of marine species.
`aphia_ids`	A numeric vector of Aphia IDs corresponding to the scientific names. If provided, it improves the accuracy and speed of the matching process. The length of `aphia_ids` must match the length of `scientific_names`. Defaults to `NULL`, in which case the function will attempt to assign plankton groups based only on the scientific names.
`diatom_class`	A character string or vector representing the diatom class. Default is "Bacillariophyceae", "Coscinodiscophyceae", "Mediophyceae" and "Diatomophyceae".
`dinoflagellate_class`	A character string or vector representing the dinoflagellate class. Default is "Dinophyceae".
`cyanobacteria_class`	A character string or vector representing the cyanobacteria class. Default is "Cyanophyceae".

`cyanobacteria_phylum`
: A character string or vector representing the cyanobacteria phylum. Default is "Cyanobacteria".

`match_first_word`
: A logical value indicating whether to match the first word of the scientific name if the Aphia ID is missing. Default is TRUE.

`marine_only`
: A logical value indicating whether to restrict the results to marine taxa only. Default is `FALSE`.

`return_class`
: A logical value indicating whether to include class information in the result. Default is `FALSE`.

`custom_groups`
: A named list of additional custom plankton groups (optional). The names of the list correspond to the custom group names (e.g., "Cryptophytes"), and the values should be character vectors specifying one or more of the following taxonomic levels: `phylum`, `class`, `order`, `family`, `genus`, or `scientific_name`. For example: `list("Green Algae" = list(class = c("Chlorophyceae", "Ulvophyceae")))`. This allows users to extend the default classifications (e.g., Cyanobacteria, Diatoms, Dinoflagellates) with their own groups.

`verbose`
: A logical value indicating whether to print progress messages. Default is TRUE.

Details

The `aphia_ids` parameter is not necessary but, if provided, will improve the certainty of the matching process. If `aphia_ids` are available, they will be used directly to retrieve more accurate WoRMS records. If missing, the function will attempt to match the scientific names to Aphia IDs by querying WoRMS using the scientific name(s), with an additional fallback mechanism to match based on the first word of the scientific name.

To skip one of the default plankton groups, you can set the class or phylum of the respective group to an empty string (`""`). For example, to skip the "Cyanobacteria" group, you can set `cyanobacteria_class = ""` or `cyanobacteria_phylum = ""`. These taxa will then be placed in `Others`.

Custom groups are processed in the order they appear in the `custom_groups` list. If a taxon matches multiple custom groups, it will be assigned to the group that appears last in the list, as later matches overwrite earlier ones. For example, if `Teleaulax amphioxeia` matches both `Cryptophytes` (class-based) and a specific group `Teleaulax` (name-based), it will be assigned to `Teleaulax` if `Teleaulax` is listed after `Cryptophytes` in the `custom_groups` list.

Value

A `tibble` with two columns: `scientific_name` and `plankton_group`, where the plankton group is assigned based on taxonomic classification.

See Also

`https://marinespecies.org/` for WoRMS website.

`https://CRAN.R-project.org/package=worrms`

Examples

```
# Assign plankton groups to a list of species names
try(result <- assign_phytoplankton_group(
  scientific_names = c("Tripos fusus", "Diatoma", "Nodularia spumigena", "Octactis speculum"),
  verbose = FALSE))

if (exists("result")) print(result)

# Improve classification by explicitly providing Aphia IDs for ambiguous taxa
# Actinocyclus and Navicula are names shared by both diatoms and animals,
# which can lead to incorrect group assignment without an Aphia ID
try(result <- assign_phytoplankton_group(
  scientific_names = c("Actinocyclus", "Navicula", "Nodularia spumigena", "Tripos fusus"),
  aphia_ids = c(148944, 149142, NA, NA),
  verbose = FALSE))

if (exists("result")) print(result)

# Assign plankton groups using additional custom grouping
custom_groups <- list(
    Cryptophytes = list(class = "Cryptophyceae"),
    Ciliates = list(phylum = "Ciliophora")
)

# Assign with custom groups
try(result_custom <- assign_phytoplankton_group(
  scientific_names = c("Teleaulax amphioxeia", "Mesodinium rubrum", "Dinophysis acuta"),
  aphia_ids = c(106306, 232069, 109604),
  custom_groups = custom_groups,         # Adding custom groups
  verbose = FALSE
))

if (exists("result_custom")) print(result_custom)
```

`check_codes` *Check matches of reported codes in SMHI's SHARK codelist*

Description

This function checks whether the codes reported in a specified column of a dataset (e.g., project codes, ship codes, etc.) are present in the official SHARK codelist provided by SMHI. If a cell contains multiple codes separated by commas, each code is checked individually. The function downloads and caches the codelist if necessary, compares the reported values against the valid codes, and returns a `tibble` showing which codes matched. Informative messages are printed if unmatched codes are found.

Usage

```
check_codes(
  data,
  field = "sample_project_name_en",
  code_type = "PROJ",
  match_column = "Description/English translate",
  clean_cache_days = 30,
  verbose = TRUE
)
```

Arguments

`data`	A tibble (or data.frame) containing the codes to check.
`field`	Character; name of the column in `data` that contains the codes to be validated against the SHARK codelist. If a cell contains multiple codes separated by commas, each code is validated separately. Default is `"sample_project_name_en"`.
`code_type`	Character; the type of code to check (e.g., `"PROJ"`). Defaults to `"PROJ"`.
`match_column`	Character; the column in the SHARK codelist to match against. Must be one of `"Code"` or `"Description/English translate"`. Defaults to `"Description/English translate"`.
`clean_cache_days`	Numeric; if not `NULL`, cached SHARK code Excel files older than this number of days will be automatically deleted and replaced by a new download. Defaults to 30. Set to `NULL` to disable automatic cleanup.
`verbose`	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Value

A `tibble` with unique reported codes (after splitting comma-separated entries) and a logical column `match_type` indicating if they exist in the SHARK codelist.

See Also

`get_shark_codes()` to get the current code list.

`clean_shark4r_cache()` to manually clear cached files.

`check_datatype` *Validate SHARK system fields in a data frame*

Description

This function checks whether the required and recommended global and datatype-specific SHARK system fields are present in a data frame.

Usage

```
check_datatype(data, level = "error")
```

Arguments

`data`	A `data.frame` or `tibble` containing SHARK data to validate.
`level`	Character. The level of validation: • `"error"` (default) — checks only required fields. • `"warning"` — checks both required and recommended fields.

Details

- **Required fields**: Missing or empty required fields are reported as **errors**.
- **Recommended fields**: Missing or empty recommended fields are reported as **warnings**, but only if `level = "warning"` is specified.

Value

A `tibble` summarizing missing or empty fields, with columns:

- `level`: `"error"` or `"warning"`.
- `field`: Name of the missing or empty field.
- `row`: Row number where the value is missing (NA) or `NA` if the whole column is missing.
- `message`: Description of the issue.

Examples

```
# Example with required fields missing
df <- data.frame(
  visit_year = 2024,
  station_name = NA
)
check_datatype(df, level = "error")

# Example checking recommended fields as warnings
check_datatype(df, level = "warning")
```

`check_depth` *Validate depth values against bathymetry and logical constraints*

Description

`check_depth()` inspects one or two depth columns in a dataset and reports potential problems such as missing values, non-numeric entries, or values that conflict with bathymetry and shoreline information. It can also validate depths against bathymetry data retrieved from a terra::SpatRaster object or, if `bathymetry = NULL`, via the `lookup_xy()` function, which calls the OBIS XY lookup API to obtain bathymetry (using EMODnet Bathymetry) and shore distance.

Usage

```
check_depth(
  data,
  depth_cols = c("sample_min_depth_m", "sample_max_depth_m"),
  lat_col = "sample_latitude_dd",
  lon_col = "sample_longitude_dd",
  report = TRUE,
  depthmargin = 0,
  shoremargin = NA,
  bathymetry = NULL
)
```

Arguments

`data`	A data frame containing sample metadata, including longitude, latitude, and one or two depth columns.
`depth_cols`	Character vector naming the depth column(s). Can be one column (e.g., `"water_depth_m"`) or two columns (minimum and maximum depth, e.g., `c("sample_min_depth_m", "sample_max_depth_m")`).
`lat_col`	Name of the column containing latitude values. Default: `"sample_latitude_dd"`.
`lon_col`	Name of the column containing longitude values. Default: `"sample_longitude_dd"`.
`report`	Logical. If `TRUE` (default), returns a tibble of detected problems. If `FALSE`, returns the subset of input rows that failed validation.
`depthmargin`	Numeric. Allowed deviation (in meters) above bathymetry before a depth is flagged as an error. Default = `0`.
`shoremargin`	Numeric. Minimum offshore distance (in meters) required for negative depths to be considered valid. If `NA` (default), this check is skipped.
`bathymetry`	Optional terra::SpatRaster object with one layer giving bathymetry values. If `NULL` (default), bathymetry and shore distance are retrieved using `lookup_xy()`, which calls the OBIS XY lookup API.

Details

The following checks are performed:

1. **Missing depth column** → warning
2. **Empty depth column** (all values missing) → warning
3. **Non-numeric depth values** → warning
4. **Depth exceeds bathymetry + margin** (`depthmargin`) → warning
5. **Negative depth at offshore locations** (beyond `shoremargin`) → warning
6. **Minimum depth greater than maximum depth** (if two columns supplied) → error
7. **Longitude/latitude outside raster bounds** → warning
8. **Missing bathymetry value** at coordinate → warning

The function has been modified from the `obistools` package (Provoost and Bosch, 2024).

Value

A tibble with one row per detected problem, containing:

level Severity of the issue ("warning" or "error").

row Row index in the input data where the issue occurred.

field Name of the column(s) involved.

message Human-readable description of the problem.

If `report = FALSE`, returns the subset of input rows that failed any check.

References

Provoost P, Bosch S (2024). “obistools: Tools for data enhancement and quality control” Ocean Biodiversity Information System. Intergovernmental Oceanographic Commission of UNESCO. R package version 0.1.0, `https://iobis.github.io/obistools/`.

See Also

`lookup_xy`, `check_onland`

Examples

```
# Example dataset with one depth column
example_data <- data.frame(
  sample_latitude_dd = c(59.3, 58.1, 57.5),
  sample_longitude_dd = c(18.0, 17.5, 16.2),
  sample_depth_m = c(10, -5, NA)
)

# Validate depths using OBIS XY lookup (bathymetry = NULL)
try(check_depth(example_data, depth_cols = "sample_depth_m"))

# Example dataset with min/max depth columns
example_data2 <- data.frame(
  sample_latitude_dd = c(59.0, 58.5),
  sample_longitude_dd = c(18.0, 17.5),
  sample_min_depth_m = c(5, 15),
  sample_max_depth_m = c(3, 20)
)

try(check_depth(example_data2, depth_cols = c("sample_min_depth_m", "sample_max_depth_m")))

# Return only failing rows
try(check_depth(example_data, depth_cols = "sample_depth_m", report = FALSE))
```

`check_fields` *Validate SHARK data fields for a given datatype*

Description

This function checks a SHARK data frame against the required and recommended fields defined for a specific datatype. It verifies that all required fields are present and contain non-empty values. If `level = "warning"`, it also checks for recommended fields and empty values within them.

Usage

```
check_fields(
  data,
  datatype,
  level = "error",
  stars = 1,
  bacterioplankton_subtype = "abundance",
  field_definitions = .field_definitions
)
```

Arguments

`data`	A data frame containing SHARK data to be validated.
`datatype`	A string giving the SHARK datatype to validate against. Must exist as a name in the provided `field_definitions`.
`level`	Character string, either `"error"` or `"warning"`. If `"error"`, only required fields are validated. If `"warning"`, recommended fields are also checked and reported as warnings.
`stars`	Integer. Maximum number of "" *levels to include. Default = 1 (only single* ""). For example, `stars = 2` includes "" *and "**",* `stars = 3` *includes* "", "**", and** "*".
`bacterioplankton_subtype`	Character. For "Bacterioplankton" only: either "abundance" (default) or "production". Ignored for other datatypes.
`field_definitions`	A named list of field definitions. Each element should contain two character vectors: `required` and `recommended`. Defaults to the package's built-in `SHARK4R:::.field_definitions`. Alternatively, the latest definitions can be loaded directly from the official SHARK4R GitHub repository using `load_shark4r_fields()`.

Details

Note: A single "*" marks required fields in the standard SHARK template. A double "**" is often used to specify columns required for **national monitoring only**. For more information, see: https://www.smhi.se/data/hav-och-havsmiljo/datavardskap-oceanografi-och-marinbiologi/leverera-data

Field definitions for SHARK data can be loaded in two ways:

1. **From the SHARK4R package bundle (default):** The package contains a built-in object, `.field_definitions`, which stores required and recommended fields for each datatype.
2. **From GitHub (latest official version):** To use the most up-to-date field definitions, you can load them directly from the SHARK4R-statistics repository:

```
defs <- load_shark4r_fields()
check_fields(my_data, "Phytoplankton", field_definitions = defs)
```

Delivery-format (all-caps) data: If the column names in `data` are all uppercase (e.g. SDATE), `check_fields()` assumes the dataset follows the official SHARK delivery template. In this case:

- Required fields are determined from the delivery template using `get_delivery_template()` and `find_required_fields()`.
- Recommended fields are ignored because the delivery templates do not define them.
- The function validates that all required columns exist and contain non-empty values.

This ensures that both internal `SHARK4R` datasets (with camelCase or snake_case columns) and official delivery files (ALL_CAPS columns) are validated correctly using the appropriate rules.

Stars in the template

Leading asterisks in the delivery template indicate required levels:

- * = standard required column
- * = required for national monitoring
- Other symbols = additional requirement level

The `stars` parameter in `check_fields()` controls how many levels of required columns to include.

Value

A tibble with the following columns:

level Either `"error"` or `"warning"`.

field The name of the field that triggered the check.

row Row number(s) in `data` where the issue occurred, or `NA` if the whole field is missing.

message A descriptive message explaining the problem.

The tibble will be empty if no problems are found.

See Also

`load_shark4r_fields` for fetching the latest field definitions from GitHub, `get_delivery_template` for downloading delivery templates from SMHI's website.

Examples

```
# Example 1: Using built-in field definitions for "Phytoplankton"
df_phyto <- data.frame(
  visit_date = "2023-06-01",
  sample_id = "S1",
  scientific_name = "Skeletonema marinoi",
  value = 123
)

# Check fields
check_fields(df_phyto, "Phytoplankton", level = "warning")


# Example 2: Load latest definitions from GitHub and use them
try(defs <- load_shark4r_fields(verbose = FALSE))

# Check fields using loaded field definitions
if (exists("defs")) try(check_fields(df_phyto, "Phytoplankton", field_definitions = defs))


# Example 3: Custom datatype with required + recommended fields
defs <- list(
  ExampleType = list(
    required = c("id", "value"),
    recommended = "comment"
  )
)

# Example data
df_ok <- data.frame(id = 1, value = "x", comment = "ok")

# Check fields using custom field definitions
check_fields(df_ok, "ExampleType", level = "warning", field_definitions = defs)
```

`check_logical_parameter` *General checker for parameter-specific logical rules*

Description

This function checks for logical rule violations in benthos/epibenthos data by applying a user-defined condition to values for a given parameter. It is intended to replace the old family of `check_*_*_logical()` functions.

Usage

```
check_logical_parameter(
  data,
  param_name,
  condition,
  return_df = FALSE,
  return_logical = FALSE
)
```

Arguments

`data`	A data frame. Must contain columns `parameter` and `value`.
`param_name`	Character; the name of the parameter to check.
`condition`	A function that takes a numeric vector of values and returns a logical vector (TRUE for rows considered problematic).
`return_df`	Logical. If TRUE, return a plain data.frame of problematic rows.
`return_logical`	Logical. If TRUE, return a logical vector of length nrow(data). Overrides return_df.

Value

A DT datatable, a data.frame, a logical vector, or NULL if no problems found.

Examples

```
# Example dataset
df <- dplyr::tibble(
  station_name = c("A1", "A2", "A3", "A4"),
  sample_date = as.Date("2023-05-01") + 0:3,
  sample_id = 101:104,
  parameter = c("Biomass", "Biomass", "Abundance", "Biomass"),
  value = c(5, -2, 10, 0)
)

# 1. Check that Biomass is never negative
check_logical_parameter(df, "Biomass", function(x) x < 0,  return_df = TRUE)

# 2. Same check, but return problematic rows as a data frame
check_logical_parameter(df, "Biomass", function(x) x < 0, return_df = TRUE)

# 3. Return logical vector marking problematic rows
check_logical_parameter(df, "Biomass", function(x) x < 0, return_logical = TRUE)

# 4. Check that Abundance is not zero (no problems found -> returns NULL)
abundance_check <- check_logical_parameter(df, "Abundance", function(x) x == 0)
print(abundance_check)
```

check_nominal_station *Check if stations are reported as nominal positions*

Description

This function attempts to determine whether stations in a dataset are reported using nominal positions (i.e., generic or repeated coordinates across events), rather than actual measured coordinates.

Usage

```
check_nominal_station(data, verbose = TRUE)
```

Arguments

data	A data frame containing at least the columns: `sample_date`, `station_name`, `sample_longitude_dd`, and `sample_latitude_dd`.
verbose	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Details

The function compares the number of unique sampling dates with the number of unique station coordinates.

If the number of unique sampling dates is larger than the number of unique station coordinates, the function suspects nominal station positions and issues a warning.

Value

A data frame with distinct station names and their corresponding latitude/longitude positions, if nominal positions are suspected. Otherwise, returns `NULL`.

Examples

```
df <- data.frame(
  sample_date = rep(seq.Date(Sys.Date(), by = "day", length.out = 3), each = 2),
  station_name = rep(c("ST1", "ST2"), 3),
  sample_longitude_dd = rep(c(15.0, 16.0), 3),
  sample_latitude_dd = rep(c(58.5, 58.6), 3)
)
check_nominal_station(df)
```

check_onland — *Check whether points are located on land*

Description

Identifies records whose coordinates fall on land, optionally applying a buffer to allow points near the coast.

