Package 'BOLDconnectR'

July 23, 2026

Title Retrieve, Transform and Analyze the Barcode of Life Data Systems Data

Version 1.0.1

Maintainer Sameer Padhye <spadhye@uoguelph.ca>

Description Facilitates retrieval, transformation and analysis of the data from the Barcode of Life Data Systems (BOLD) database <https://boldsystems.org/>. This package allows both public and private user data to be easily downloaded into the R environment using a variety of inputs such as: IDs (processid, sampleid), BINs, dataset codes, project codes, taxonomy, geography etc. It provides frictionless data conversion into formats compatible with other R-packages and third-party tools, as well as functions for sequence alignment & clustering, biodiversity analysis and spatial mapping.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.0.0)

Imports ape (>= 5.5), BAT (>= 2.0), data.table (>= 1.13), dplyr (>= 1.0.1), ggplot2 (>= 3.3.2), httr (>= 1.4.2), jsonlite (>= 1.7), maps (>= 3.3), methods, rlang, rnaturalearth, sf (>= 0.9.4), skimr (>= 2.1.2), tidyr (>= 1.1.1), utils, vegan (>= 2.5.7)

Suggests Biostrings, BiocManager, DT, msa, muscle, knitr, rmarkdown

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Sameer Padhye [aut, cre], Liliana Ballesteros-Mejia [aut], Timothy Agda [aut], Jireh Agda [aut], Sujeevan Ratnasingham [aut]

Repository CRAN

Date/Publication 2026-07-23 18:10:02 UTC

Contents

bold.analyze.align . . . 2
bold.analyze.diversity . . . 4
bold.analyze.map . . . 8
bold.analyze.tree . . . 10
bold.apikey . . . 13
bold.data.summarize . . . 14
bold.export . . . 16
bold.fetch . . . 19
bold.fields.info . . . 22
bold.full.search . . . 23
bold.public.search . . . 25
test.data . . . 27
test.data2 . . . 28

Index **29**

`bold.analyze.align` *Transform and align the sequence data retrieved from BOLD*

Description

Function designed to transform and align the sequence data retrieved from the function `bold.fetch`.

Usage

```
bold.analyze.align(
  bold_df,
  marker = NULL,
  align_method = c("ClustalOmega", "Muscle"),
  cols_for_seq_names = NULL,
  ...
)
```

Arguments

`bold_df`	A data frame obtained from `bold.fetch()`.
`marker`	A single character value specifying the gene marker for which the output is generated. Default is NULL (all data is used).
`align_method`	Character vector specifying the type of multiple sequence alignment algorithm to be used (ClustalOmega and Muscle available).
`cols_for_seq_names`	A single or multiple character vector specifying the column headers to be used to name each sequence in the fasta file. Default is NULL in which case, only the processid is used as a name.
`...`	additional arguments that can be passed to `msa::msa()` function.

Details

`bold.analyze.align` takes the sequence information obtained using `bold.fetch()` function and performs a multiple sequence alignment. It uses the `msa::msa()` function with default settings but additional arguments from the `msa` function can be passed through the `...` argument. The clustering method can be specified using the `align_method` argument, with options including `Muscle` and `ClustalOmega` (available via the `msa` package). The provided marker name must match the standard marker names (Ex. COI-5P) available on the BOLD webpage (Ratnasingham et al. 2024; pg.404). The name for individual sequences in the output can be customized by using the `cols_for_seq_names` argument. If multiple fields are specified, the sequence name will follow the order of fields given in the vector. Performing a multiple sequence alignment on large sequence data might slow (or crash) the system. Additionally, users are responsible for verifying the sequence quality and integrity, as the function does not automatically check for issues like STOP codons and indels within the data.

Note: . Users are required to install and load the `Biostrings`, `msa` and `muscle` packages using `BiocManager` before running this function.

Value

- bold_df.mod = A modified BCDM data frame with two additional columns ('aligned_seq' and 'msa.seq.name').

References

Ratnasingham S, Wei C, Chan D, Agda J, Agda J, Ballesteros-Mejia L, Ait Boutou H, El Bastami Z M, Ma E, Manjunath R, Rea D, Ho C, Telfer A, McKeowan J, Rahulan M, Steinke C, Dorsheimer J, Milton M, Hebert PDN . "BOLD v4: A Centralized Bioinformatics Platform for DNA-Based Biodiversity Data." In DNA Barcoding: Methods and Protocols, pp. 403-441. Chapter 26. New York, NY: Springer US, 2024.