Usage

```
check_onland(
  data,
  land = NULL,
  report = FALSE,
  buffer = 0,
  offline = FALSE,
  plot_leaflet = FALSE,
  only_bad = FALSE
)
```

Arguments

`data`	A data frame containing at least `sample_longitude_dd` and `sample_latitude_dd`. Both columns must be numeric, within valid ranges (longitude: -180 to 180, latitude: -90 to 90), and use WGS84 coordinates (EPSG:4326).
`land`	Optional `sf` object containing land polygons. Used only in offline mode.
`report`	Logical; if `TRUE`, returns a tibble listing rows on land and warnings. If `FALSE` (default), returns a subset of `data` containing only records on land.
`buffer`	Numeric; distance in meters inland for which points are still considered valid. Only used in online mode. Default is 0.
`offline`	Logical; if `TRUE`, the function uses the local cached shoreline (if available). If `FALSE` (default), the OBIS web service is queried.
`plot_leaflet`	Logical; if `TRUE`, returns a leaflet map showing points colored by whether they are on land (red) or in water (green). Default is `FALSE`.
`only_bad`	Logical; if `TRUE` and `plot_leaflet = TRUE`, only points on land (red) are plotted. Default is `FALSE`, meaning all points are plotted.

Details

The function supports both offline and online modes:

- **Offline mode** (`offline = TRUE`): uses a local simplified shoreline from a cached geopackage (`land.gpkg`). If the file does not exist, it is downloaded automatically and cached across R sessions.
- **Online mode** (`offline = FALSE`): uses the OBIS web service to determine distance to the shore.

The function assumes all coordinates are in WGS84 (EPSG:4326). Supplying coordinates in a different CRS will result in incorrect intersection tests.

Optionally, a leaflet map can be plotted. Points on land are displayed as red markers, while points in water are green. If `only_bad = TRUE`, only the red points (on land) are plotted.

Value

If `report = TRUE`, a tibble with columns:

- `field`: always `NA` (placeholder for future extension)
- `level`: `"warning"` for all flagged rows
- `row`: row numbers in `data` flagged as located on land
- `message`: description of the issue

If `report = FALSE` and `plot_leaflet = FALSE`, returns a subset of `data` with only the flagged rows. If `plot_leaflet = TRUE`, returns a leaflet map showing points on land (red) and in water (green), unless `only_bad = TRUE`, in which case only red points are plotted.

Examples

```
# Example data frame with coordinates
example_data <- data.frame(
  sample_latitude_dd = c(59.3, 58.1, 57.5),
  sample_longitude_dd = c(18.6, 17.5, 16.7)
)

# Report points on land with a 100 m buffer
try(report <- check_onland(example_data, report = TRUE, buffer = 100))
if (exists("report")) print(report)

# Plot all points colored by land/water
try(map <- check_onland(example_data, plot_leaflet = TRUE))

# Plot only bad points on land
try(map_bad <- check_onland(example_data, plot_leaflet = TRUE, only_bad = TRUE))

# Remove points on land by adding a buffer of 2000 m
try(ok <- check_onland(example_data, report = FALSE, buffer = 2000))
if (exists("ok")) print(nrow(ok))
```

`check_outliers` *General outlier check function for SHARK data*

Description

This function checks whether values for a specified parameter exceed a predefined threshold. Thresholds are provided in a dataframe (default `.threshold_values`), which should contain columns for `parameter`, `datatype`, and at least one numeric threshold column (e.g., `extreme_upper`). Only rows in `data` matching both the `parameter` and `delivery_datatype` (`datatype`) are considered. Optionally, data can be grouped by a custom column (e.g., `location_sea_basin`) when thresholds vary by group.

Usage

```
check_outliers(
  data,
  parameter,
  datatype,
  threshold_col = "extreme_upper",
  thresholds = .threshold_values,
  custom_group = NULL,
  direction = c("above", "below"),
  return_df = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	A tibble containing data in SHARK format. Must include columns: `parameter`, `value`, `delivery_datatype`, `station_name`, `sample_date`, `sample_id`, `shark_sample_id_md5`, `sample_min_depth_m`, `sample_max_depth_m`, and any custom grouping column used in `custom_group`.
`parameter`	Character. Name of the parameter to check. Must exist in both `data$parameter` and `thresholds$parameter`.
`datatype`	Character. Data type to match against `delivery_datatype` in `data` and `datatype` in `thresholds`.
`threshold_col`	Character. Name of the threshold column in `thresholds` to use for comparison. Defaults to `"extreme_upper"`. Other columns (e.g., `"min"`, `"Q1"`, `"median"`, `"max"`, `"mild_upper"`, etc.) can also be used if present.
`thresholds`	A tibble/data frame of thresholds. Must include columns `parameter`, `datatype`, and at least one numeric threshold column. Defaults to `.threshold_values`.
`custom_group`	Character or NULL. Optional column name in `data` and `thresholds` for grouping (e.g., `"location_sea_basin"`). If specified, thresholds are matched by group as well as `parameter` and `datatype`.
`direction`	Character. Either `"above"` (flag values above threshold) or `"below"` (flag values below threshold). Default is `"above"`.
`return_df`	Logical. If TRUE, returns a plain data.frame of flagged rows instead of a DT datatable. Default = FALSE.
`verbose`	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Details

- Only rows in data matching both parameter and delivery_datatype are checked.
- If custom_group is specified, thresholds are applied per group.
- If threshold_col does not exist in thresholds, the function stops with a warning.
- Values exceeding (or below) the threshold are flagged as outliers.
- Intended for interactive use in Shiny apps where threshold_col can be selected dynamically.

Value

If outliers are found, returns a DT::datatable or a data.frame (if return_df = TRUE) containing: datatype, station_name, sample_date, sample_id, parameter, value, threshold, and custom_group if specified. Otherwise, prints a message indicating that values are within the threshold range (if verbose = TRUE) and returns invisible(NULL).

See Also

get_shark_statistics() for preparing updated threshold data.

Examples

```
# Minimal example dataset
example_data <- dplyr::tibble(
  station_name = c("S1", "S2"),
  sample_date = as.Date(c("2025-01-01", "2025-01-02")),
  sample_id = 1:2,
  shark_sample_id_md5 = letters[1:2],
  sample_min_depth_m = c(0, 5),
  sample_max_depth_m = c(1, 6),
  parameter = c("Param1", "Param1"),
  value = c(5, 12),
  delivery_datatype = c("TypeA", "TypeA")
)

example_thresholds <- dplyr::tibble(
  parameter = "Param1",
  datatype = "TypeA",
  extreme_upper = 10,
  mild_upper = 8
)

# Check for values above "extreme_upper"
check_outliers(
  data = example_data,
  parameter = "Param1",
  datatype = "TypeA",
  threshold_col = "extreme_upper",
  thresholds = example_thresholds,
  return_df = TRUE
)
```

```
# Check for values above "mild_upper"
check_outliers(
  data = example_data,
  parameter = "Param1",
  datatype = "TypeA",
  threshold_col = "mild_upper",
  thresholds = example_thresholds,
  return_df = TRUE
)
```

`check_parameter_rules` *Check parameter values against logical rules*

Description

Applies parameter-specific and row-wise logical rules to benthos/epibenthos data, flagging measurements that violate defined conditions. This function replaces multiple deprecated `check_*_logical()` functions with a general, flexible implementation.

Usage

```
check_parameter_rules(
  data,
  param_conditions = get(".param_conditions", envir = asNamespace("SHARK4R")),
 rowwise_conditions = get(".rowwise_conditions", envir = asNamespace("SHARK4R")),
  return_df = FALSE,
  return_logical = FALSE,
  verbose = TRUE
)
```

Arguments

`data` A data frame containing at least the columns `parameter` and `value`.

`param_conditions`
A named list of parameter-specific rules. Each element should be a list with:

condition Function taking a numeric vector and returning a logical vector (TRUE = violation).

range_msg Character string describing the expected range.

Defaults to `SHARK4R:::.param_conditions` defined in the package namespace.

`rowwise_conditions`
A named list of row-wise rules applied across multiple parameters. Each element should be a function taking the full data frame and returning a logical vector. Defaults to `SHARK4R:::.rowwise_conditions` defined in the package namespace.

`return_df` Logical. If TRUE, problematic rows are returned as plain `data.frames`.

`return_logical`	Logical. If TRUE, problematic rows are returned as logical vectors. Overrides `return_df`.
`verbose`	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Details

This function evaluates each parameter in `param_conditions` and `rowwise_conditions`. Only parameters present in the dataset are checked. Messages are printed indicating whether values are within expected ranges or which rows violate rules.

Value

A named list of results for each parameter:

Logical vector If `return_logical = TRUE`.

Data frame If `return_df = TRUE` and violations exist.

DT datatable If violations exist and `return_df = FALSE`.

NULL If no violations exist for the parameter.

Invisible return.

Examples

```
df <- data.frame(
  station_name = c("A1", "A2", "A3", "A4"),
  sample_date = as.Date("2023-05-01") + 0:3,
  sample_id = 101:104,
  parameter = c("Wet weight", "Wet weight", "Abundance", "BQIm"),
  value = c(0, 5, 0, 3)
)

# Check against default package rules
check_parameter_rules(df)

# Return problematic rows as data.frame
check_parameter_rules(df, return_df = TRUE)

# Return logical vectors for each parameter
rule_check <- check_parameter_rules(df, return_logical = TRUE)
print(rule_check)
```

check_setup — *Download and set up SHARK4R support files*

Description

This function downloads the `products` folder from the SHARK4R GitHub repository and places them in a user-specified directory. These folders contain Shiny applications and R Markdown documents used for quality control (QC) of SHARK data.

Usage

```
check_setup(path, run_app = FALSE, force = FALSE, verbose = TRUE)
```

Arguments

`path`	Character string giving the directory where the products folder should be created. Must be provided by the user.
`run_app`	Logical, if `TRUE` runs the QC Shiny app located in the `products` folder after setup. Default is `FALSE`.
`force`	Logical, if `TRUE` forces a re-download and overwrites existing folder. Default is `FALSE`.
`verbose`	Logical, if `TRUE` prints progress messages. Default is `TRUE`.

Details

If the `path` folders already exist, the download will be skipped unless `force = TRUE` is specified. Optionally, the function can launch the QC Shiny app directly after setup.

Value

An (invisible) list with the path to the local `products` folder:

Examples

```
# Download support files into a temporary directory
try(check_setup(path = tempdir()))

# Force re-download if already present
try(check_setup(path = tempdir(), force = TRUE))

# Download and run the QC Shiny app
if(interactive()){
 try(check_setup(path = tempdir(), run_app = TRUE))
}
```

check_station_distance

Check station distances against SMHI station list

Description

Matches reported station names against the SMHI curated station list ("station.txt") and checks whether matched stations fall within pre-defined distance limits. This helps ensure that station assignments are spatially consistent.

Usage

```
check_station_distance(
  data,
  station_file = NULL,
  plot_leaflet = FALSE,
  try_synonyms = TRUE,
  fallback_crs = 4326,
  only_bad = FALSE,
  verbose = TRUE
)
```

Arguments

data	A data frame containing at least the columns: station_name, sample_longitude_dd, sample_latitude_dd.
station_file	Optional path to a custom station file (tab-delimited). If NULL (default), the function will first attempt to use the NODC_CONFIG environment variable, and if that fails, will use the bundled "station.zip" from the SHARK4R package.
plot_leaflet	Logical; if TRUE, displays a leaflet map with SMHI stations (blue circles with radius in popup) and reported stations (green/red/gray markers). Default is FALSE.
try_synonyms	Logical; if TRUE (default), unmatched station names are also compared against the SYNONYM_NAMES column in the station database.
fallback_crs	Integer; CRS (EPSG code) to use when creating spatial points if no CRS is available. Defaults to 4326 (WGS84). Change this if your coordinates are reported in another CRS (e.g., 3006 for SWEREF99 TM).
only_bad	Logical; if TRUE, the leaflet map will only display stations that are outside the allowed radius (red markers). Default is FALSE.
verbose	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Details

Optionally, a leaflet map of stations can be plotted. SMHI stations that match the reported data are shown as blue circles, with their allowed radius visualized and displayed in the popup (e.g., "ST1

(Radius: 1000 m)"). Reported stations are shown as markers colored by whether they fall within the radius (green), outside the radius (red), or unmatched (gray).

If `try_synonyms = TRUE`, the function will attempt a second match using the `SYNONYM_NAMES` column in the station database, splitting multiple synonyms separated by `<or>`.

The function first checks if a station file path is provided via the `station_file` argument. If not, it looks for the `NODC_CONFIG` environment variable. This variable can point to a folder where the NODC (Swedish National Oceanographic Data Center) configuration and station file are stored, typically including:

- `<NODC_CONFIG>/config/station.txt`

If `NODC_CONFIG` is set and the folder exists, the function will use `station.txt` from that location. Otherwise, it falls back to the bundled `station.zip` included in the `SHARK4R` package.

Value

If `plot_leaflet = FALSE`, returns a data frame with columns:

station_name Reported station name.

match_type `TRUE` if station matched in SMHI list, `FALSE` otherwise.

distance_m Distance in meters from reported station to matched SMHI station.

within_limit `TRUE` if distance <= allowed radius, `FALSE` if outside, `NA` if unmatched.

If `plot_leaflet = TRUE`, the function produces a leaflet map showing:

- Blue circles for SMHI stations with radius in the popup.
- Reported stations colored by status: green (within radius), red (outside radius), gray (unmatched).
- If `only_bad = TRUE`, only the red stations (outside radius) are displayed.

Examples

```
# Example data
df <- data.frame(
  station_name = c("ANHOLT E", "BY5 BORNHOLMSDJ", "NEW STATION"),
  sample_longitude_dd = c(12.1, 15.97, 17.5),
  sample_latitude_dd  = c(56.7, 55.25, 58.7)
)

# Check station distance
try(check_station_distance(df, try_synonyms = TRUE, verbose = FALSE))

# Plot bad points in leaflet map
try(map <- check_station_distance(df,
                              plot_leaflet = TRUE,
                              only_bad = TRUE,
                              verbose = FALSE))
```

check_value_logical *Identify non-numeric or non-logical values in measurement data*

Description

This function checks whether entries in the `value` column of a dataset are valid numeric or logical values. It is particularly useful for identifying common data entry errors such as inequality symbols (<, >) or unintended text strings (e.g., "NA", "below detection"). The function reports any invalid entries in an interactive `DT::datatable` for easy inspection.

Usage

```
check_value_logical(data, return_df = FALSE)
```

Arguments

`data`	A data frame. Must contain a column named `value`.
`return_df`	Logical. If TRUE, return a plain data.frame of problematic rows instead of a DT datatable. Default = FALSE.

Value

A `DT::datatable` or data frame listing unique invalid entries, or `NULL` (invisibly) if all values are correctly formatted as numeric or logical.

Examples

```
# Example dataset with mixed valid and invalid values
df <- data.frame(
  station_name = c("A", "B", "C", "D", "E"),
  value = c("3.4", "<0.2", "TRUE", "NA", "5e-3")
)

# Check for invalid (non-numeric / non-logical) entries
check_value_logical(df, return_df = TRUE)

# Example with all valid numeric and logical values
df_valid <- data.frame(value = c(1.2, 0, TRUE, FALSE, 3.5))
check_value_logical(df_valid)
```

`check_zero_positions` *Identify samples with zero-valued station coordinates*

Description

This function inspects a dataset containing sample coordinates to identify potential issues where longitude or latitude values are zero (0), which typically indicate missing or erroneous station positions. The function can return a summary table, a filtered data frame, or a logical vector highlighting problematic rows. It is useful as a data quality control step before spatial analyses or database imports.

Usage

```
check_zero_positions(
  data,
  coord = "longitude",
  return_df = FALSE,
  return_logical = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	A data frame. Must contain `sample_longitude_dd` and/or `sample_latitude_dd`.
`coord`	Character. Which coordinate(s) to check: "longitude", "latitude", or "both". Default = "longitude".
`return_df`	Logical. If TRUE, return a plain data.frame of problematic rows instead of a DT datatable. Default = FALSE.
`return_logical`	Logical. If TRUE, return a logical vector of length nrow(data) indicating which rows have zero in the selected coordinate(s). Overrides return_df. Default = FALSE.
`verbose`	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Value

A DT datatable, a data.frame, a logical vector, or NULL (if no problems found and return_logical = FALSE).

Examples

```
# Example data
df <- data.frame(
  station_name = c("A", "B", "C"),
  sample_longitude_dd = c(15.2, 0, 18.7),
  sample_latitude_dd = c(56.3, 58.1, 0)
)
```

```
# Check for zeroes in both coordinates and return as data.frame
check_zero_positions(df, coord = "both", return_df = TRUE)

# Return a logical vector instead of a table
check_zero_positions(df, coord = "both", return_logical = TRUE)
```

`check_zero_value` *Identify records with zero-valued measurement data*

Description

This function scans a dataset for cases where the measurement column (`value`) contains zero (0) values, which may indicate missing, censored, or erroneous data. It returns either a `DT::datatable` for easy inspection or a plain `data.frame` of the affected rows. This function is useful for quality control and validation prior to data aggregation, reporting, or database submission.

Usage

```
check_zero_value(data, return_df = FALSE)
```

Arguments

`data`	A data frame. Must contain a column named `value`.
`return_df`	Logical. If TRUE, return a plain data.frame of problematic rows instead of a DT datatable. Default = FALSE.

Value

A DT datatable or a data.frame of zero-value records, or `NULL` (invisibly) if no zero values are found.

Examples

```
# Example dataset
df <- data.frame(
  station_name = c("A", "B", "C", "D"),
  sample_date = as.Date(c("2023-06-01", "2023-06-02", "2023-06-03", "2023-06-04")),
  value = c(3.2, 0, 1.5, 0)
)

# Return a plain data.frame of zero-value records
check_zero_value(df, return_df = TRUE)
```

clean_shark4r_cache *Clean SHARK4R cache by file age and session*

Description

Deletes cached files in the SHARK4R cache directory that are older than a specified number of days.

Usage

```
clean_shark4r_cache(
  days = 1,
  cache_dir = NULL,
  clear_perm_cache = FALSE,
  search_pattern = NULL,
  verbose = TRUE
)
```

Arguments

`days`	Numeric; remove files older than this number of days. Default is 1.
`cache_dir`	Character; path to the cache directory to clean. Defaults to the package cache directory in the user-specific R folder (via the internal `cache_dir()` helper). You can override this parameter to specify a custom cache location.
`clear_perm_cache`	Logical. If `TRUE`, files that are cached across R sessions are cleared, i.e. geographical shape files. Defaults to `FALSE`.
`search_pattern`	Character; optional regex pattern to filter which files to consider for deletion.
`verbose`	Logical. If `TRUE`, displays messages of cache cleaning progress. Defaults to `TRUE`.

Details

The cache is automatically cleared for files older than 24 hours. Files in the `perm` subdirectory are not removed automatically and must be cleared explicitly using `clear_perm_cache = TRUE`.

Value

Invisible `NULL`. Messages are printed about what was deleted and whether the in-memory session cache was cleared.

See Also

`get_peg_list()`, `get_nomp_list()`, `get_shark_codes()`, `get_dyntaxa_dwca()`, `get_shark_statistics()` for functions that populate the cache.

Examples

```
# Remove files older than 60 days and clear session cache
try(clean_shark4r_cache(days = 60))
```

`construct_dyntaxa_table` *Construct a hierarchical taxonomy table from Dyntaxa*

Description

This function constructs a taxonomy table based on Dyntaxa taxon IDs. It queries the SLU Artdatabanken API (Dyntaxa) to fetch taxonomy information and organizes the data into a hierarchical table.

Usage

```
construct_dyntaxa_table(
  taxon_ids,
  subscription_key = Sys.getenv("DYNTAXA_KEY"),
  shark_output = TRUE,
  add_parents = TRUE,
  add_descendants = FALSE,
  add_descendants_rank = "genus",
  add_synonyms = TRUE,
  add_missing_taxa = FALSE,
  add_hierarchy = FALSE,
  verbose = TRUE,
  add_genus_children = deprecated(),
  recommended_only = deprecated(),
  parent_ids = deprecated()
)
```

Arguments

`taxon_ids`	An integer vector containing taxon IDs for which taxonomy information is requested. These IDs should correspond to specific taxonomic entities within the Dyntaxa database.
`subscription_key`	A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`. You can provide the key in three ways: • **Directly as a parameter**: `construct_dyntaxa_table(238366, subscription_key = "your_key_here")`. • **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.

- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`shark_output`	Logical. If TRUE, the function will return a table formatted with SHARK-compatible columns. If FALSE, all available columns are returned. Default is TRUE.
`add_parents`	Logical. If TRUE, the function will include parent taxa (higher ranks) for the specified taxon IDs in the output. Default is TRUE.
`add_descendants`	Logical. If TRUE, the output will include descendant taxa (lower ranks) for the specified taxon IDs and the rank specified in `add_descendants_rank`. Default is FALSE.
`add_descendants_rank`	Character string specifying the rank of descendant taxa to include. Allowed values are "kingdom", "phylum", "class", "order", "family", "genus", and "species". Default is "genus".
`add_synonyms`	Logical. If TRUE, the function will include synonyms for the accepted taxa in the output. Default is TRUE.
`add_missing_taxa`	Logical. If TRUE, the function will attempt to fetch missing taxa (i.e., taxa not found in the initial Dyntaxa DwC-A query, such as species complexes). Default is FALSE.
`add_hierarchy`	Logical. If TRUE, the function will add a hierarchy column indicating the taxonomic relationships (e.g., parent-child) among the taxa. Default is FALSE.
`verbose`	Logical. If TRUE, the function will print additional messages to provide feedback on its progress. Default is TRUE.
`add_genus_children`	**[Deprecated]** Use `add_descendants` instead.
`recommended_only`	**[Deprecated]** Use `add_synonyms` instead.
`parent_ids`	**[Deprecated]** Use `taxon_ids` instead. `construct_dyntaxa_table` now handles taxon IDs.

Details

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Note: Please review the API conditions and register for access before using the API. Data collected through the API is stored at SLU Artdatabanken. Please also note that the authors of `SHARK4R` are not affiliated with SLU Artdatabanken.

Value

A `tibble` representing the constructed taxonomy table.

See Also

`get_worms_taxonomy_tree` for an equivalent WoRMS function

SLU Artdatabanken API Documentation

Examples

```
## Not run:
# Construct Dyntaxa taxonomy table for taxon IDs 238366 and 1010380
taxon_ids <- c(238366, 1010380)
taxonomy_table <- construct_dyntaxa_table(taxon_ids, "your_subscription_key")
print(taxonomy_table)

## End(Not run)
```

`convert_ddmm_to_dd` *Convert coordinates from DDMM format to decimal degrees*

Description

This function converts geographic coordinates provided in the DDMM format (degrees and minutes) to decimal degrees. It can handle:

- DDMM (e.g., 5733 to 57°33' to 57.55°)
- DDMMss or DDMMss... (extra digits after minutes are interpreted as fractional minutes, e.g., 573345 to 57°33.45' to 57.5575°)

Usage

```
convert_ddmm_to_dd(coord)
```

Arguments

`coord`	A numeric or character vector of coordinates in DDMM format.

Details

Non-numeric characters are removed before conversion. Coordinates shorter than 4 digits are returned as `NA`.

Value

A numeric vector of decimal degrees corresponding to the input coordinates. Names from the input vector are removed.

Examples

```
# Basic DDMM input
convert_ddmm_to_dd(c(5733, 6045))
# Input with fractional minutes
convert_ddmm_to_dd(c("573345", "604523"))
# Input with non-numeric characters
convert_ddmm_to_dd(c("57°33'", "60°45'23\""))
```

`find_required_fields` *Find required fields in a SHARK delivery template*

Description

Identifies which columns are mandatory in the SHARK delivery template based on rows starting with "*" (one or more). You can specify how many levels of asterisks to include.

Usage

```
find_required_fields(
  datatype,
  stars = 1,
  bacterioplankton_subtype = "abundance"
)
```

Arguments

`datatype` Character. The datatype name. Available options include:

- "Bacterioplankton" (subtypes: "abundance", "production")
- "Chlorophyll"
- "Epibenthos" (dive transect)
- "Dropvideo" (epibenthos drop video)
- "Grey seal"
- "Harbour seal"
- "Ringed seal"
- "Harbour Porpoise"
- "Physical and Chemical"
- "Primary production"
- "Phytoplankton"
- "Picoplankton"
- "Sedimentation"
- "Seal pathology"
- "Profile"
- "Zooplankton"

- "Zoobenthos"

`stars` Integer. Maximum number of "" *levels to include. Default = 1 (only single* ""). For example, `stars = 2` includes "" *and "**", `stars = 3` includes* "", "**", and** "*".

`bacterioplankton_subtype`
Character. For "Bacterioplankton" only: either "abundance" (default) or "production". Ignored for other datatypes.

Details

Note: A single "*" marks required fields in the standard SHARK template. A double "**" is often used to specify columns required for **national monitoring only**. For more information, see: https://www.smhi.se/data/hav-och-havsmiljo/datavardskap-oceanografi-och-marinbiologi/leverera-data

Value

A character vector of column names that are required in the template.

Examples

```
# Only single "*" required columns
try(find_required_fields("Bacterioplankton"))

# Include both "*" and "**" required columns (national monitoring too)
try(find_required_fields("Bacterioplankton", stars = 2))

# Include up to three levels of "*"
try(find_required_fields("Phytoplankton", stars = 3))
```

`get_delivery_template` *Get a delivery template for a SHARK datatype*

Description

Downloads and reads the SHARK Excel delivery template for a given datatype. The template contains the column definitions and headers used for submission.

Usage

```
get_delivery_template(
  datatype,
  sheet = "Kolumner",
  header_row = 4,
  skip = 1,
  bacterioplankton_subtype = "abundance",
  force = FALSE,
  clean_cache_days = 1
)
```

Arguments

`datatype`	Character. The datatype name. Available options include: • "Bacterioplankton" (subtypes: "abundance", "production") • "Chlorophyll" • "Epibenthos" (dive transect) • "Dropvideo" (epibenthos drop video) • "Grey seal" • "Harbour seal" • "Ringed seal" • "Harbour Porpoise" • "Physical and Chemical" • "Primary production" • "Phytoplankton" • "Picoplankton" • "Sedimentation" • "Seal pathology" • "Profile" • "Zooplankton" • "Zoobenthos"
`sheet`	Character or numeric. Name (e.g., "Kolumner") or index (e.g., 1) of the sheet in the Excel file to read. Default is "Kolumner".
`header_row`	Integer. Row number in the Excel file that contains the column headers. Default is 4.
`skip`	Integer. Number of rows to skip before reading data. Default is 1.
`bacterioplankton_subtype`	Character. For "Bacterioplankton" only: either "abundance" (default) or "production". Ignored for other datatypes.
`force`	Logical; if `TRUE`, forces re-download even if cached copy exists.
`clean_cache_days`	Numeric; if not `NULL`, cached template files older than this number of days are deleted automatically. Default is 1.

Value

A `tibble` containing the delivery template. Column names are set from the header row.

Examples

```
# Bacterioplankton abundance
try(abun <- get_delivery_template("Bacterioplankton",
                              bacterioplankton_subtype = "abundance"))

if (exists("abun")) print(abun)
```

```
# Bacterioplankton production
try(prod <- get_delivery_template("Bacterioplankton",
                                  bacterioplankton_subtype = "production"))

# Phytoplankton template
try(phyto <- get_delivery_template("Phytoplankton"))

# Phytoplankton column explanation (sheet number 3)
try(phyto_column_explanation <- get_delivery_template("Phytoplankton",
                                                      sheet = 3,
                                                      header_row = 4,
                                                      skip = 3))

if (exists("phyto_column_explanation")) print(phyto_column_explanation)
```

get_dyntaxa_dwca *Download and read Darwin Core Archive files from Dyntaxa*

Description

This function downloads a complete Darwin Core Archive (DwCA) of Dyntaxa from the SLU Artdatabanken API, extracts the archive, and reads the specified CSV file into R.

Usage

```
get_dyntaxa_dwca(
  subscription_key = Sys.getenv("DYNTAXA_KEY"),
  file_to_read = "Taxon.csv",
  force = FALSE,
  verbose = TRUE
)
```

Arguments

`subscription_key`
A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`.

You can provide the key in three ways:

- **Directly as a parameter**: `get_dyntaxa_dwca(subscription_key = "your_key_here")`.
- **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`file_to_read`	A string specifying the name of the CSV file to read from the extracted archive. Allowed options are: `"Reference.csv"`, `"SpeciesDistribution.csv"`, `"Taxon.csv"`, or `"VernacularName.csv"`. Defaults to `"Taxon.csv"`.
`force`	A logical value indicating whether to force a fresh download of the archive, even if a cached copy is available. Defaults to `FALSE`.
`verbose`	A logical value indicating whether to show download progress. Defaults to `TRUE`.

Details

By default, the archive is downloaded and cached across R sessions. On subsequent calls, the function reuses the cached copy of the extracted files to avoid repeated downloads. Use the `force` parameter to re-download the archive if needed. The cache is cleared automatically after 24 hours, but you can also manually clear it using `clean_shark4r_cache`.

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Note: Please review the API conditions and register for access before using the API. Data collected through the API is stored at SLU Artdatabanken. Please also note that the authors of `SHARK4R` are not affiliated with SLU Artdatabanken.

Value

A tibble containing the data from the specified CSV file.

See Also

`clean_shark4r_cache()` to manually clear cached files.

Examples

```
## Not run:
# Provide your Dyntaxa API subscription key
subscription_key <- "your_subscription_key"

# Download and read the Taxon.csv file
taxon_data <- get_dyntaxa_dwca(subscription_key, file_to_read = "Taxon.csv")

## End(Not run)
```

`get_dyntaxa_parent_ids` *Get parent taxon IDs for specified taxon IDs from Dyntaxa*

Description

This function queries the SLU Artdatabanken API (Dyntaxa) to retrieve parent taxon IDs for the specified taxon IDs. It constructs a request with the provided taxon IDs, sends the request to the SLU Artdatabanken API, and processes the response to return a list of parent taxon IDs.

Usage

```
get_dyntaxa_parent_ids(
  taxon_ids,
  subscription_key = Sys.getenv("DYNTAXA_KEY"),
  verbose = TRUE
)
```

Arguments

`taxon_ids` A vector of numeric taxon IDs for which parent taxon IDs are requested.

`subscription_key`
A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`.

You can provide the key in three ways:

- **Directly as a parameter**: `get_dyntaxa_parent_ids(238366, subscription_key = "your_key_here")`.
- **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`verbose` Logical. Default is TRUE.

Details

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Note: Please review the API conditions and register for access before using the API. Data collected through the API is stored at SLU Artdatabanken. Please also note that the authors of `SHARK4R` are not affiliated with SLU Artdatabanken.

Value

A list containing parent taxon IDs corresponding to the specified taxon IDs.

See Also

SLU Artdatabanken API Documentation

Examples

```
## Not run:
# Get parent taxon IDs for taxon IDs 238366 and 1010380
parent_ids <- get_dyntaxa_parent_ids(c(238366, 1010380), "your_subscription_key")
print(parent_ids)

## End(Not run)
```

`get_dyntaxa_records` *Get taxonomic information from Dyntaxa for specified taxon IDs*

Description

This function queries the SLU Artdatabanken API (Dyntaxa) to retrieve taxonomic information for the specified taxon IDs. It constructs a request with the provided taxon IDs, sends the request to the SLU Artdatabanken API, and processes the response to return taxonomic information in a data frame.

Usage

```
get_dyntaxa_records(taxon_ids, subscription_key = Sys.getenv("DYNTAXA_KEY"))
```

Arguments

`taxon_ids` A vector of numeric taxon IDs (Dyntaxa ID) for which taxonomic information is requested.

`subscription_key`
A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`.

You can provide the key in three ways:

- **Directly as a parameter**: `get_dyntaxa_records(238366, subscription_key = "your_key_here")`.
- **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

Details

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Note: Please review the API conditions and register for access before using the API. Data collected through the API is stored at SLU Artdatabanken. Please also note that the authors of `SHARK4R` are not affiliated with SLU Artdatabanken.

Value

A `tibble` containing taxonomic information for the specified taxon IDs. Columns include `taxonId`, `names`, `category`, `rank`, `isRecommended`, and `parentTaxonId`.

See Also

SLU Artdatabanken API Documentation

Examples

```
## Not run:
# Get taxonomic information for taxon IDs 238366 and 1010380
taxon_info <- get_dyntaxa_records(c(238366, 1010380), "your_subscription_key")
print(taxon_info)

## End(Not run)
```

`get_hab_list` *Download the IOC-UNESCO Taxonomic Reference List of Harmful Microalgae*

Description

This function retrieves the IOC-UNESCO Taxonomic Reference List of Harmful Microalgae (Lundholm et al. 2009) from the World Register of Marine Species (WoRMS). The data is returned as a dataframe, with options to customize the fields included in the download.

Usage

```
get_hab_list(
  species_only = TRUE,
  harmful_non_toxic_only = FALSE,
  aphia_id = TRUE,
  scientific_name = TRUE,
  authority = TRUE,
  fossil = TRUE,
  rank_name = TRUE,
  status_name = TRUE,
  qualitystatus_name = TRUE,
  modified = TRUE,
  lsid = TRUE,
  parent_id = TRUE,
  stored_path = TRUE,
  citation = TRUE,
  classification = TRUE,
  environment = TRUE,
  accepted_taxon = TRUE,
  verbose = TRUE
)
```

Arguments

`species_only`	Logical. If `TRUE`, only species-level records are returned (i.e., rows where the `Species` column is not `NA`). Note that this filter is only applied when `harmful_non_toxic_only = FALSE`; it is ignored when `harmful_non_toxic_only = TRUE`.
`harmful_non_toxic_only`	Logical. If `TRUE`, retrieves only non-toxigenic marine microalgal species flagged with harmful effects. Defaults to `FALSE`. **[Experimental]**
`aphia_id`	Logical. Include the AphiaID field. Defaults to `TRUE`.
`scientific_name`	Logical. Include the scientific name field. Defaults to `TRUE`.
`authority`	Logical. Include the authority field. Defaults to `TRUE`.
`fossil`	Logical. Include information about fossil status. Defaults to `TRUE`.
`rank_name`	Logical. Include the taxonomic rank (e.g., species, variety, forma). Defaults to `TRUE`.
`status_name`	Logical. Include the taxonomic status field. Defaults to `TRUE`.
`qualitystatus_name`	Logical. Include the quality status field. Defaults to `TRUE`.
`modified`	Logical. Include the date of last modification field. Defaults to `TRUE`.
`lsid`	Logical. Include the Life Science Identifier (LSID) field. Defaults to `TRUE`.
`parent_id`	Logical. Include the parent AphiaID field. Defaults to `TRUE`.
`stored_path`	Logical. Include the stored path field. Defaults to `TRUE`.
`citation`	Logical. Include citation information. Defaults to `TRUE`.
`classification`	Logical. Include the full taxonomic classification (e.g., kingdom, phylum, class). Defaults to `TRUE`.
`environment`	Logical. Include environmental data (e.g., marine, brackish, freshwater, terrestrial). Defaults to `TRUE`.
`accepted_taxon`	Logical. Include information about the accepted taxon (e.g., scientific name and authority). Defaults to `TRUE`.
`verbose`	Logical. Whether to display progress information. Default is `TRUE`.

Details

This function submits a POST request to the WoRMS database to retrieve the IOC-UNESCO Taxonomic Reference List of Harmful Microalgae. The downloaded data can include various fields, which are controlled by the input parameters. If a field is not required, set the corresponding parameter to `FALSE` to exclude it from the output.

Value

A `tibble` containing the HABs taxonomic list, with columns based on the selected parameters.