Examples

```
## Not run:
# Search for ids
seq.data.ids <- bold.public.search(taxonomy = list(
  "Oreochromis tanganicae",
  "Oreochromis karongae"
))
# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

seq.data <- bold.fetch(
  get_by = "processid",
  identifiers = seq.data.ids$processid
)

# R packages `msa` and `Biostrings` are required for this function to run.
# For `align_method` = "Muscle", package `muscle` is required as well.
```

```
# Both the packages are installed using `BiocManager`.

# Align the data (using  bin_uri as the name for each sequence)
seq.align <- bold.analyze.align(seq.data,
  cols_for_seq_names = c("bin_uri"),
  align_method = "ClustalOmega"
)

# Dataframe of the sequences (aligned) with their corresponding names
head(seq.align[, c("aligned_seq", "msa.seq.name")])

## End(Not run)
```

`bold.analyze.diversity`

Create a biodiversity profile of the retrieved data

Description

This function creates a biodiversity profile of the downloaded data using `bold.fetch()`.

Usage

```
bold.analyze.diversity(
  bold_df,
  taxon_rank,
  taxon_name = NULL,
  site_type = c("locations", "grids"),
  location_type = NULL,
  gridsize = NULL,
  presence_absence = FALSE,
  diversity_profile = c("richness", "preston", "shannon", "beta", "all"),
  beta_index = NULL
)
```

Arguments

`bold_df`	A data frame obtained from `bold.fetch()`.
`taxon_rank`	A single character string specifying the taxonomic hierarchical rank. Needs to be provided by default.
`taxon_name`	A single or multiple character vector specifying the taxonomic names associated with the 'taxon_rank'. Default value is NULL.
`site_type`	A character string specifying one of two broad categories of `sites` (`locations` or `grids`). Needs to be provided by default.

`location_type` A single character vector specifying the geographic category if `locations` is selected as the `site_type` and for which a community matrix should be created. Default value is NULL.

`gridsize` A numeric value of the size of the grid if `grids` is selected as the `site_type`. Size is in sq.m. Default value is NULL.

`presence_absence` A logical value specifying whether the generated matrix should be converted into a 'presence-absence' matrix. Default value is FALSE.

`diversity_profile` A character string specifying the type of diversity profile ("richness","preston","shannon","beta","all"). Needs to be provided by default.

`beta_index` A character vector specifying the type of beta diversity index ('jaccard' or 'sorensen' available) if `beta` or `all` `diversity_profile` selected. Default value is NULL.

Details

`bold.analyze.diversity` estimates the richness, Shannon diversity and beta diversity from the BIN counts or presence-absence data. Internally, the function converts the downloaded BCDM data into a community matrix (site X species) which is also provided as a part of the output. `taxon_rank` refers to a specific taxonomic rank (Ex. class, order, family etc or even BINs) and the `taxon_name` to one or more names of organisms in that specific rank. `taxon_rank` cannot be NULL while all the data will be used if `taxon_name = NULL` for a specified `taxon_rank`. A mismatch between `taxon_rank` and `taxon_name` will throw an error (e.g., `taxon_rank` is `species` and `taxon_name` is `Oreochromis` which is a generic name). The `site_type=locations` followed by providing a `location_type` refers to any geographic field (country.ocean,province.state etc.; for more information check the `bold.fields.info()` function help). `site_type=grids` generates grids based on BIN occurrence data (latitude, longitude) with grid size determined by the user in square meters using the `gridsize` argument. `site_type=grids` converts the Coordinate Reference System (CRS) of the data to a 'Mollweide' projection by which distance-based grid can be correctly specified (Gott III et al. 2007).Each grid is assigned a cell id, with the lowest number given to the lowest latitudinal point in the dataset. Rows lacking latitude and longitude data (NULL values) are removed when `site_type=grids`. Conversely, NULL entries are permitted when `site_type=locations`, even if latitude and longitude values are missing. This distinction exists because grids rely on bounding boxes, which require latitude and longitude values. This filtering could impact the richness values and other analyses, as all records for the selected `taxon_rank` that contain `location` information but lack latitude and longitude will be excluded if `site_type=grids`. This means that the same dataset could yield different results depending on the chosen `site_type`. `location_type` has to be specified when `site_type=locations` to avoid errors. The community matrix generated based on the sites/grids is then used to create richness profiles using `BAT::alpha.accum()` and Preston and Shannon diversity analyses using `vegan::prestondistr()` and `vegan::diversity()` respectively. The `BAT::alpha.accum()` currently offers various richness estimators, including Observed diversity (Obs); Singletons (S1); Doubletons (S2); Uniques (Q1); Duplicates (Q2); Jackknife1 abundance (Jack1ab); Jackknife1 incidence (Jack1in); Jackknife2 abundance (Jack2ab); Jackknife2 incidence (Jack2in); Chao1 and Chao2. The results depend on the input data (true abundances vs counts vs incidences) and users should be careful in the subsequent interpretation. Preston plots are generated using the data from the `prestondistr` results in `ggplot2` featuring cyan bars for observed species (or equivalent taxonomic group) and orange dots for expected counts. Beta diversity values are calculated using `BAT::beta()` function, which partitions the data using the Podani &

Schmera (2011)/Carvalho et al. (2012) approach. These results are stored as distance matrices in the output.

Note on the community matrix: Each cell in this matrix contains the counts (or abundances) of the specimens whose sequences have an assigned BIN, in a given `site_type` (`locations` or `grids`). These counts can be generated at any taxonomic hierarchical level, applicable to one or multiple taxa including `bin_uri`. The `presence_absence` argument converts these counts (or abundances) to 1s and 0s.

Important Note: Results, including counts, adapt based on `taxon_rank` argument.

Value

An 'output' list containing results based on the profile selected:

#Common to all

- comm.matrix = site X species like matrix required for the biodiversity results #Common to all if `site_type=grids`
- comm.matrix = site X species like matrix required for the biodiversity results

#Based on the type of diversity profile #1. richness

- richness = A richness profile matrix #2. shannon
- Shannon_div = Shannon diversity values for the given sites/grids (from gen.comm.mat) #3. preston
- preston.res = a Preston plot numerical data output
- preston.plot = a ggplot2 visualization of the preston.plot #4. beta
- total.beta = beta.total
- replace = beta.replace (replacement)
- richnessd = beta.richnessd (richness difference) #5. all
- All of the above results

References

Carvalho, J.C., Cardoso, P. & Gomes, P. (2012) Determining the relative roles of species replacement and species richness differences in generating beta-diversity patterns. Global Ecology and Biogeography, 21, 760-771.

Podani, J. & Schmera, D. (2011) A new conceptual and methodological framework for exploring and explaining pattern in presence-absence data. Oikos, 120, 1625-1638.

Richard Gott III, J., Mugnolo, C., & Colley, W. N. (2007). Map projections minimizing distance errors. Cartographica: The International Journal for Geographic Information and Geovisualization, 42(3), 219-234.

Examples

```
## Not run:
# Search for ids
comm.mat.data <- bold.public.search(taxonomy = list("Poecilia"))

# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

BCDMdata <- bold.fetch(
  get_by = "processid",
  identifiers = comm.mat.data$processid
)

# Remove rows which have no species data
BCDMdata <- BCDMdata[!BCDMdata$species == "", ]

# 1. Analyze richness data
res.rich <- bold.analyze.diversity(
  bold_df = BCDMdata,
  taxon_rank = "species",
  site_type = "locations",
  location_type = "country.ocean",
  diversity_profile = "richness"
)

# Community matrix (BCDM data converted to community matrix)
res.rich$comm.matrix

# richness results
res.rich$richness

# 2. Shannon diversity (based on grids)
res.shannon <- bold.analyze.diversity(
  bold_df = BCDMdata,
  taxon_rank = "species",
  site_type = "grids",
  gridsize = 1000000,
  diversity_profile = "shannon"
)

# Shannon diversity results
res.shannon$shannon_div

# Grid data (sf)
res.shannon$grids.data

# grid map
res.shannon$grid.map
```

```
# 3. Preston plots and results
pres.res <- bold.analyze.diversity(
  bold_df = BCDMdata,
  taxon_rank = "species",
  site_type = "locations",
  location_type = "country.ocean",
  diversity_profile = "preston"
)

# Preston plot
pres.res$preston.plot

# Preston plot data
pres.res$preston.res

# 4. beta diversity
beta.res <- bold.analyze.diversity(
  bold_df = BCDMdata,
  taxon_rank = "species",
  site_type = "locations",
  location_type = "country.ocean",
  diversity_profile = "beta",
  beta_index = "jaccard"
)

# Total diversity
beta.res$total.beta

# Replacement
beta.res$replace

# Richness difference
beta.res$richnessd

# 5. All profiles
all.diversity.res <- bold.analyze.diversity(
  bold_df = BCDMdata,
  taxon_rank = "species",
  site_type = "locations",
  location_type = "country.ocean",
  diversity_profile = "all",
  beta_index = "jaccard"
)
# Explore all results
all.diversity.res

## End(Not run)
```