References

Lundholm, N.; Bernard, C.; Churro, C.; Escalera, L.; Hoppenrath, M.; Iwataki, M.; Larsen, J.; Mertens, K.; Murray, S.; Probert, I.; Salas, R.; Tillmann, U.; Zingone, A. (Eds) (2009 onwards). IOC-UNESCO Taxonomic Reference List of Harmful Microalgae. https://www.marinespecies.org/hab/. doi:10.14284/362

See Also

`https://www.marinespecies.org/hab/` for IOC-UNESCO Taxonomic Reference List of Harmful Microalgae

Examples

```
# Download the default HABs taxonomic list
try(habs_taxlist_df <- get_hab_list())
if (exists("habs_taxlist_df")) head(habs_taxlist_df)

# Include higher taxa records
try(habs_taxlist_df <- get_hab_list(species_only = FALSE))
if (exists("habs_taxlist_df")) head(habs_taxlist_df)

# Retrieve only non-toxigenic harmful species (experimental stage)
try(habs_taxlist_df <- get_hab_list(harmful_non_toxic_only = TRUE, verbose = FALSE))
if (exists("habs_taxlist_df")) head(habs_taxlist_df)

# Include only specific fields in the output
try(habs_taxlist_df <- get_hab_list(aphia_id = TRUE, scientific_name = TRUE, authority = FALSE))
if (exists("habs_taxlist_df")) head(habs_taxlist_df)
```

`get_nomp_list` *Get the latest NOMP biovolume Excel list*

Description

This function downloads the latest available Nordic Marine Phytoplankton Group (NOMP) biovolume zip archive from SMHI, unzips it, and reads the first Excel file by default. You can also specify which file in the archive to read.

Usage

```
get_nomp_list(
  year = as.numeric(format(Sys.Date(), "%Y")),
  file = NULL,
  sheet = NULL,
  force = FALSE,
  base_url = NULL,
  clean_cache_days = 30,
  verbose = TRUE
)
```

Arguments

`year`	Numeric year to download. Default is current year; if not available, previous years are automatically tried.
`file`	Character string specifying which file in the zip archive to read. Defaults to the first Excel file in the archive.
`sheet`	Character or numeric; the name or index of the sheet to read from the Excel file. If neither argument specifies the sheet, defaults to the first sheet.
`force`	Logical; if `TRUE`, forces re-download of the zip file even if cached copy exists.
`base_url`	Base URL (without "/nomp_taxa_biovolumes_and_carbon_YYYY.zip") for the NOMP biovolume files. Defaults to the SMHI directory.
`clean_cache_days`	Numeric; if not `NULL`, cached NOMP zip files older than this number of days will be automatically deleted and replaced by a new download. Defaults to 30. Set to `NULL` to disable automatic cleanup.
`verbose`	A logical indicating whether to print progress messages. Default is TRUE.

Value

A tibble with the contents of the requested Excel file.

See Also

`clean_shark4r_cache()` to manually clear cached files.

Examples

```
# Get the latest available list
try(nomp_list <- get_nomp_list())
if (exists("nomp_list")) head(nomp_list)

# Get the 2023 list and clean old cache files older than 60 days
try(nomp_list_2023 <- get_nomp_list(2023, clean_cache_days = 60))
if (exists("nomp_list_2023")) head(nomp_list_2023)
```

get_nua_external_links

Retrieve external links or facts for taxa from Nordic Microalgae

Description

This function retrieves external links related to algae taxa from the Nordic Microalgae API. It takes a vector of slugs (taxon identifiers) and returns a data frame containing the external links associated with each taxon. The data includes the provider, label, external ID, and the URL of the external link.

Usage

```
get_nua_external_links(slug, verbose = TRUE, unparsed = FALSE)
```

Arguments

`slug`	A vector of taxon slugs (identifiers) for which to retrieve external links.
`verbose`	A logical flag indicating whether to display a progress bar. Default is `TRUE`.
`unparsed`	Logical. If `TRUE`, the API response with all facts is returned as an unparsed list. Default is `FALSE`.

Details

The slugs (taxon identifiers) used in this function can be retrieved using the `get_nua_taxa()` function, which returns a data frame with a column for taxon slugs, along with other relevant metadata for each taxon.

Value

When unparsed = `FALSE`: a `tibble` containing the following columns:

`slug`	The slug (identifier) of the taxon.
`provider`	The provider of the external link.
`label`	The label of the external link.
`external_id`	The external ID associated with the external link.
`external_url`	The URL of the external link.
`collection`	The collection category, which is "External Links" for all rows.

See Also

https://nordicmicroalgae.org/ for Nordic Microalgae website.

https://nordicmicroalgae.org/api/ for Nordic Microalgae API documentation.

Examples

```
# Retrieve external links for a vector of slugs
slugs <- c("chaetoceros-debilis", "alexandrium-tamarense")
try(external_links <- get_nua_external_links(
  slug = slugs, verbose = FALSE
))
if (exists("external_links")) head(external_links)
```

`get_nua_harmfulness` *Retrieve harmfulness for taxa from Nordic Microalgae*

Description

This function retrieves harmfulness information related to algae taxa from the Nordic Microalgae API. It takes a vector of slugs (taxon identifiers) and returns a data frame containing the harmfulness information associated with each taxon. The data includes the provider, label, external ID, and the URL of the external link.

Usage

```
get_nua_harmfulness(slug, verbose = TRUE)
```

Arguments

`slug`	A vector of taxon slugs (identifiers) for which to retrieve external links.
`verbose`	A logical flag indicating whether to display a progress bar. Default is `TRUE`.

Details

The slugs (taxon identifiers) used in this function can be retrieved using the `get_nua_taxa()` function, which returns a data frame with a column for taxon slugs, along with other relevant metadata for each taxon.

Value

A `tibble` containing the following columns:

`slug`	The slug (identifier) of the taxon.
`provider`	The provider of the external link.
`label`	The label of the external link.
`external_id`	The external ID associated with the external link.
`external_url`	The URL of the external link.
`collection`	The collection category, which is "Harmful algae blooms" for all rows.

See Also

https://nordicmicroalgae.org/ for Nordic Microalgae website.

https://nordicmicroalgae.org/api/ for Nordic Microalgae API documentation.

Examples

```
# Retrieve external links for a vector of slugs
try(harmfulness <- get_nua_harmfulness(slug = c("dinophysis-acuta",
                                              "alexandrium-ostenfeldii"),
                                     verbose = FALSE))
if (exists("harmfulness")) print(harmfulness)
```

---

get_nua_image_labeling_links

Retrieve image labeling media links from Nordic Microalgae

Description

This function retrieves media URLs for automated imaging images from the Nordic Microalgae API. These are images from automated imaging instruments (e.g., IFCB) used for image labeling purposes. It returns URLs for different renditions (large, medium, original, small) along with basic attribution.

Usage

```
get_nua_image_labeling_links(unparsed = FALSE)
```

Arguments

`unparsed`	Logical. If `TRUE`, complete API response is returned as an unparsed list. Default is `FALSE`.

Value

When unparsed = `FALSE`: a `tibble` with the following columns:

- `slug`: The slug of the related taxon.
- `image_l_url`: The URL for the "large" rendition.
- `image_o_url`: The URL for the "original" rendition.
- `image_s_url`: The URL for the "small" rendition.
- `image_m_url`: The URL for the "medium" rendition.
- `contributor`: The contributor of the media item.
- `copyright_holder`: The copyright holder.
- `license`: The license of the media item.
- `imaging_instrument`: Comma-separated list of imaging instruments.
- `priority`: The priority of the image.

See Also

https://nordicmicroalgae.org/ for Nordic Microalgae website.

https://nordicmicroalgae.org/api/ for Nordic Microalgae API documentation.

`get_nua_image_labeling_metadata` for retrieving full metadata for image labeling images.

`get_nua_media_links` for retrieving regular media image URLs.

Examples

```
# Retrieve image labeling media links
try(il_links <- get_nua_image_labeling_links(unparsed = FALSE))

# Preview the extracted data
if (exists("il_links")) head(il_links)
```

---

get_nua_image_labeling_metadata

Retrieve image labeling metadata from Nordic Microalgae

Description

This function retrieves detailed metadata for automated imaging images from the Nordic Microalgae API. These are images from automated imaging instruments (e.g., IFCB) used for image labeling purposes. It returns comprehensive metadata including location, instrument, dataset, and taxonomic information.

Usage

```
get_nua_image_labeling_metadata(unparsed = FALSE)
```

Arguments

`unparsed`	Logical. If `TRUE`, complete API response is returned as an unparsed list. Default is `FALSE`.

Value

When unparsed = `FALSE`: a `tibble` with the following columns:

- `slug`: The slug of the media item.
- `taxon_slug`: The slug of the related taxon.
- `scientific_name`: The scientific name of the related taxon.
- `file`: The filename of the media item.
- `type`: The MIME type of the media item.
- `title`: The title of the media item.

- `caption`: The caption of the media item.
- `license`: The license of the media item.
- `location`: The location where the media was collected.
- `contributor`: The contributor of the media item.
- `copyright_holder`: The copyright holder.
- `imaging_instrument`: Comma-separated list of imaging instruments.
- `training_dataset`: DOI or URL of the training dataset.
- `sampling_date`: The date the sample was collected.
- `geographic_area`: The geographic area of collection.
- `latitude_degree`: The latitude in degrees.
- `longitude_degree`: The longitude in degrees.
- `institute`: Comma-separated list of institutes.
- `contributing_organisation`: The contributing organisation.
- `priority`: The priority of the image.
- `created_at`: The creation timestamp.
- `updated_at`: The last update timestamp.

See Also

`https://nordicmicroalgae.org/` for Nordic Microalgae website.

`https://nordicmicroalgae.org/api/` for Nordic Microalgae API documentation.

`get_nua_image_labeling_links` for retrieving image labeling media URLs.

`get_nua_media_metadata` for retrieving regular media metadata.

Examples

```
# Retrieve image labeling metadata
try(il_metadata <- get_nua_image_labeling_metadata(unparsed = FALSE))

# Preview the extracted data
if (exists("il_metadata")) head(il_metadata)
```

get_nua_media_links *Retrieve and extract media URLs from Nordic Microalgae*

Description

This function retrieves media information from the Nordic Microalgae API and extracts slugs and URLs for different renditions (large, original, small, medium) for each media item.

Usage

```
get_nua_media_links(unparsed = FALSE)
```

Arguments

`unparsed`	Logical. If `TRUE`, complete API response is returned as an unparsed list. Default is `FALSE`.

Value

When unparsed = `FALSE`: a `tibble` with the following columns:

- `slug`: The slug of the related taxon.
- `l_url`: The URL for the "large" rendition.
- `o_url`: The URL for the "original" rendition.
- `s_url`: The URL for the "small" rendition.
- `m_url`: The URL for the "medium" rendition.

See Also

`https://nordicmicroalgae.org/` for Nordic Microalgae website.

`https://nordicmicroalgae.org/api/` for Nordic Microalgae API documentation.

Examples

```
# Retrieve media information
try(media_info <- get_nua_media_links(unparsed = FALSE))

# Preview the extracted data
if (exists("media_info")) head(media_info)
```

---

get_nua_media_metadata

Retrieve media metadata from Nordic Microalgae

Description

This function retrieves metadata for media items from the Nordic Microalgae API. It returns detailed attributes such as title, caption, location, sampling date, geographic coordinates, and contributor information.

Usage

```
get_nua_media_metadata(unparsed = FALSE)
```

Arguments

`unparsed`	Logical. If `TRUE`, complete API response is returned as an unparsed list. Default is `FALSE`.

Value

When unparsed = `FALSE`: a `tibble` with the following columns:

- `slug`: The slug of the media item.
- `taxon_slug`: The slug of the related taxon.
- `scientific_name`: The scientific name of the related taxon.
- `file`: The filename of the media item.
- `type`: The MIME type of the media item.
- `title`: The title of the media item.
- `caption`: The caption of the media item.
- `license`: The license of the media item.
- `location`: The location where the media was collected.
- `contributor`: The contributor of the media item.
- `photographer_artist`: The photographer or artist.
- `copyright_holder`: The copyright holder.
- `copyright_stamp`: The copyright stamp.
- `galleries`: Comma-separated list of galleries.
- `technique`: The imaging technique used.
- `contrast_enhancement`: The contrast enhancement method used.
- `preservation`: The preservation method used.
- `stain`: The stain used.
- `sampling_date`: The date the sample was collected.
- `geographic_area`: The geographic area of collection.
- `latitude_degree`: The latitude in degrees.
- `longitude_degree`: The longitude in degrees.
- `institute`: Comma-separated list of institutes.
- `contributing_organisation`: The contributing organisation.
- `created_at`: The creation timestamp.
- `updated_at`: The last update timestamp.

See Also

`https://nordicmicroalgae.org/` for Nordic Microalgae website.

`https://nordicmicroalgae.org/api/` for Nordic Microalgae API documentation.

`get_nua_media_links` for retrieving media image URLs.

Examples

```
# Retrieve media metadata
try(media_metadata <- get_nua_media_metadata(unparsed = FALSE))

# Preview the extracted data
if (exists("media_metadata")) head(media_metadata)
```

---

get_nua_taxa *Retrieve taxa information from Nordic Microalgae*

Description

This function retrieves all taxonomic information for algae taxa from the Nordic Microalgae API. It fetches details including scientific names, authorities, ranks, and image URLs (in different sizes: large, medium, original, and small).

Usage

```
get_nua_taxa(unparsed = FALSE)
```

Arguments

`unparsed`	Logical. If `TRUE`, complete API response is returned as an unparsed list. Default is `FALSE`.

Value

When unparsed = `FALSE`: a `tibble` containing the following columns:

`slug`	A unique identifier for the taxon.
`scientific_name`	The scientific name of the taxon.
`authority`	The authority associated with the scientific name.
`rank`	The taxonomic rank of the taxon.

See Also

`https://nordicmicroalgae.org/` for Nordic Microalgae website.

`https://nordicmicroalgae.org/api/` for Nordic Microalgae API documentation.

Examples

```
# Retrieve and display taxa data
try(taxa_data <- get_nua_taxa(unparsed = FALSE))
if (exists("taxa_data")) head(taxa_data)
```

get_peg_list *Get the latest EG-Phyto/PEG biovolume Excel list*

Description

This function downloads the EG-Phyto (previously PEG) biovolume zip archive from ICES (using `cache_peg_zip()`), unzips it, and reads the first Excel file by default. You can also specify which file in the archive to read.

Usage

```
get_peg_list(
  file = NULL,
  sheet = NULL,
  force = FALSE,
  url = "https://www.ices.dk/data/Documents/ENV/PEG_BVOL.zip",
  clean_cache_days = 30,
  verbose = TRUE
)
```

Arguments

`file`	Character string specifying which file in the zip archive to read. Defaults to the first Excel file in the archive.
`sheet`	Character or numeric; the name or index of the sheet to read from the Excel file. If neither argument specifies the sheet, defaults to the first sheet.
`force`	Logical; if `TRUE`, forces re-download of the zip file even if a cached copy exists.
`url`	Character string with the URL of the PEG zip file. Defaults to the official ICES link.
`clean_cache_days`	Numeric; if not `NULL`, cached PEG zip files older than this number of days will be automatically deleted and replaced by a new download. Defaults to 30. Set to `NULL` to disable automatic cleanup.
`verbose`	A logical indicating whether to print progress messages. Default is TRUE.

Value

A tibble with the contents of the requested Excel file.

See Also

`clean_shark4r_cache()` to manually clear cached files.

Examples

```
# Read the first Excel file from the PEG zip
try(peg_list <- get_peg_list())
if (exists("peg_list")) head(peg_list)
```

get_shark_codes *Get SHARK codelist from SMHI*

Description

This function downloads the SHARK codes Excel file from SMHI (if not already cached) and reads it into R. The file is stored in a persistent cache directory so it does not need to be downloaded again in subsequent sessions.

Usage

```
get_shark_codes(
  url =
   "https://smhi.se/oceanografi/oce_info_data/shark_web/downloads/codelist_SMHI.xlsx",
  sheet = 1,
  skip = 1,
  force = FALSE,
  clean_cache_days = 30
)
```

Arguments

`url`	Character string with the URL to the SHARK codes Excel file. Defaults to the official SMHI codelist.
`sheet`	Sheet to read. Can be either the sheet name or its index (default is `1`).
`skip`	Number of rows to skip before reading data (default is `1`, to skip the header row).
`force`	Logical; if `TRUE`, forces re-download of the Excel file even if a cached copy is available. Default is `FALSE`.
`clean_cache_days`	Numeric; if not `NULL`, cached SHARK code Excel files older than this number of days will be automatically deleted. Defaults to 30. Set to `NULL` to disable automatic cleanup.

Value

A `tibble` containing the contents of the requested sheet.

See Also

`clean_shark4r_cache()` to manually clear cached files.

Examples

```
# Read the first sheet, skipping the first row
try(codes <- get_shark_codes())
if (exists("codes")) head(codes)

# Force re-download of the Excel file
try(codes <- get_shark_codes(force = TRUE))
```

`get_shark_data` *Retrieve tabular data from SHARK*

Description

The `get_shark_data()` function retrieves tabular data from the SHARK database hosted by SMHI. The function sends a POST request to the SHARK API with customizable filters, including year, month, taxon name, water category, and more, and returns the retrieved data as a structured `tibble`. To view available filter options, see `get_shark_options`.

Usage

```
get_shark_data(
  tableView = "sharkweb_overview",
  headerLang = "internal_key",
  save_data = FALSE,
  file_path = NULL,
  delimiters = "point-tab",
  lineEnd = "win",
  encoding = "utf_8",
  dataTypes = c(),
  bounds = c(),
  fromYear = NULL,
  toYear = NULL,
  months = c(),
  parameters = c(),
  checkStatus = "",
  qualityFlags = c(),
  deliverers = c(),
  orderers = c(),
  projects = c(),
  datasets = c(),
  minSamplingDepth = "",
  maxSamplingDepth = "",
  redListedCategory = c(),
  taxonName = c(),
  stationName = c(),
  vattenDistrikt = c(),
  seaBasins = c(),
  counties = c(),
  municipalities = c(),
  waterCategories = c(),
  typOmraden = c(),
  helcomOspar = c(),
  seaAreas = c(),
  hideEmptyColumns = FALSE,
  row_limit = 10^7,
  prod = TRUE,
  utv = FALSE,
  verbose = TRUE
)
```

Arguments

`tableView` Character. Specifies the columns of the table to retrieve. Options include:

- `"sharkweb_overview"`: Overview table
- `"sharkweb_all"`: All available columns
- `"sharkdata_bacterioplankton"`: Bacterioplankton table
- `"sharkdata_chlorophyll"`: Chlorophyll table
- `"sharkdata_epibenthos"`: Epibenthos table
- `"sharkdata_greyseal"`: Greyseal table
- `"sharkdata_harbourporpoise"`: Harbour porpoise table
- `"sharkdata_harbourseal"`: Harbour seal table
- `"sharkdata_jellyfish"`: Jellyfish table
- `"sharkdata_physicalchemical"`: Physical chemical table
- `"sharkdata_physicalchemical_columns"`: Physical chemical table: column view
- `"sharkdata_phytoplankton"`: Phytoplankton table
- `"sharkdata_picoplankton"`: Picoplankton table
- `"sharkdata_planktonbarcoding"`: Plankton barcoding table
- `"sharkdata_primaryproduction"`: Primary production table
- `"sharkdata_ringedseal"`: Ringed seal table
- `"sharkdata_sealpathology"`: Seal pathology table
- `"sharkdata_sedimentation"`: Sedimentation table
- `"sharkdata_zoobenthos"`: Zoobenthos table
- `"sharkdata_zooplankton"`: Zooplankton table
- `"report_sum_year_param"`: Report sum per year and parameter
- `"report_sum_year_param_taxon"`: Report sum per year, parameter and taxon
- `"report_sampling_per_station"`: Report sampling per station
- `"report_obs_taxon"`: Report observed taxa
- `"report_stations"`: Report stations
- `"report_taxon"`: Report taxa

	Default is "sharkweb_overview".
headerLang	Character. Language option for column headers. Possible values: • "sv": Swedish. • "en": English. • "short": Shortened version. • "internal_key": Internal key (default).
save_data	Logical. If TRUE, the downloaded data is written to file_path on disk. If FALSE (default), data is temporarily written to a file and then read into memory as a data.frame, after which the temporary file is deleted.
file_path	Character. The file path where the data should be saved. Required if save_data is TRUE. Ignored if save_data is FALSE.
delimiters	Character. Specifies the delimiter used to separate values in the file, if save_data is TRUE. Options are "point-tab" (tab-separated) or "point-semi" (semicolon-separated). Default is "point-tab".
lineEnd	Character. Defines the type of line endings in the file, if save_data is TRUE. Options are "win" (Windows-style, \r\n) or "unix" (Unix-style, \n). Default is "win".
encoding	Character. Sets the file's text encoding, if save_data is TRUE. Options are "cp1252", "utf_8", "utf_16", or "latin_1". Default is "utf_8".
dataTypes	Character vector. Specifies data types to filter. Possible values include: • "Bacterioplankton" • "Chlorophyll" • "Epibenthos" • "Grey seal" • "Harbour Porpoise" • "Harbour seal" • "Jellyfish" • "Physical and Chemical" • "Phytoplankton" • "Picoplankton" • "PlanktonBarcoding" • "Primary production" • "Profile" • "Ringed seal" • "Seal pathology" • "Sedimentation" • "Zoobenthos" • "Zooplankton"
bounds	A numeric vector of length 4 specifying the geographical search boundaries in decimal degrees, formatted as c(lon_min, lat_min, lon_max, lat_max), e.g., c(11, 58, 12, 59). Default is c() to include all data.
fromYear	Integer (optional). The starting year for data retrieval. If set to NULL (default), the function will use the earliest available year in SHARK.

`toYear`	Integer (optional). The ending year for data retrieval. If set to `NULL` (default), the function will use the latest available year in SHARK.
`months`	Integer vector. The months to retrieve data for, e.g., `c(4, 5, 6)` for April to June.
`parameters`	Character vector. Optional parameters to filter the results by, such as `"Chlorophyll-a"`.
`checkStatus`	Character string. Optional status check to filter results.
`qualityFlags`	Character vector. Specifies the quality flags to filter the data. By default, all data are included, including those with the "B" flag (Bad).
`deliverers`	Character vector. Specifies the data deliverers to filter by.
`orderers`	Character vector. Orderers to filter by specific organizations or individuals.
`projects`	Character vector. Projects to filter data by specific research or monitoring projects.
`datasets`	Character vector. Datasets to filter data by specific datasets.
`minSamplingDepth`	Numeric. Minimum sampling depth (in meters) to filter the data.
`maxSamplingDepth`	Numeric. Maximum sampling depth (in meters) to filter the data.
`redListedCategory`	Character vector. Red-listed taxa for conservation filtering.
`taxonName`	Character vector. Optional vector of taxa names to filter by.
`stationName`	Character vector. Station names to filter data by specific stations.
`vattenDistrikt`	Character vector. Water district names to filter by Swedish water districts.
`seaBasins`	Character vector. Sea basins to filter by.
`counties`	Character vector. Counties to filter by specific administrative regions.
`municipalities`	Character vector. Municipalities to filter by.
`waterCategories`	Character vector. Water categories to filter by.
`typOmraden`	Character vector. Type areas to filter by.
`helcomOspar`	Character vector. HELCOM or OSPAR areas for regional filtering.
`seaAreas`	Character vector. Sea area codes to filter by specific sea areas.
`hideEmptyColumns`	Logical. Whether to hide empty columns. Default is FALSE.
`row_limit`	Numeric. Specifies the maximum number of rows that can be retrieved in a single request. If the requested data exceeds this limit, the function automatically downloads the data in yearly chunks (ignored when `tableView = "report_*"`). The default value is 10 million rows.
`prod`	Logical, whether to download from the production (`TRUE`, default) or test (`FALSE`) SHARK server. Ignored if `utv` is `TRUE`.
`utv`	Logical. Select UTV server when `TRUE`.
`verbose`	Logical. Whether to display progress information. Default is TRUE.

Details

This function sends a POST request to the SHARK API with the specified filters. The API returns a delimited text file (e.g., tab- or semicolon-separated), which is downloaded and read into R as a `tibble`. If the `row_limit` parameter is exceeded, the data is retrieved in yearly chunks and combined into a single table. Adjusting the `row_limit` parameter may be necessary when retrieving large datasets or detailed reports. Note that making very large requests (e.g., retrieving the entire SHARK database) can be extremely time- and memory-intensive.

Value

A `tibble` containing the retrieved SHARK data, parsed from the API's delimited text response. Column types are inferred automatically.

Note

For large queries spanning multiple years or including several data types, retrieval can be time-consuming and memory-intensive. Consider filtering by year, data type, or region for improved performance.

See Also

- `https://shark.smhi.se/en` – SHARK database portal
- `get_shark_options()` – Retrieve available filters
- `get_shark_table_counts()` – Check table row counts before download
- `get_shark_datasets()` – To download datasets as zip-archives

Examples

```
# Retrieve chlorophyll data from 2019 to 2020 for April to June
try(shark_data <- get_shark_data(fromYear = 2019, toYear = 2020,
                                 months = c(4, 5, 6), dataTypes = "Chlorophyll",
                                 verbose = FALSE))
if (exists("shark_data")) print(shark_data)
```

`get_shark_datasets` *Download SHARK dataset zip archives*

Description

Downloads one or more datasets (zip-archives) from the SHARK database (Swedish national marine environmental data archive) and optionally unzips them. The function matches provided dataset names against all available SHARK datasets.

Usage

```
get_shark_datasets(
  dataset_name,
  save_dir = NULL,
  prod = TRUE,
  utv = FALSE,
  unzip_file = FALSE,
  return_df = FALSE,
  encoding = "latin_1",
  guess_encoding = TRUE,
  verbose = TRUE
)
```

Arguments

`dataset_name`	Character vector with one or more dataset names (or partial names). Each entry will be matched against available SHARK dataset identifiers (e.g., `"SHARK_Phytoplankton_2023_SMHI_B` for a specific dataset, or `"SHARK_Phytoplankton"` for all Phytoplankton datasets).
`save_dir`	Directory where zip files (and optionally their extracted contents) should be stored. Defaults to `NULL`. If `NULL` or `""`, a temporary directory is used.
`prod`	Logical, whether to download from the production (`TRUE`, default) or test (`FALSE`) SHARK server. Ignored if `utv` is `TRUE`.
`utv`	Logical. Select UTV server when `TRUE`.
`unzip_file`	Logical, whether to extract downloaded zip archives (`TRUE`) or only save them (`FALSE`, default).
`return_df`	Logical, whether to return a combined data frame with the contents of all downloaded datasets (`TRUE`) instead of a list of file paths (`FALSE`, default).
`encoding`	Character. File encoding of `shark_data.txt`. Options: `"cp1252"`, `"utf_8"`, `"utf_16"`, `"latin_1"`. Default is `"latin_1"`. If `guess_encoding = TRUE`, detected encoding overrides this value. Ignored if `return_df` is `FALSE`.
`guess_encoding`	Logical. If `TRUE` (default), automatically detect file encoding. If `FALSE`, the function uses only the user-specified encoding. Ignored if `return_df` is `FALSE`.
`verbose`	Logical, whether to show download and extraction progress messages. Default is `TRUE`.

Value

If `return_df = FALSE`, a named list of character vectors. Each element corresponds to one matched dataset and contains either the path to the downloaded zip file (if `unzip_file = FALSE`) or the path to the extraction directory (if `unzip_file = TRUE`). If `return_df = TRUE`, a single combined data frame with all dataset contents, including a `source` column indicating the dataset.

See Also

`https://shark.smhi.se/en` for SHARK database.

`get_shark_options()` for listing available datasets.

`get_shark_data()` for downloading tabular data.

Examples

```
# Get a specific dataset
try(get_shark_datasets("SHARK_Phytoplankton_2023_SMHI_BVVF", verbose = FALSE))

# Get all Zooplankton datasets from 2022 and unzip them
try(get_shark_datasets(
  dataset_name = "Zooplankton_2022",
  unzip_file = TRUE,
  verbose = FALSE
))

# Get all Chlorophyll datasets and return as a combined data frame
try(combined_df <- get_shark_datasets(
  dataset_name = "Chlorophyll",
  return_df = TRUE,
  verbose = FALSE
))
if (exists("combined_df")) head(combined_df)
```

---

get_shark_options *Retrieve available search options from SHARK*

Description

The `get_shark_options()` function retrieves available search options from the SHARK database. It sends a GET request to the SHARK API and returns the results as a structured named list.

Usage

```
get_shark_options(prod = TRUE, utv = FALSE, unparsed = FALSE)
```

Arguments

`prod`	Logical value that selects the production server when `TRUE` and the test server when `FALSE`, unless `utv` is `TRUE`.
`utv`	Logical value that selects the UTV server when `TRUE`.
`unparsed`	Logical. If `TRUE`, returns the complete JSON response as a nested list without parsing. Defaults to `FALSE`.

Details

This function sends a GET request to the `/api/options` endpoint of the SHARK API to retrieve available search filters and options that can be used in SHARK data queries.

Value

A named `list` of available search options from the SHARK API. If `unparsed = TRUE`, returns the raw JSON structure as a list.

See Also

`get_shark_data()` for retrieving actual data from the SHARK API.

`https://shark.smhi.se/en` for the SHARK database portal.

Examples

```
# Retrieve available search options (simplified)
try(shark_options <- get_shark_options())
if (exists("shark_options")) names(shark_options)

# Retrieve full unparsed JSON response
try(raw_options <- get_shark_options(unparsed = TRUE))

# View available datatypes
if (exists("shark_options")) print(shark_options$dataTypes)
```

`get_shark_statistics` *Summarize numeric SHARK parameters with ranges and outlier thresholds*

Description

Downloads SHARK data for a given time period, filters to numeric parameters, and calculates descriptive statistics and Tukey outlier thresholds.

Usage

```
get_shark_statistics(
  fromYear = NULL,
  toYear = NULL,
  datatype = NULL,
  group_col = NULL,
  min_obs = 3,
  max_non_numeric_frac = 0.05,
  cache_result = FALSE,
  prod = TRUE,
  utv = FALSE,
  verbose = TRUE
)
```

Arguments

`fromYear`	Start year for download (numeric). Defaults to 5 years before the last complete year.
`toYear`	End year for download (numeric). Defaults to the last complete year.
`datatype`	Optional, one or more datatypes to filter on (e.g. `"Bacterioplankton"`). If `NULL`, all datatypes are included.
`group_col`	Optional column name in the SHARK data to group by (e.g. `"station_name"`). If provided, statistics will be computed separately for each group. Default is `NULL`.
`min_obs`	Minimum number of numeric observations required for a parameter to be included (default: 3).
`max_non_numeric_frac`	Maximum allowed fraction of non-numeric values for a parameter to be kept (default: 0.05).
`cache_result`	Logical, whether to save the result in a persistent cache (`statistics.rds`) for use by other functions. Default is `FALSE`.
`prod`	Logical, whether to download from the production (`TRUE`, default) or test (`FALSE`) SHARK server. Ignored if `utv` is `TRUE`.
`utv`	Logical. Select UTV server when `TRUE`.
`verbose`	Logical, whether to show download progress messages. Default is `TRUE`.

Details

By default, the function uses the *previous five complete years*. For example, if called in 2025 it will use data from 2020–2024.

Value

A tibble with one row per parameter (and optionally per group) and the following columns:

parameter Parameter name (character).

datatype SHARK datatype (character).

min, Q1, median, Q3, max Observed quantiles.

P01, P05, P95, P99 1st, 5th, 95th and 99th percentiles.

IQR Interquartile range.

mean Arithmetic mean of numeric values.

sd Standard deviation of numeric values.

var Variance of numeric values.

cv Coefficient of variation (sd / mean).

mad Median absolute deviation.

mild_lower, mild_upper Lower/upper bounds for mild outliers (1.5 × IQR).

extreme_lower, extreme_upper Lower/upper bounds for extreme outliers (3 × IQR).

n Number of numeric observations used.

fromYear First year included in the SHARK data download (numeric).

toYear Last year included in the SHARK data download (numeric).

<group_col> Optional grouping column if provided.

Examples

```
# Uses previous 5 years automatically, Chlorophyll data only
try(res <- get_shark_statistics(datatype = "Chlorophyll", verbose = FALSE))
if (exists("res")) print(res)

# Group by station name and save result in persistent cache
try(res_station <- get_shark_statistics(datatype = "Chlorophyll",
                                    group_col = "station_name",
                                    cache_result = TRUE,
                                    verbose = FALSE))
if (exists("res_station")) print(res_station)
```

---

get_shark_table_counts

Retrieve SHARK data table row counts

Description

The `get_shark_table_counts()` function retrieves the number of records (row counts) from various SHARK data tables based on specified filters such as year, months, data type, stations, and taxa. To view available filter options, see `get_shark_options`.

Usage

```
get_shark_table_counts(
  tableView = "sharkweb_overview",
  fromYear = 2019,
  toYear = 2020,
  months = c(),
  dataTypes = c(),
  parameters = c(),
  orderers = c(),
  qualityFlags = c(),
  deliverers = c(),
  projects = c(),
  datasets = c(),
  minSamplingDepth = "",
  maxSamplingDepth = "",
  checkStatus = "",
  redListedCategory = c(),
  taxonName = c(),
  stationName = c(),
  vattenDistrikt = c(),
  seaBasins = c(),
  counties = c(),
  municipalities = c(),
  waterCategories = c(),
  typOmraden = c(),
  helcomOspar = c(),
  seaAreas = c(),
  prod = TRUE,
  utv = FALSE
)
```

Arguments

`tableView` Character. Specifies the view of the table to retrieve. Options include:

- `"sharkweb_overview"`: Overview table
- `"sharkweb_all"`: All available columns
- `"sharkdata_bacterioplankton"`: Bacterioplankton table
- `"sharkdata_chlorophyll"`: Chlorophyll table
- `"sharkdata_epibenthos"`: Epibenthos table
- `"sharkdata_greyseal"`: Greyseal table
- `"sharkdata_harbourporpoise"`: Harbour porpoise table
- `"sharkdata_harbourseal`: Harbour seal table
- `"sharkdata_jellyfish"`: Jellyfish table
- `"sharkdata_physicalchemical"`: Physical chemical table
- `"sharkdata_physicalchemical_columns"`: Physical chemical table: column view
- `"sharkdata_phytoplankton"`: Phytoplankton table
- `"sharkdata_picoplankton"`: Picoplankton table
- `"sharkdata_planktonbarcoding"`: Plankton barcoding table
- `"sharkdata_primaryproduction"`: Primary production table
- `"sharkdata_ringedseal"`: Ringed seal table
- `"sharkdata_sealpathology"`: Seal pathology table
- `"sharkdata_sedimentation"`: Sedimentation table
- `"sharkdata_zoobenthos"`: Zoobenthos table
- `"sharkdata_zooplankton"`: Zooplankton table
- `"report_sum_year_param"`: Report sum per year and parameter
- `"report_sum_year_param_taxon"`: Report sum per year, parameter and taxon
- `"report_sampling_per_station"`: Report sampling per station
- `"report_obs_taxon"`: Report observed taxa
- `"report_stations"`: Report stations
- `"report_taxon"`: Report taxa

	Default is "sharkweb_overview".
`fromYear`	Integer. The starting year for the data to retrieve. Default is `2019`.
`toYear`	Integer. The ending year for the data to retrieve. Default is `2020`.
`months`	Integer vector. The months to retrieve data for (e.g., `c(4, 5, 6)` for April to June).
`dataTypes`	Character vector. Specifies data types to filter, such as "Chlorophyll" or "Epibenthos".
`parameters`	Character vector. Optional. Parameters to filter results, such as "Chlorophyll-a".
`orderers`	Character vector. Optional. Orderers to filter data by specific organizations.
`qualityFlags`	Character vector. Optional. Quality flags to filter data.
`deliverers`	Character vector. Optional. Deliverers to filter data by data providers.
`projects`	Character vector. Optional. Projects to filter data by specific research or monitoring projects.
`datasets`	Character vector. Optional. Datasets to filter data by specific dataset names.
`minSamplingDepth`	Numeric. Optional. Minimum depth (in meters) for sampling data.
`maxSamplingDepth`	Numeric. Optional. Maximum depth (in meters) for sampling data.
`checkStatus`	Character string. Optional. Status check to filter results.
`redListedCategory`	Character vector. Optional. Red-listed taxa for conservation filtering.
`taxonName`	Character vector. Optional. Taxa names for filtering specific species or taxa.
`stationName`	Character vector. Optional. Station names to retrieve data from specific stations.
`vattenDistrikt`	Character vector. Optional. Water district names to filter data by Swedish water districts.
`seaBasins`	Character vector. Optional. Sea basin names to filter data by different sea areas.
`counties`	Character vector. Optional. Counties to filter data within specific administrative regions in Sweden.
`municipalities`	Character vector. Optional. Municipalities to filter data within specific local regions.
`waterCategories`	Character vector. Optional. Water categories to filter data by.
`typOmraden`	Character vector. Optional. Type areas to filter data by specific areas.
`helcomOspar`	Character vector. Optional. HELCOM or OSPAR areas for regional filtering.
`seaAreas`	Character vector. Optional. Sea area codes for filtering by specific sea areas.
`prod`	Logical. Select production server when `TRUE` (default). Ignored if `utv` is `TRUE`.
`utv`	Logical. Select UTV server when `TRUE`.