`bold.analyze.map` *Visualize BIN occurrence data on maps*

Description

This function creates basic maps of BIN occurrences at different scales.

Usage

```
bold.analyze.map(bold_df, country = NULL, bbox = NULL)
```

Arguments

bold_df	The data.frame retrieved from bold.fetch().
country	A single or multiple character vector of country names. Default value is NULL.
bbox	A numeric vector specifying the min, max values of the latitude and longitude. Default value is NULL.

Details

bold.analyze.map extracts out the geographic information from the bold.fetch() output to generate an occurrence map. Data points having NA values for either latitude or longitude or both are removed. Latitude and longitude values are in 'decimal degrees' format with a 'WGS84' Coordinate Reference System (CRS) projection. Default view includes data mapped onto a world shape file using the rnaturalearth::ne_countries() at a 110 scale (low resolution). If the country is specified (single or multiple values), the function will specifically plot the occurrences on the specified country. Alternatively, a bounding box (bbox) can be defined for a specific region to be visualized (First two elements of the bbox are longitude values (xmin and xmax) and the remaining two are latitude values (ymin and ymax)). The function also provides a sf data frame of the GIS data which can be used for any other application/s. For names of countries, please refer to https://www.geonames.org/.

Value

An 'output' list containing:

- geo.df = A simple features (sf) 'data.frame' containing the geographic data.
- plot = A visualization of the occurrences.

Examples

```
## Not run:
# Download the ids
geo_data.ids <- bold.public.search(taxonomy = list("Musca domestica"))

# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

geo_data <- bold.fetch(
  get_by = "processid",
  identifiers = geo_data.ids$processid
```

```
)

# All data plotted.
geo.viz <- bold.analyze.map(geo_data)
# View plot
geo.viz$plot

# Data plotted only in one country
geo.viz.country <- bold.analyze.map(geo_data,
  country = c("Saudi Arabia")
)
# View plot
geo.viz.country$plot
# The sf dataframe of the downloaded data
geo.viz$geo.df

# Data plotted based on a bounding box
bold.analyze.map(
  bold_df = geo_data,
  bbox = c(41, 100, 20.36501, 55.506)
)

## End(Not run)
```

`bold.analyze.tree` *Analyze and visualize the multiple sequence alignment*

Description

Calculates genetic distances and performs a Neighbor Joining (NJ) tree estimation of the multiple sequence alignment output obtained from `bold.analyze.align()`.

Usage

```
bold.analyze.tree(
  bold_df,
  dist_model,
  clus_method = c("nj", "njs"),
  save_dist_mat = FALSE,
  newick_tree_export = NULL,
  tree_plot = FALSE,
  tree_plot_type,
  ...
)
```

Arguments

`bold_df`	A modified BCDM data frame obtained from `bold.analyze.align()`.

dist_model	A character string specifying the model to generate the distances.
clus_method	A character string specifying either nj (neighbour joining) or njs (neighbour joining with NAs) clustering algorithm.
save_dist_mat	A logical value specifying whether the distance matrix should be saved in the output. Default value is FALSE.
newick_tree_export	A character string specifying the folder path where the file should be saved along with the name and extension (.nwk) for the file. Default value is NULL.
tree_plot	Logical value specifying if a neighbor joining plot should be generated. Default value is FALSE.
tree_plot_type	A character string specifying the layout of the tree. Needs to be provided by default.
...	additional arguments from ape::dist.dna.