Value

An integer representing the total number of rows in the requested SHARK table after applying the specified filters.

See Also

https://shark.smhi.se/en for SHARK database.

get_shark_options to see filter options

get_shark_data to download SHARK data

Examples

```
# Retrieve chlorophyll data for April to June from 2019 to 2020
try(shark_data_counts <- get_shark_table_counts(fromYear = 2019, toYear = 2020,
                                    months = c(4, 5, 6), dataTypes = c("Chlorophyll")))
if (exists("shark_data_counts")) print(shark_data_counts)
```

---

get_toxin_list *Retrieve marine biotoxin data from IOC-UNESCO Toxins Database*

Description

This function collects data from the IOC-UNESCO Toxins Database and returns information about toxins.

Usage

```
get_toxin_list(return_count = FALSE)
```

Arguments

return_count	Logical. If TRUE, the function returns the count of toxins available in the database. If FALSE (default), it returns detailed toxin data.

Value

If return_count = TRUE, the function returns a numeric value representing the number of toxins in the database. Otherwise, it returns a tibble of toxins with detailed information.

See Also

https://toxins.hais.ioc-unesco.org/ for IOC-UNESCO Toxins Database.

Examples

```
# Retrieve the full list of toxins
try(toxin_list <- get_toxin_list())
if (exists("toxin_list")) head(toxin_list)

# Retrieve only the count of toxins
try(toxin_count <- get_toxin_list(return_count = TRUE))
if (exists("toxin_count")) print(toxin_count)
```

`get_worms_classification` *Retrieve hierarchical classification from WoRMS*

Description

Retrieves the hierarchical taxonomy for one or more AphiaIDs from the World Register of Marine Species (WoRMS) and returns it in a wide format. Optionally, a hierarchy string column can be added that concatenates ranks.

Usage

```
get_worms_classification(
  aphia_ids,
  add_rank_to_hierarchy = FALSE,
  verbose = TRUE
)
```

Arguments

`aphia_ids`	Numeric vector of AphiaIDs to retrieve classification for. Must not be NULL or empty. Duplicates are allowed and will be preserved in the output.
`add_rank_to_hierarchy`	Logical (default FALSE). If TRUE, the hierarchy string prepends rank names (e.g., `[Kingdom] Animalia - [Phylum] Chordata`) to each taxon name in the `worms_hierarchy` column. Only applies if `worms_hierarchy` is present.
`verbose`	Logical (default TRUE). If TRUE, prints progress messages and a progress bar during data retrieval.

Details

The function performs the following steps:

1. Validates input AphiaIDs and removes NA values.
2. Retrieves the hierarchical classification for each AphiaID using `worrms::wm_classification()`.
3. Converts the hierarchy to a wide format with one column per rank.

4. Adds a worms_hierarchy string concatenating all ranks.
5. Preserves input order and duplicates.

Value

A tibble where each row corresponds to an input AphiaID. Typical columns include:

aphia_id The AphiaID of the taxon (matches input).

scientific_name The last scientific name in the hierarchy for this AphiaID.

taxonomic ranks Columns for each rank present in the WoRMS hierarchy (e.g., Kingdom, Phylum, Class, Order, Family, Genus, Species). Missing ranks are NA.

worms_hierarchy A concatenated string of all ranks for this AphiaID. Added for every row if wm_classification() returned hierarchy data. Format depends on add_rank_to_hierarchy.

See Also

wm_classification, https://marinespecies.org/

Examples

```
# Single AphiaID
try(single_taxa <- get_worms_classification(109604, verbose = FALSE))
if (exists("single_taxa")) print(single_taxa)

# Multiple AphiaIDs
try(multiple_taxa <- get_worms_classification(c(109604, 376667), verbose = FALSE))
if (exists("multiple_taxa")) print(multiple_taxa)

# Hierarchy with ranks in the string
try(with_rank <- get_worms_classification(c(109604, 376667),
                                          add_rank_to_hierarchy = TRUE,
                                          verbose = FALSE))

# Print hierarchy columns with ranks
if (exists("with_rank")) print(with_rank$worms_hierarchy[1])

# Compare with result when add_rank_to_hierarchy = FALSE
if (exists("multiple_taxa")) print(multiple_taxa$worms_hierarchy[1])
```

---

get_worms_records *Retrieve WoRMS records*

Description

This function retrieves records from the WoRMS (World Register of Marine Species) database using the worrms R package for a given list of Aphia IDs. If the retrieval fails, it retries a specified number of times before stopping.

Usage

```
get_worms_records(
  aphia_ids,
  max_retries = 3,
  sleep_time = 10,
  verbose = TRUE,
  aphia_id = deprecated()
)
```

Arguments

`aphia_ids`	A vector of Aphia IDs for which records should be retrieved.
`max_retries`	An integer specifying the maximum number of retry attempts for each Aphia ID in case of failure. Default is 3.
`sleep_time`	A numeric value specifying the time (in seconds) to wait between retry attempts. Default is 10 seconds.
`verbose`	A logical indicating whether to print progress messages. Default is TRUE.
`aphia_id`	**[Deprecated]** Use `aphia_ids` instead.

Details

The function attempts to fetch records for each Aphia ID in the provided vector. If a retrieval fails, it retries up to the specified `max_retries`, with a pause of `sleep_time` seconds between attempts. If all retries fail for an Aphia ID, the function stops with an error message.

Value

A `tibble` containing the retrieved WoRMS records for the provided Aphia IDs. Each row corresponds to one Aphia ID.

See Also

https://marinespecies.org/ for WoRMS website.

https://CRAN.R-project.org/package=worrms

Examples

```
# Example usage with a vector of Aphia IDs
aphia_ids <- c(12345, 67890, 112233)
try(worms_records <- get_worms_records(aphia_ids, verbose = FALSE))

if (exists("worms_records")) print(worms_records)
```

get_worms_taxonomy_tree

Retrieve hierarchical taxonomy data from WoRMS

Description

Retrieves the hierarchical taxonomy for one or more AphiaIDs from the World Register of Marine Species (WoRMS). Optionally, the function can include all descendants of taxa at a specified rank and/or synonyms for all retrieved taxa.

Usage

```
get_worms_taxonomy_tree(
  aphia_ids,
  add_descendants = FALSE,
  add_descendants_rank = "Species",
  add_synonyms = FALSE,
  add_hierarchy = FALSE,
  add_rank_to_hierarchy = FALSE,
  verbose = TRUE
)
```

Arguments

`aphia_ids`	Numeric vector of AphiaIDs to retrieve taxonomy for. Must not be missing or all NA.
`add_descendants`	Logical (default FALSE). If TRUE, retrieves all child taxa for each taxon at the rank specified by `add_descendants_rank`.
`add_descendants_rank`	Character (default `"Species"`). The taxonomic rank of descendants to retrieve. For example, if set to `"Species"`, the function will collect all species belonging to each genus present in the initial dataset.
`add_synonyms`	Logical (default FALSE). If TRUE, retrieves synonym records for all retrieved taxa and appends them to the dataset.
`add_hierarchy`	Logical (default FALSE). If TRUE, adds a `hierarchy` column that contains the concatenated hierarchy of each taxon (e.g. Kingdom - Phylum - Class).
`add_rank_to_hierarchy`	Logical (default FALSE). If TRUE, the hierarchy string prepends rank names (e.g., `[Kingdom] Animalia - [Phylum] Chordata`) to each taxon name in the `hierarchy` column. Only used if `add_hierarchy` = `TRUE`.
`verbose`	Logical (default TRUE). If TRUE, prints progress messages and progress bars during data retrieval.

Details

The function performs the following steps:

1. Validates input AphiaIDs and removes NA values.
2. Retrieves the hierarchical classification for each AphiaID using `worrms::wm_classification()`.
3. Optionally retrieves all descendants at the rank specified by `add_descendants_rank` if `add_descendants = TRUE`.
4. Optionally retrieves synonyms for all retrieved taxa if `add_synonyms = TRUE`.
5. Optionally adds a `hierarchy` column if `add_hierarchy = TRUE`.
6. Returns a combined, distinct dataset of all records.

Value

A `tibble` containing detailed WoRMS records for all requested AphiaIDs, including optional descendants and synonyms. Typical columns include:

AphiaID The AphiaID of the taxon.

parentNameUsageID The AphiaID of the parent taxon.

scientificname Scientific name of the taxon.

rank Taxonomic rank (e.g., Kingdom, Phylum, Genus, Species).

status Taxonomic status (e.g., accepted, unaccepted).

valid_AphiaID AphiaID of the accepted taxon, if the record is a synonym.

species Added only if a `Species` rank exists in the retrieved data and if `add_hierarchy = TRUE`; otherwise not present.

parentName Added only if a `parentName` rank exists in the retrieved data and if `add_hierarchy = TRUE`; otherwise not present.

hierarchy Added only if `add_hierarchy = TRUE` and hierarchy data are available. Contains a concatenated string of the taxonomic path.

... Additional columns returned by WoRMS, including authorship and source information.

See Also

`add_worms_taxonomy`, `construct_dyntaxa_table`

`wm_classification`, `wm_children`, `wm_synonyms`

`https://marinespecies.org/` for the WoRMS website.

Examples

```
# Retrieve hierarchy for a single AphiaID
try(get_worms_taxonomy_tree(aphia_ids = 109604, verbose = FALSE))

# Retrieve hierarchy including species-level descendants
try(get_worms_taxonomy_tree(
  aphia_ids = c(109604, 376667),
  add_descendants = TRUE,
  verbose = FALSE
))

# Retrieve hierarchy including hierarchy column
try(get_worms_taxonomy_tree(
  aphia_ids = c(109604, 376667),
  add_hierarchy = TRUE,
  verbose = FALSE
))
```

is_in_dyntaxa *Check if taxon names exist in Dyntaxa*

Description

Checks whether the supplied scientific names exist in the Swedish taxonomic database Dyntaxa. Optionally, returns a data frame with taxon names, taxon IDs, and match status.

Usage

```
is_in_dyntaxa(
  taxon_names,
  subscription_key = Sys.getenv("DYNTAXA_KEY"),
  use_dwca = FALSE,
  return_df = FALSE,
  verbose = FALSE
)
```

Arguments

`taxon_names` Character vector of taxon names to check.

`subscription_key`
A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`.

You can provide the key in three ways:

- **Directly as a parameter**: `is_in_dyntaxa("Skeletonema marinoi", subscription_key = "your_key_here")`.
- **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`use_dwca` Logical; if TRUE, uses the DwCA version of Dyntaxa instead of querying the API.

`return_df`	Logical; if TRUE, returns a data frame with columns `taxon_name`, `taxon_id`, and `match`. Default is FALSE (returns a logical vector).
`verbose`	Logical; if TRUE, prints messages about unmatched taxa.

Details

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Value

If `return_df` = `FALSE` (default), a logical vector indicating whether each input name was found in Dyntaxa. Returned invisibly if `verbose` = `TRUE`. If `return_df` = `TRUE`, a data frame with columns:

- `taxon_name`: original input names
- `taxon_id`: corresponding Dyntaxa taxon IDs (NA if not found)
- `match`: logical indicating presence in Dyntaxa

Examples

```
## Not run:
# Using an environment variable (recommended for convenience)
Sys.setenv(DYNTAXA_KEY = "your_key_here")
is_in_dyntaxa(c("Skeletonema marinoi", "Nonexistent species"))

# Return a data frame instead of logical vector
is_in_dyntaxa(c("Skeletonema marinoi", "Nonexistent species"), return_df = TRUE)

# Or pass the key directly
is_in_dyntaxa("Skeletonema marinoi", subscription_key = "your_key_here")

# Suppress messages
is_in_dyntaxa("Skeletonema marinoi", verbose = FALSE)

## End(Not run)
```

`load_shark4r_fields` *Load SHARK4R fields from GitHub*

Description

This function downloads and sources the SHARK4R required and recommended field definitions directly from the SHARK4R-statistics GitHub repository.

Usage

```
load_shark4r_fields(verbose = TRUE)
```

Arguments

verbose	Logical; if TRUE (default), prints progress messages during download and loading.

Details

The definitions are stored in an R script (`fields.R`) located in the `fields/` folder of the repository. The function sources this file directly from GitHub into the current R session.

The sourced script defines two main objects:

- `required_fields` — vector or data frame of required SHARK fields.
- `recommended_fields` — vector or data frame of recommended SHARK fields.

The output of this function can be directly supplied to the `check_fields` function through its `field_definitions` argument for validating SHARK4R data consistency.

If sourcing fails (e.g., due to a network issue or repository changes), the function throws an error with a descriptive message.

Value

Invisibly returns a list with two elements:

required_fields Object containing required SHARK fields.

recommended_fields Object containing recommended SHARK fields.

See Also

`check_fields` for validating datasets using the loaded field definitions (as `field_definitions`). `load_shark4r_stats` for loading precomputed SHARK4R statistics,

Examples

```
# Load SHARK4R field definitions from GitHub
try(fields <- load_shark4r_fields(verbose = FALSE))

# Access required or recommended fields for the first entry
if (exists("fields")) fields[[1]]$required
if (exists("fields")) fields[[1]]$recommended

## Not run:
# Use the loaded definitions in check_fields()
check_fields(my_data, field_definitions = fields)

## End(Not run)
```

`load_shark4r_stats` *Load SHARK4R statistics from GitHub*

Description

This function downloads and loads precomputed SHARK4R statistical data (e.g., threshold or summary statistics) directly from the SHARK4R-statistics GitHub repository. The data are stored as `.rds` files and read into R as objects.

Usage

```
load_shark4r_stats(file_name = "sea_basin.rds", verbose = TRUE)
```

Arguments

`file_name`	Character string specifying the name of the `.rds` file to download. Defaults to `"sea_basin.rds"`.
`verbose`	Logical; if `TRUE` (default), prints progress messages during download and loading.

Details

The function retrieves the file from the GitHub repository's `data/` folder. It temporarily downloads the file to the local system and then reads it into R using `readRDS()`.

If the download fails (e.g., due to a network issue or invalid filename), the function throws an error with a descriptive message.

Value

An R object (typically a `tibble` or `data.frame`) read from the specified `.rds` file.

See Also

`check_outliers` for detecting threshold exceedances using the loaded statistics, `get_shark_statistics` for generating and caching statistical summaries used in SHARK4R. `scatterplot` for generating interactive plots with threshold values.

Examples

```
# Load the default SHARK4R statistics file
try(stats <- load_shark4r_stats(verbose = FALSE))
if (exists("stats")) print(stats)

# Load a specific file
try(thresholds <- load_shark4r_stats("scientific_name.rds", verbose = FALSE))
if (exists("thresholds")) print(thresholds)
```

`lookup_xy` *Lookup spatial information for geographic points*

Description

Retrieves shore distance, environmental grids, and area values for given coordinates. Coordinates may be supplied either through a data frame or as separate numeric vectors.

Usage

```
lookup_xy(
  data = NULL,
  lon = NULL,
  lat = NULL,
  shoredistance = TRUE,
  grids = TRUE,
  areas = FALSE,
  as_data_frame = TRUE
)
```

Arguments

`data`	Optional data frame containing coordinate columns. The expected names are `sample_longitude_dd` and `sample_latitude_dd`. These must be numeric and fall within valid geographic ranges.
`lon`	Optional numeric vector of longitudes. Must be supplied together with `lat` when used. Ignored when a data frame is provided unless both `lon` and `lat` are set.
`lat`	Optional numeric vector of latitudes. Must be supplied together with `lon` when used.
`shoredistance`	Logical; if `TRUE`, distance to the nearest shore is included.
`grids`	Logical; if `TRUE`, environmental grid values are included.
`areas`	Logical or numeric. When logical, `TRUE` requests area values at zero radius, and `FALSE` disables area retrieval. A positive integer specifies the search radius in meters for area values.
`as_data_frame`	Logical; if `TRUE`, the result is returned as a data frame. When `FALSE`, the result is returned as a list.

Details

- When both vector inputs and a data frame are provided, the vector inputs take precedence.
- Coordinates are validated and cleaned before lookup, and only unique values are queried.
- Queries are processed in batches to avoid overloading the remote service.
- Area retrieval accepts either a logical flag or a radius. A radius of zero corresponds to requesting a single area value.

- Final results are reordered to match the original input positions.
- The function has been modified from the `obistools` package (Provoost and Bosch, 2024).

Value

A data frame or list, depending on `as_data_frame`. Invalid coordinates produce `NA` entries (data frame) or `NULL` elements (list). Duplicate input coordinates return repeated results.

References

Provoost P, Bosch S (2024). "obistools: Tools for data enhancement and quality control" Ocean Biodiversity Information System. Intergovernmental Oceanographic Commission of UNESCO. R package version 0.1.0, `https://iobis.github.io/obistools/`.

See Also

`check_onland`, `check_depth`, `https://iobis.github.io/xylookup/` – OBIS xylookup web service

Examples

```
# Using a data frame
df <- data.frame(sample_longitude_dd = c(10.9, 18.3),
                 sample_latitude_dd = c(58.1, 58.3))
try(lookup_xy(df))

# Area search within a radius
try(lookup_xy(df, areas = 500))

# Using separate coordinate vectors
try(lookup_xy(lon = c(10.9, 18.3), lat = c(58.1, 58.3)))
```

`match_algaebase_genus` *Search AlgaeBase for information about a genus of algae*

Description

This function searches the AlgaeBase API for genus information and returns detailed taxonomic data, including higher taxonomy, taxonomic status, scientific names, and other related metadata.

Usage

```
match_algaebase_genus(
  genus,
  subscription_key = Sys.getenv("ALGAEBASE_KEY"),
  higher = TRUE,
  unparsed = FALSE,
  newest_only = TRUE,
  exact_matches_only = TRUE,
  apikey = deprecated()
)
```

Arguments

`genus`	The genus name to search for (character string). This parameter is required.
`subscription_key`	A character string containing the API key for accessing the AlgaeBase API. By default, the key is read from the environment variable `ALGAEBASE_KEY`. You can provide the key in three ways: • **Directly as a parameter**: `match_algaebase_genus("Skeletonema", subscription_key = "your_key_here")`. • **Temporarily for the session**: `Sys.setenv(ALGAEBASE_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function. • **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `ALGAEBASE_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.
`higher`	A boolean flag indicating whether to include higher taxonomy in the output (default is TRUE).
`unparsed`	A boolean flag indicating whether to return the raw JSON output from the API (default is FALSE).
`newest_only`	A boolean flag to return only the most recent entry (default is TRUE).
`exact_matches_only`	A boolean flag to limit results to exact matches (default is TRUE).
`apikey`	**[Deprecated]** Use `subscription_key` instead.

Details

A valid API key is requested from the AlgaeBase team.

Value

A `tibble` with the following columns:

- `id` — AlgaeBase identifier.
- `accepted_name` — Accepted scientific name (if different from the input).
- `input_name` — The genus name supplied by the user.
- `input_match` — Indicator of exact match (`1` = exact, `0` = not exact).
- `currently_accepted` — Indicator if the taxon is currently accepted (`1` = TRUE, `0` = FALSE).
- `genus_only` — Indicator if the search was for a genus only (`1` = genus, `0` = genus + species).
- `kingdom`, `phylum`, `class`, `order`, `family` — Higher taxonomy (returned if `higher` = `TRUE`).
- `taxonomic_status` — Status of the taxon (e.g., currently accepted, synonym, unverified).

- `taxon_rank` — Taxonomic rank of the accepted name (e.g., genus, species).
- `mod_date` — Date when the entry was last modified.
- `long_name` — Full scientific name including author and date (if available).
- `authorship` — Author information (if available).

See Also

https://www.algaebase.org/ for AlgaeBase website.

Examples

```
## Not run:
  match_algaebase_genus("Anabaena", subscription_key = "your_api_key")

## End(Not run)
```

`match_algaebase_species` *Search AlgaeBase for information about a species of algae*

Description

This function searches the AlgaeBase API for species based on genus and species names. It allows for flexible search parameters such as filtering by exact matches, returning the most recent results, and including higher taxonomy details.

Usage

```
match_algaebase_species(
  genus,
  species,
  subscription_key = Sys.getenv("ALGAEBASE_KEY"),
  higher = TRUE,
  unparsed = FALSE,
  newest_only = TRUE,
  exact_matches_only = TRUE,
  apikey = deprecated()
)
```

Arguments

`genus`	A character string specifying the genus name.
`species`	A character string specifying the species or specific epithet.

`subscription_key`
: A character string containing the API key for accessing the AlgaeBase API. By default, the key is read from the environment variable `ALGAEBASE_KEY`.

  You can provide the key in three ways:

  - **Directly as a parameter**: `match_algaebase_species("Skeletonema", "marinoi", subscription_key = "your_key_here")`.
  - **Temporarily for the session**: `Sys.setenv(ALGAEBASE_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
  - **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `ALGAEBASE_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`higher`
: A logical value indicating whether to include higher taxonomy details (default is `TRUE`).

`unparsed`
: A logical value indicating whether to print the full JSON response from the API (default is `FALSE`).

`newest_only`
: A logical value indicating whether to return only the most recent entries (default is `TRUE`).

`exact_matches_only`
: A logical value indicating whether to return only exact matches (default is `TRUE`).

`apikey`
: **[Deprecated]** Use `subscription_key` instead.

Details

A valid API key is requested from the AlgaeBase team.

This function queries the AlgaeBase API for species based on the genus and species names, and filters the results based on various parameters. The function handles different taxonomic ranks and formats the output for easy use. It can merge higher taxonomy data if requested.

Value

A `tibble` with details about the species, including:

- `taxonomic_status` — The current status of the taxon (e.g., accepted, synonym, unverified).
- `taxon_rank` — The rank of the taxon (e.g., species, genus).
- `accepted_name` — The currently accepted scientific name, if applicable.
- `authorship` — Author information for the scientific name (if available).
- `mod_date` — Date when the taxonomic record was last modified.
- `...` — Other relevant information returned by the data source.

See Also

https://www.algaebase.org/ for AlgaeBase website.

Examples

```
## Not run:
# Search for a species with exact matches only, return the most recent results
result <- match_algaebase_species(
  genus = "Skeletonema", species = "marinoi", subscription_key = "your_api_key"
)

# Print result
print(result)

## End(Not run)
```

`match_algaebase_taxa` *Search AlgaeBase for taxonomic information*

Description

This function queries the AlgaeBase API to retrieve taxonomic information for a list of algae names based on genus and (optionally) species. It supports exact matching, genus-only searches, and retrieval of higher taxonomic ranks.

Usage

```
match_algaebase_taxa(
  genera,
  species,
  subscription_key = Sys.getenv("ALGAEBASE_KEY"),
  genus_only = FALSE,
  higher = TRUE,
  unparsed = FALSE,
  exact_matches_only = TRUE,
  sleep_time = 1,
  newest_only = TRUE,
  verbose = TRUE,
  apikey = deprecated(),
  genus = deprecated()
)
```

Arguments

`genera`	A character vector of genus names.
`species`	A character vector of species names corresponding to the `genera` vector. Must be the same length as `genera`.
`subscription_key`	A character string containing the API key for accessing the AlgaeBase API. By default, the key is read from the environment variable `ALGAEBASE_KEY`. You can provide the key in three ways:

- **Directly as a parameter**: `match_algaebase_taxa("Skeletonema", "marinoi", subscription_key = "your_key_here")`
- **Temporarily for the session**: `Sys.setenv(ALGAEBASE_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `ALGAEBASE_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`genus_only`	Logical. If `TRUE`, searches are based solely on the genus name, ignoring species. Defaults to `FALSE`.
`higher`	Logical. If `TRUE`, includes higher taxonomy (e.g., kingdom, phylum) in the output. Defaults to `TRUE`.
`unparsed`	Logical. If `TRUE`, returns raw JSON output instead of a `tibble`. Defaults to `FALSE`.
`exact_matches_only`	Logical. If `TRUE`, restricts results to exact matches. Defaults to `TRUE`.
`sleep_time`	Numeric. The delay (in seconds) between consecutive AlgaeBase API queries. Defaults to `1`. A delay is recommended to avoid overwhelming the API for large queries.
`newest_only`	A logical value indicating whether to return only the most recent entries (default is `TRUE`).
`verbose`	Logical. If `TRUE`, displays a progress bar to indicate query status. Defaults to `TRUE`.
`apikey`	**[Deprecated]** Use `subscription_key` instead.
`genus`	**[Deprecated]** Use `genera` instead.

Details

A valid API key is requested from the AlgaeBase team.

Scientific names can be parsed using the `parse_scientific_names()` function before being processed by `match_algaebase_taxa()`.

Duplicate genus-species combinations are handled efficiently by querying each unique combination only once. Genus-only searches are performed when `genus_only = TRUE` or when the species name is missing or invalid. Errors during API queries are gracefully handled by returning rows with `NA` values for missing or unavailable data.

The function allows for integration with data analysis workflows that require resolving or verifying taxonomic names against AlgaeBase.

Value

A `tibble` containing taxonomic information for each input genus–species combination. The following columns may be included:

- `id` — AlgaeBase ID (if available).
- `kingdom`, `phylum`, `class`, `order`, `family` — Higher taxonomy (returned if `higher = TRUE`).
- `genus`, `species`, `infrasp` — Genus, species, and infraspecies names (if applicable).

- `taxonomic_status` — Status of the name (e.g., accepted, synonym, unverified).
- `currently_accepted` — Logical indicator whether the name is currently accepted (`TRUE`/`FALSE`).
- `accepted_name` — Currently accepted name if different from the input name.
- `input_name` — The name supplied by the user.
- `input_match` — Indicator of exact match (`1` = exact, `0` = not exact).
- `taxon_rank` — Taxonomic rank of the accepted name (e.g., genus, species).
- `mod_date` — Date when the entry was last modified in AlgaeBase.
- `long_name` — Full species name with authorship and date.
- `authorship` — Author(s) associated with the species name.

See Also

`https://www.algaebase.org/` for AlgaeBase website.

`parse_scientific_names` for parsing taxonomic names before passing them to the function.

Examples

```
## Not run:
# Example with genus and species vectors
genus_vec <- c("Thalassiosira", "Skeletonema", "Tripos")
species_vec <- c("pseudonana", "costatum", "furca")

algaebase_results <- match_algaebase_taxa(
  genera = genus_vec,
  species = species_vec,
  subscription_key = "your_api_key",
  exact_matches_only = TRUE,
  verbose = TRUE
)
head(algaebase_results)

## End(Not run)
```

`match_dyntaxa_taxa` *Match Dyntaxa taxon names*

Description

This function matches a list of taxon names against the SLU Artdatabanken API (Dyntaxa) and retrieves the best matches along with their taxon IDs.

Usage

```
match_dyntaxa_taxa(
  taxon_names,
  subscription_key = Sys.getenv("DYNTAXA_KEY"),
  multiple_options = FALSE,
  searchFields = "Both",
  isRecommended = "NotSet",
  isOkForObservationSystems = "NotSet",
  culture = "sv_SE",
  page = 1,
  pageSize = 100,
  verbose = TRUE
)
```

Arguments

`taxon_names` A vector of taxon names to match.

`subscription_key`
A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`.

You can provide the key in three ways:

- **Directly as a parameter**: `match_dyntaxa_taxa("Skeletonema marinoi", subscription_key = "your_key_here")`.
- **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`multiple_options`
Logical. If TRUE, the function will return multiple matching names. Default is FALSE, selecting the first match.

`searchFields` A character string indicating the search fields. Defaults to 'Both'.

`isRecommended` A character string indicating whether the taxon is recommended. Defaults to 'NotSet'.

`isOkForObservationSystems`
A character string indicating whether the taxon is suitable for observation systems. Defaults to 'NotSet'.

`culture` A character string indicating the culture. Defaults to 'sv_SE'.

`page` An integer specifying the page number for pagination. Defaults to 1.

`pageSize` An integer specifying the page size for pagination. Defaults to 100.

`verbose` Logical. Print progress bar. Default is TRUE.

Details

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Note: Please review the API conditions and register for access before using the API. Data collected through the API is stored at SLU Artdatabanken. Please also note that the authors of `SHARK4R` are not affiliated with SLU Artdatabanken.

Value

A `tibble` containing the search pattern, taxon ID, and best match for each taxon name.

See Also

SLU Artdatabanken API Documentation

Examples

```
## Not run:
# Match taxon names against SLU Artdatabanken API
matched_taxa <- match_dyntaxa_taxa(c("Homo sapiens", "Canis lupus"), "your_subscription_key")
print(matched_taxa)

## End(Not run)
```

`match_station` *Match station names against SMHI station list*

Description

Matches reported station names in your dataset against a curated station list ("`station.txt`"), which is synced with "Stationsregistret": `https://stationsregister.miljodatasamverkan.se/`.

Usage

```
match_station(names, station_file = NULL, try_synonyms = TRUE, verbose = TRUE)
```

Arguments

`names`	Character vector of station names to match.
`station_file`	Optional path to a custom station file (tab-delimited). If `NULL` (default), the function will first attempt to use the `NODC_CONFIG` environment variable, and if that fails, will use the bundled "`station.zip`" from the `SHARK4R` package.
`try_synonyms`	Logical; if `TRUE` (default), unmatched names are also compared against the `SYNONYM_NAMES` column in the database.
`verbose`	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Details

This function is useful for validating station names and identifying any unmatched or misspelled entries.

If `try_synonyms = TRUE`, unmatched station names are also compared against the `SYNONYM_NAMES` column in the station database, splitting multiple synonyms separated by `<or>`.

The function first checks if a station file path is provided via the `station_file` argument. If not, it looks for the `NODC_CONFIG` environment variable. This variable can point to a folder where the NODC (Swedish National Oceanographic Data Center) configuration and station file are stored, typically including:

- `<NODC_CONFIG>/config/station.txt`

If `NODC_CONFIG` is set and the folder exists, the function will use `station.txt` from that location. Otherwise, it falls back to the bundled `station.zip` included in the `SHARK4R` package.

Value

A data frame with two columns:

reported_station_name The input station names.

match_type Logical; `TRUE` if the station was found in the SMHI station list (including synonyms if enabled), otherwise `FALSE`.

Examples

```
# Example stations
stations <- c("ANHOLT E", "BY5 BORNHOLMSDJ", "STX999")

# Check if stations names are in stations.txt (including synonyms)
match_station(stations, try_synonyms = TRUE, verbose = FALSE)
```

`match_worms_taxa` *Retrieve WoRMS records by taxonomic names with retry logic*

Description

This function retrieves records from the WoRMS database using the `worrms` R package for a vector of taxonomic names. It includes retry logic to handle temporary failures and ensures all names are processed. The function can query all names at once using a bulk API call or iterate over names individually.

Usage

```
match_worms_taxa(
  taxa_names,
  fuzzy = TRUE,
  best_match_only = TRUE,
  max_retries = 3,
  sleep_time = 10,
  marine_only = TRUE,
  bulk = FALSE,
  chunk_size = 500,
  verbose = TRUE
)
```

Arguments

`taxa_names`	A character vector of taxonomic names for which to retrieve records.
`fuzzy`	A logical value indicating whether to perform a fuzzy search. Default is TRUE. **Note:** Fuzzy search is only applied in iterative mode (`bulk = FALSE`) and is ignored in bulk mode.
`best_match_only`	A logical value indicating whether to automatically select the first match and return a single match. Default is TRUE.
`max_retries`	Integer specifying the maximum number of retries for the request in case of failure. Default is 3.
`sleep_time`	Numeric specifying the number of seconds to wait before retrying a failed request. Default is 10.
`marine_only`	Logical indicating whether to restrict results to marine taxa only. Default is TRUE.
`bulk`	Logical indicating whether to perform a bulk API call for all unique names at once. Default is FALSE.
`chunk_size`	Integer specifying the maximum number of taxa per bulk API request. Default is 500. Only used when `bulk = TRUE`. WoRMS API may reject very large requests, so chunking prevents overload.
`verbose`	Logical indicating whether to print progress messages. Default is TRUE.

Details

- If `bulk = TRUE`, all unique names are sent to the API in a single request. Fuzzy matching is ignored.
- If `bulk = FALSE`, the function iterates over names individually, optionally using fuzzy matching.
- The function retries failed requests up to `max_retries` times, pausing for `sleep_time` seconds between attempts.
- Names for which no records are found will have `status = "no content"` and `AphiaID = NA`.

- Names are cleaned before being passed to the API call by converting them to UTF-8, replacing problematic symbols with spaces, removing trailing periods, collapsing extra spaces and by trimming whitespace.

Value

A `tibble` containing the retrieved WoRMS records. Each row corresponds to a record for a taxonomic name. Repeated taxa in the input are preserved in the output.

See Also

https://marinespecies.org/ for WoRMS website.

https://CRAN.R-project.org/package=worrms

Examples

```
# Retrieve WoRMS records iteratively for two taxonomic names
try(records <- match_worms_taxa(c("Amphidinium", "Karenia"),
                            max_retries = 3,
                            sleep_time = 5,
                            marine_only = TRUE,
                            verbose = FALSE))
if (exists("records")) print(records)

# Retrieve WoRMS records in bulk mode (faster for many names)
try(records_bulk <- match_worms_taxa(c("Amphidinium", "Karenia", "Navicula"),
                                 bulk = TRUE,
                                 marine_only = TRUE,
                                 verbose = FALSE))
```

---

parse_scientific_names *Parse scientific names into genus and species components.*

Description

This function processes a character vector of scientific names, splitting them into genus and species components. It handles binomial names (e.g., "Homo sapiens"), removes undesired descriptors (e.g., 'Cfr.', 'cf.', 'sp.', 'spp.'), and manages cases involving varieties, subspecies, or invalid species names. Special characters and whitespace are handled appropriately.

Usage

```
parse_scientific_names(
  scientific_names,
  remove_undesired_descriptors = TRUE,
  remove_subspecies = TRUE,
  remove_invalid_species = TRUE,
  encoding = "UTF-8",
  scientific_name = deprecated()
)
```

Arguments

`scientific_names`	A character vector containing scientific names, which may include binomials, additional descriptors, or varieties.
`remove_undesired_descriptors`	Logical, if TRUE, undesired descriptors (e.g., 'Cfr.', 'cf.', 'colony', 'cells', etc.) are removed. Default is TRUE.
`remove_subspecies`	Logical, if TRUE, subspecies/variety descriptors (e.g., 'var.', 'subsp.', 'f.', etc.) are removed. Default is TRUE.
`remove_invalid_species`	Logical, if TRUE, invalid species names (e.g., 'sp.', 'spp.') are removed. Default is TRUE.
`encoding`	A string specifying the encoding to be used for the input names (e.g., 'UTF-8'). Default is 'UTF-8'.
`scientific_name`	**[Deprecated]** Use `scientific_names` instead.

Value

A `tibble` with two columns:

- `genus` — Genus names.
- `species` — Species names (empty if unavailable or invalid). Invalid descriptors such as `"sp."`, `"spp."`, and numeric entries are excluded from this column.

See Also

`https://www.algaebase.org/` for AlgaeBase website.