Details

bold.analyze.tree analyzes the multiple sequence alignment output of the bold.analyze.align() function to generate a distance matrix using the models available in the ape::dist.dna(). The default dist_model is K80 (Kimura 1980 model). Two forms of Neighbor Joining clustering are currently available (ape::nj() & ape::njs()). save_dist_mat= TRUE will store the underlying distance matrix in the output; however, the default value for the argument is deliberately kept at FALSE to avoid potential memory issues with large data. newick_tree_export will save the tree in a newick format locally. Data path with the name of the file should be provided (Ex. 'C:/Users/xyz/Desktop/newickoutput' for Windows). Setting tree_plot= TRUE generates a basic visualization of the Neighbor Joining (NJ) tree using the distance matrix from ape::dist.dna() and the ape::plot.phylo() function. tree_plot_type specifies the type of tree and has the following options ("phylogram", "cladogram", "fan", "unrooted", "radial", "tidy" based on type argument of ape::plot.phylo(); The first alphabet can be used instead of the whole word). Both ape::nj() and ape::njs() are available for generating the tree. Additional arguments for calculating distances can be passed to ape::dist.dna() using the ... argument (arguments such as gamma, pairwise.deletion & base.freq). The function also provides base frequencies from the data.

Value

An 'output' list containing:

- dist_mat = A distance matrix based on the model selected if save_dist_mat=TRUE.
- base_freq = Overall base frequencies of the align.seq result.
- plot = Neighbor Joining clustering visualization (if tree_plot=TRUE).
- data_for_plot = A phylo object used for the plot.
- NJ/NJS tree in a newick format (only if newick_tree_export=TRUE).

Examples

```
## Not run:
# Download the data ids
seq.data.ids <- bold.public.search(taxonomy = list(
  "Oreochromis tanganicae",
  "Oreochromis karongae"
))

# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

seq.data <- bold.fetch(
  get_by = "processid",
  identifiers = seq.data.ids$processid,
  filt_marker = "COI-5P"
)

# Remove rows without species name information
seq <- seq.data[seq.data$species != "", ]

# Align the data
# Users need to install and load packages `msa` and `Biostrings`.
# For `align_method` = "Muscle", package `muscle` is required as well.

seq.align <- bold.analyze.align(
  bold_df = seq.data,
  marker = "COI-5P",
  align_method = "ClustalOmega",
  cols_for_seq_names = c("species", "bin_uri")
)

# Analyze the data to get a tree

seq.analysis <- bold.analyze.tree(
  bold_df = seq.align,
  dist_model = "K80",
  clus_method = "nj",
  tree_plot = TRUE,
  tree_plot_type = "p",
  save_dist_mat = T,
  pairwise.deletion = T
)

# Output
# A ‘phylo’ object of the plot
seq.analysis$data_for_plot
# A distance matrix based on the distance model selected
seq.analysis$save_dist_mat
# Base frequencies of the sequences
```

```
seq.analysis$base_freq

## End(Not run)
```

`bold.apikey` *Set the BOLD private data API key*

Description

Stores the BOLD-provided access token 'apikey' in a variable, making it available for use in other functions within the R session.

Usage

```
bold.apikey(apikey)
```

Arguments

`apikey`	A character string required for authentication and data access.

Details

`bold.apikey` creates a variable called `apikey` that stores the access token provided by BOLD. This apikey variable is then used internally by the `bold.fetch()` and `bold.full.search()` functions, so that the user does not have to input it again. To set the `apikey`, the token must be provided as an input for the function before any other functions are called. The `apikey` is a UUID v4 hexadecimal string and is valid for few months, after which it must be renewed.

Obtaining the API key: The API key is found in the BOLD Workbench(`https://bench.boldsystems.org/index.php/Login/page?destination=MAS_Management_UserConsole`). After logging in, navigate to `Your Name` (located at the top left-hand side of the window) and click `Edit User Preferences`. You can find the API key in the `User Data` section.