Examples

```
# Example with a vector of scientific names
scientific_names <- c("Skeletonema marinoi", "Cf. Azadinium perforatum", "Gymnodinium sp.",
                      "Melosira varians", "Aulacoseira islandica var. subarctica")

# Parse names
result <- parse_scientific_names(scientific_names)

# Check the resulting data
print(result)
```

plot_map_leaflet *Create an interactive Leaflet map of sampling stations*

Description

Generates an interactive map using the `leaflet` package, plotting sampling stations from a data frame. The function automatically detects column names for station, longitude, and latitude, supporting both standard and delivery-style datasets.

Usage

```
plot_map_leaflet(data, provider = "CartoDB.Positron")
```

Arguments

`data`	A data frame containing station coordinates and names. The function accepts either: • Standard format: `station_name`, `sample_longitude_dd`, `sample_latitude_dd` • Delivery format: `STATN`, `LONGI`, `LATIT`
`provider`	Character. The tile provider to use for the map background. See available providers at https://leaflet-extras.github.io/leaflet-providers/preview/. Defaults to `"CartoDB.Positron"`.

Value

An HTML widget object (`leaflet` map) that can be printed or displayed in R Markdown or Shiny applications.

Examples

```
# Example data
df <- data.frame(
  station_name = c("Station A", "Station B"),
  sample_longitude_dd = c(10.0, 10.5),
  sample_latitude_dd = c(59.0, 59.5)
)

# Plot points on map
map <- plot_map_leaflet(df)

# Example data in SHARK delivery format
df_deliv <- data.frame(
  STATN = c("Station A", "Station B"),
  LONGI = c(10.0, 10.5),
  LATIT = c(59.0, 59.5)
)

# Plot points on map
```

```
map_deliv <- plot_map_leaflet(df_deliv)
```

---

positions_are_near_land

Determine if positions are near land

Description

This function is a **wrapper/re-export** of `iRfcb::ifcb_is_near_land()`. The `iRfcb` package is only required if you want to actually call this function.

Usage

```
positions_are_near_land(
  latitudes,
  longitudes,
  distance = 500,
  shape = NULL,
  source = "obis",
  crs = 4326,
  remove_small_islands = TRUE,
  small_island_threshold = 2e+06,
  plot = FALSE,
  verbose = TRUE
)
```

Arguments

`latitudes`	Numeric vector of latitudes for positions.
`longitudes`	Numeric vector of longitudes for positions. Must be the same length as `latitudes`.
`distance`	Buffer distance (in meters) from the coastline to consider "near land." Default is 500 meters.
`shape`	Optional path to a shapefile (`.shp`) containing coastline data. If provided, this file will be used instead of the default OBIS land vectors. A high-resolution shapefile can improve the accuracy of buffer distance calculations. You can retrieve a more detailed European coastline by setting the `source` argument to `"eea"`. Downloaded shape files are cached across R sessions in a user-specific cache directory.
`source`	Character string indicating which default coastline source to use when `shape = NULL`. Options are `"obis"` (Ocean Biodiversity Information System, default), `"ne"` (Natural Earth 1:10 vectors) and `"eea"` (European Environment Agency). Ignored if `shape` is provided.
`crs`	Coordinate reference system (CRS) to use for input and output. Default is EPSG code 4326 (WGS84).

`remove_small_islands`	Logical indicating whether to remove small islands from the coastline. Useful in archipelagos. Default is `TRUE`.
`small_island_threshold`	Area threshold in square meters below which islands will be considered small and removed, if remove_small_islands is set to `TRUE`. Default is 2 square km.
`plot`	A boolean indicating whether to plot the points, land polygon and buffer. Default is `FALSE`.
`verbose`	A logical indicating whether to print progress messages. Default is TRUE.

Details

Determines whether given positions are near land based on a land polygon shape file.

This function calculates a buffered area around the coastline using a polygon shapefile and determines if each input position intersects with this buffer or the landmass itself. By default, it uses the OBIS land vector dataset.

The EEA shapefile is downloaded from `https://www.eea.europa.eu/data-and-maps/data/eea-coastline-for-analysis-2/gis-data/eea-coastline-polygon` when `source = "eea"`.

Value

If `plot = FALSE` (default), a logical vector is returned indicating whether each position is near land or not, with `NA` for positions where coordinates are missing. If `plot = TRUE`, a `ggplot` object is returned showing the land polygon, buffer area, and position points colored by their proximity to land.

See Also

`clean_shark4r_cache()` to manually clear cached shape files.

`iRfcb::ifcb_is_near_land` for the original function.

Examples

```
# Define coordinates
latitudes <- c(62.500353, 58.964498, 57.638725, 56.575338)
longitudes <- c(17.845993, 20.394418, 18.284523, 16.227174)

# Call the function
try(near_land <- positions_are_near_land(latitudes, longitudes, distance = 300, crs = 4326))

# Print the result
if (exists("near_land")) print(near_land)
```

`read_ptbx` *Read a Plankton Toolbox export file*

Description

This function reads a sample file exported as an Excel (.xlsx) file from Plankton Toolbox and extracts data from a specified sheet. The default sheet is "sample_data.txt", which contains count data.

Usage

```
read_ptbx(
  file_path,
  sheet = c("sample_data.txt", "sample_info.txt", "counting_method.txt",
    "Sample summary", "README")
)
```

Arguments

`file_path`	Character. Path to the Excel file.
`sheet`	Character. The name of the sheet to read. Must be one of: "sample_data.txt", "Sample summary", "sample_info.txt", "counting_method.txt", or "README". Default is "sample_data.txt".

Value

A `tibble` containing the contents of the selected sheet.

See Also

`https://nordicmicroalgae.org/plankton-toolbox/` for downloading Plankton Toolbox.

`https://github.com/planktontoolbox/plankton-toolbox/` for Plankton Toolbox source code.

Examples

```
# Read the default data sheet
sample_data <- read_ptbx(system.file("extdata/Anholt_E_2024-09-15_0-10m.xlsx",
                                     package = "SHARK4R"))

# Print output
sample_data


# Read a specific sheet
sample_info <- read_ptbx(system.file("extdata/Anholt_E_2024-09-15_0-10m.xlsx",
                                     package = "SHARK4R"),
                         sheet = "sample_info.txt")
# Print output
```

```
sample_info
```

---

read_shark *Read SHARK export files (tab- or semicolon-delimited, plain text or zipped)*

Description

Reads tab- or semicolon-delimited SHARK export files with standardized format. The function can handle plain text files (`.txt`) or zip archives (`.zip`) containing a file named `shark_data.txt`. It automatically detects and converts column types and can optionally coerce the `"value"` column to numeric. The `"sample_date"` column is converted to `Date` if it exists.

Usage

```
read_shark(
  filename,
  delimiters = "point-tab",
  encoding = "utf_8",
  guess_encoding = TRUE,
  value_numeric = TRUE
)
```

Arguments

`filename`	Path to the SHARK export file. Can be a `.txt` or `.zip` file. If a zip file is provided, it should contain a file named `shark_data.txt`.
`delimiters`	Character. Specifies the delimiter used in the file. Options: `"point-tab"` (tab-separated, default) or `"point-semi"` (semicolon-separated).
`encoding`	Character. File encoding. Options: `"cp1252"`, `"utf_8"`, `"utf_16"`, `"latin_1"`. Default is `"utf_8"`. If `guess_encoding = TRUE`, detected encoding overrides this value.
`guess_encoding`	Logical. If `TRUE` (default), automatically detect file encoding. If `FALSE`, the function uses only the user-specified encoding.
`value_numeric`	Logical. If `TRUE` (default), attempts to convert the `"value"` column to numeric. If `FALSE`, leaves `"value"` as-is.

Details

This function is robust to file encoding issues. By default (`guess_encoding = TRUE`), it attempts to automatically detect the file encoding and will use it if it differs from the user-specified `encoding`. Automatic detection can be disabled.

Value

A data frame containing the parsed contents of the SHARK export file, or `NULL` if the file is empty or could not be read.

See Also

`read_shark_deliv()` for reading SHARK Excel delivery files (`.xls`/`.xlsx`).

Examples

```
## Not run:
# Read a plain text SHARK export
df_txt <- read_shark("sharkweb_data.txt")

# Read a SHARK export from a zip archive
df_zip <- read_shark("shark_data.zip")

# Read with explicit encoding and do not convert value
df_custom <- read_shark("shark_data.txt",
                        encoding = "latin_1",
                        guess_encoding = FALSE,
                        value_numeric = FALSE)

## End(Not run)
```

`read_shark_deliv` *Read SHARK Excel delivery files (.xls or .xlsx)*

Description

Reads Excel files delivered to SHARK in a standardized format. The function automatically detects whether the file is `.xls` or `.xlsx` and reads the specified sheet, skipping a configurable number of rows. Column types are automatically converted, and if a column `"SDATE"` exists, it is converted to `Date`.

Usage

```
read_shark_deliv(filename, skip = 2, sheet = 2)
```

Arguments

`filename`	Path to the Excel file to be read.
`skip`	Minimum number of rows to skip before reading anything (column names or data). Leading empty rows are automatically skipped, so this is a lower bound. Ignored if `sheet` or `range` specifies a starting row. Default is 2.
`sheet`	Sheet to read. Either a string (sheet name) or integer (sheet index). If neither is specified, defaults to the second sheet.

Value

A data frame containing the parsed contents of the Excel file, or `NULL` if the file does not exist, is empty, or cannot be read.

See Also

read_shark() for reading SHARK tab- or semicolon-delimited export files or zip-archives.

Examples

```
## Not run:
# Read the second sheet of a .xlsx file (default)
df_xlsx <- read_shark_deliv("shark_delivery.xlsx")

# Read the first sheet of a .xls file, skipping 3 rows
df_xls <- read_shark_deliv("shark_delivery.xls", skip = 3, sheet = 1)

## End(Not run)
```

---

run_qc_app *Launch the SHARK4R Bio-QC Tool*

Description

This function launches the interactive Shiny application for performing quality control (QC) on SHARK data. The application provides a graphical interface for exploring and validating data before or after submission to SHARK.

Usage

```
run_qc_app(interactive = TRUE)
```

Arguments

interactive	Logical value whether the session is interactive or not.

Details

The function checks that all required packages for the app are installed before launching. If any are missing, the user is notified. In interactive sessions, the function will prompt whether the missing packages should be installed automatically. In non-interactive sessions (e.g. scripts or CI), the function instead raises an error and lists the missing packages so they can be installed manually.

Value

This function is called for its side effect of launching a Shiny application. It does not return a value.

Examples

```
# Launch the SHARK4R Bio-QC Tool
if(interactive()){
  run_qc_app()
}
```

`scatterplot` *Scatterplot with optional horizontal threshold lines*

Description

This function creates a scatterplot from a data frame, optionally coloring points by a grouping column and adding horizontal threshold lines. Supports both static `ggplot2` plots and interactive `plotly` plots with a linear/log toggle.

Usage

```
scatterplot(
  data,
  x = c("station_name", "sample_date"),
  parameter = NULL,
  hline = NULL,
  hline_group_col = NULL,
  hline_value_col = NULL,
  hline_style = list(linetype = "dashed", size = 0.8),
  max_hlines = 5,
  interactive = TRUE,
  verbose = TRUE
)
```

Arguments

`data`	A data.frame or tibble containing at least the following columns: `"station_name"`, `"sample_date"`, `"value"`, `"parameter"`, `"unit"`.
`x`	Character. The column to use for the x-axis. Either `"station_name"` or `"sample_date"`.
`parameter`	Optional character. If provided, only data for this parameter will be plotted. If `NULL`, the function will plot the first parameter found in the dataset.
`hline`	Numeric or data.frame. Horizontal line(s) to add. If numeric, a single line is drawn at that y-value. If a data.frame, must contain `hline_group_col` and `hline_value_col` columns.
`hline_group_col`	Character. Column used for grouping when `hline` is a data.frame and/or for coloring points (optional).

`hline_value_col`	Character. Column in `hline` used for the y-values of horizontal lines.
`hline_style`	List. Appearance settings for horizontal lines. Should contain `linetype` and `size`.
`max_hlines`	Integer. Maximum number of horizontal line groups to display per parameter when `hline` is a data.frame.
`interactive`	Logical. If TRUE, returns an interactive `plotly` plot; if FALSE, returns a static `ggplot2` plot.
`verbose`	Logical. If TRUE, messages will be displayed during execution. Defaults to TRUE.

Details

- If `hline` is numeric, a single horizontal line is drawn across the plot.
- If `hline` is a data.frame, only the first `max_hlines` groups (sorted alphabetically) are displayed.
- Points can be colored by `hline_group_col` if provided.
- Interactive plots include buttons to switch between linear and log y-axis scales.

Value

A `ggplot` object (if `interactive = FALSE`) or a `plotly` object (if `interactive = TRUE`).

See Also

`load_shark4r_stats` for loading threshold or summary statistics that can be used to define horizontal lines in the plot.

Examples

```
## Not run:
scatterplot(
  data = my_data,
  x = "station_name",
  parameter = "Chlorophyll-a",
  hline = c(10, 20)
)

scatterplot(
  data = my_data,
  x = "sample_date",
  parameter = "Bacterial abundance",
  hline = thresholds_df,
  hline_group_col = "location_sea_basin",
  hline_value_col = "P99"
)

## End(Not run)
```

`translate_shark_datatype` *Translate SHARK4R datatype names*

Description

Converts user-facing datatype names (e.g., "Grey seal") to internal SHARK4R names (e.g., "GreySeal") based on `SHARK4R:::.type_lookup`. See available user-facing datatypes in `get_shark_options()$dataTypes`.

Usage

```
translate_shark_datatype(x)
```

Arguments

`x`	Character vector of datatype names to translate

Value

Character vector of translated datatype names

Examples

```
# Example strings
datatypes <- c("Grey seal", "Primary production", "Physical and Chemical")

# Basic translation
translate_shark_datatype(datatypes)
```

`update_dyntaxa_taxonomy` *Update SHARK taxonomy records using Dyntaxa*

Description

This function updates Dyntaxa taxonomy records based on a list of Dyntaxa taxon IDs. It collects parent IDs from SLU Artdatabanken API (Dyntaxa), retrieves full taxonomy records, and organizes the data into a full taxonomic table that can be joined with data downloaded from SHARK

Usage

```
update_dyntaxa_taxonomy(
  dyntaxa_ids,
  subscription_key = Sys.getenv("DYNTAXA_KEY"),
  add_missing_taxa = FALSE,
  verbose = TRUE
)
```

Arguments

`dyntaxa_ids` A vector of Dyntaxa taxon IDs to update.

`subscription_key`
A Dyntaxa API subscription key. By default, the key is read from the environment variable `DYNTAXA_KEY`.

You can provide the key in three ways:

- **Directly as a parameter**: `update_dyntaxa_taxonomy(238366, subscription_key = "your_key_here")`.
- **Temporarily for the session**: `Sys.setenv(DYNTAXA_KEY = "your_key_here")`. After this, you do not need to pass `subscription_key` to the function.
- **Permanently across sessions** by adding it to your `~/.Renviron` file. Use `usethis::edit_r_environ()` to open the file, then add: `DYNTAXA_KEY=your_key_here`. After this, you do not need to pass `subscription_key` to the function.

`add_missing_taxa`
Logical. If TRUE, the function will attempt to fetch missing taxa (i.e., taxon_ids not found in the initial Dyntaxa DwC-A query). Default is FALSE.

`verbose` Logical. Print progress messages. Default is TRUE.

Details

A valid Dyntaxa API subscription key is required. You can request a free key for the "Taxonomy" service from the ArtDatabanken API portal: `https://api-portal.artdatabanken.se/`

Note: Please review the API conditions and register for access before using the API. Data collected through the API is stored at SLU Artdatabanken. Please also note that the authors of `SHARK4R` are not affiliated with SLU Artdatabanken.

Value

A `tibble` representing the updated Dyntaxa taxonomy table.

See Also

`get_shark_data`, `update_worms_taxonomy`, SLU Artdatabanken API Documentation

Examples

```
## Not run:
# Update Dyntaxa taxonomy for taxon IDs 238366 and 1010380
updated_taxonomy <- update_dyntaxa_taxonomy(c(238366, 1010380), "your_subscription_key")
print(updated_taxonomy)

## End(Not run)
```

`which_basin` *Determine if points are in a specified sea basin*

Description

This function is a **wrapper/re-export** of `iRfcb::ifcb_which_basin()`. The `iRfcb` package is only required if you want to actually call this function.

Usage

```
which_basin(latitudes, longitudes, plot = FALSE, shape_file = NULL)
```

Arguments

`latitudes`	A numeric vector of latitude points.
`longitudes`	A numeric vector of longitude points.
`plot`	A boolean indicating whether to plot the points along with the sea basins. Default is FALSE.
`shape_file`	The absolute path to a custom polygon shapefile in WGS84 (EPSG:4326) that represents the sea basin. Defaults to the Baltic Sea, Kattegat, and Skagerrak basins included in the `iRfcb` package.

Details

This function identifies which sub-basin a set of latitude and longitude points belong to, using a user-specified or default shapefile. The default shapefile includes the Baltic Sea, Kattegat, and Skagerrak basins and is included in the `iRfcb` package.

This function reads a pre-packaged shapefile of the Baltic Sea, Kattegat, and Skagerrak basins from the `iRfcb` package by default, or a user-supplied shapefile if provided. The shapefiles originate from SHARK (https://shark.smhi.se/en/). It sets the CRS, transforms the CRS to WGS84 (EPSG:4326) if necessary, and checks if the given points fall within the specified sea basin. Optionally, it plots the points and the sea basin polygons together.

Value

A vector indicating the basin each point belongs to, or a ggplot object if `plot = TRUE`.

See Also

`iRfcb::ifcb_which_basin` for the original function.

Examples

```
# Define example latitude and longitude vectors
latitudes <- c(55.337, 54.729, 56.311, 57.975)
longitudes <- c(12.674, 14.643, 12.237, 10.637)

# Check in which Baltic sea basin the points are in
points_in_the_baltic <- which_basin(latitudes, longitudes)
print(points_in_the_baltic)

# Plot the points and the basins
map <- which_basin(latitudes, longitudes, plot = TRUE)
```

Index