Please note: To have an API key available in the workbench, a user must have uploaded ~ 10K records to BOLD, though, in case there aren't those many submissions on BOLD, the user can email BOLD support to request for a token. Such requests will be assessed on a case by case basis.

Value

Token saved as 'apikey'

Examples

```
## Not run:

# This example below is for documentation only

bold.apikey("00000000-0000-0000-0000-000000000000")
```

```
## End(Not run)
```

`bold.data.summarize` *Generate specific summaries from the downloaded BCDM data*

Description

The function is used to obtain a different types of data summaries for the downloaded BCDM data via `bold.fetch` function.

Usage

```
bold.data.summarize(
  bold_df,
 summary_type = c("concise_summary", "detailed_taxon_counts", "barcode_summary",
    "data_completeness"),
  primer_f = NULL,
  primer_r = NULL,
  rem_na_bin = FALSE
)
```

Arguments

`bold_df`	the data.frame retrieved from the `bold.fetch()` function.
`summary_type`	A character string specifying the type of summary required ('concise_summary', 'detailed_taxon_counts','barcode_summary','data_completeness','all')
`primer_f`	A character string specifying the forward primer. Default value is NULL.
`primer_r`	A character string specifying the reverse primer. Default value is NULL.
`rem_na_bin`	A logical value specifying whether NA BINs should be removed from the BCDM dataframe. Default value is FALSE.

Details

`bold.data.summarize` provides different types of data summaries for the downloaded BCDM dataset. Current options include:

- concise_summary = A high level overview of the downloaded data that would include total records, counts of unique BINs, countries , institutes etc.
- data_completeness = A data profile that includes information on missing data, proportion of complete cases for each field in the BCDM data along with data type specific insights like distribution, average and median values for numeric data. Also provides a bar chart visualizing the missing data and total records.
- detailed_taxon_counts = Taxonomy focused counts of total records with and without BINs, unique countries and institutes.

- barcode_summary = BIN focused summary of nucleotide basepair length, ambiguous basepair number (if present), presence of primer sequences (forward and/or reverse) in the sequence along with the processid, country and institute associated with the BIN. `rem_na_bin`= TRUE removes all records that don't have a BIN (Please note that this might result into empty data frames sometimes due to lot of missing data). The forward or reverse primer also needs to be specified. Details on all/specific fields can be checked using the `bold.field.info()`.

Note: . Users are required to install and load the `Biostrings` package in case they want to generate the `barcode_summary` before running this function. For the data in the `nuc_basecount` column in the `barcode_summary`, please refer to the `bold.field.info()` for details.

Value

An output list containing:

- A data frame of detailed summary based on the `summary_type`
- A bar chart in case `summary_type` = `data_completeness` in addition to the dataframe.

Examples

```
## Not run:
bold_data.ids <- bold.public.search(taxonomy = list("Oreochromis"))

# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

bold.data <- bold.fetch(
  get_by = "processid",
  identifiers = bold_data.ids$processid
)

# 1. Generate a concise summary of the data

test.data.summary.concise <- bold.data.summarize(
  bold_df = bold.data,
  summary_type = "concise_summary"
)
# Result
test.data.summary.concise$concise_summary


# 2. Generate a detailed taxon counts summary

test.data.summary <- bold.data.summarize(
  bold_df = bold.data,
  summary_type = "detailed_taxon_counts"
)

# Result
```

```
test.data.summary$detailed_taxon_counts


# 3. Generate data completeness profile

test.data.summary.completeness <- bold.data.summarize(
  bold_df = bold.data,
  summary_type = "data_completeness"
)

# Results
# Summary
test.data.summary.completeness$completeness_summary

# Plot
test.data.summary.completeness$completeness_plot


# 4. Barcode summary (forward primer LCO1490)

# Users need to first load the package `Biostrings`

test.data.summary.barcode <- bold.data.summarize(
  bold_df = bold.data,
  summary_type = "barcode_summary",
  primer_f = "GGTCAACAAATCATAAAGATATTGG"
)

# Results
test.data.summary.barcode$barcode_summary

## End(Not run)
```

`bold.export` *Export files generated by BOLDconnectR*

Description

The function is used to export some of the output data generated by BOLDconnectR

Usage

```
bold.export(
  bold_df,
  export_type = c("preset_df", "msa", "fas"),
  presets = NULL,
  cols_for_fas_names = NULL,
  export
)
```

Arguments

bold_df	The data.frame either retrieved from bold.fetch(),bold.analyze.align or a user modified BCDM dataset.
export_type	A character input specifying the type of output required. Should be either of "preset_df","msa" or "fas".
presets	A single character vector specifying a preset for which a data summary is sought (Check the details section for more information). Default value is NULL.
cols_for_fas_names	A single or multiple character vector indicating the column headers that should be used to name each sequence for the unaligned FASTA file. Default is NULL; in this case, only the processid is used as the name.
export	A character value specifying the data file path and the name for the file. Extension should be included.

Details

bold.export offers an added export option for some of the sequence-based outputs obtained from functions within the BOLDconnectR package as well as a preset defined modified BCDM dataframe. Sequence information from the BCDM data downloaded via bold.fetch() can be directly exported as an unaligned FASTA file with export_type=fas, while the aligned sequences (in the modified BCDM dataframe) obtained from bold.analyze.align can be exported as a FASTA file with export_type=msa. The FASTA headers for individual sequences when export_type=fas can be customized by using the cols_for_fas_names argument. If more than one field is specified, the name will follow the sequence of the fields given in the vector. The multiple sequence aligned FASTA file uses the same name provided by the user in the bold.analyze.align() function and using the cols_for_fas_names argument in this case will throw an error. presets can be considered as collections of predefined columns from the fetched BCDM data that relate to a common theme. The number of columns in each preset varies based on data availability. There are six presets currently available in the package (taxonomy, geography, sequences, attributions, ecology_biogeography & other_meta_data). Fields included in each preset is as follows:

- taxonomy = "kingdom", "phylum", "class", "order", "family", "subfamily", "genus", "species", "bin_uri".
- geography = "country.ocean", "country_iso", "province.state", "region", "sector", "site", "site_code", "coord", "coord_accuracy", "coord_source".
- sequences = "nuc", "nuc_basecount", "marker_code", "sequence_run_site", "sequence_upload_date".
- attributions = "inst", "identification", "identification_method", "identification_rank", "identified_by", "collectors".
- ecology_biogeography = "elev", "elev_accuracy", "depth", "depth_accuracy", "habitat", "ecoregion", "biome", "realm", "coord", "coord_source".
- other_meta_data = "notes", "taxonomy_notes", "funding_src", "voucher_type", "tissue_type", "sampling_protocol". "processids" and "sampleids" are present in all the presets. Only one preset can be used at a time. presets should be NULL when exporting a FASTA file to avoid errors . Tabular data can be exported as a csv/tsv file. Data path with the name of the output file with the corresponding file extension (csv or tsv) should be provided (Ex. 'C:/Users/xyz/Desktop/fetch_data_output.csv' for Windows). This functionality is developed with the future potential of uploading data to BOLD using the package.

Value

It exports a .fas or a csv/tsv file based on the export argument.

Examples

```
## Not run:
# Download the records
data_for_export_ids <- bold.public.search(taxonomy = list("Poecilia reticulata"))

# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

# Fetching the data using the ids
data_for_export <- bold.fetch(
  get_by = "processid",
  identifiers = data_for_export_ids$processid
)

# 1. Export the BCDM data using 'presets' as a csv file
bold.export(
  bold_df = data_for_export,
  export_type = "preset_df",
  presets = "taxonomy",
  export = file.path(tempdir(), "file_path_with_intended_name.csv")
)

# 2. Export the fasta file (unaligned)
# Note that input data here is the original BCDM data (data_for_export)
bold.export(
  bold_df = data_for_export,
  export_type = "fas",
  cols_for_fas_names = c("bin_uri", "genus", "species"),
  export = file.path(tempdir(), "file_path_with_intended_name.fas")
)

# 3. Export multiple sequence alignment
# a. Align the data
# (using processid and bin_uri as fields for sequence names)
# Users need to install and load packages `msa` and `Biostrings` before using bold.analyze.align.
seq_align <- bold.analyze.align(data_for_export,
  cols_for_seq_names = c("processid", "bin_uri"),
  align_method = "ClustalOmega"
)

# b. Export the multiple sequence alignment
# Note the input data here is the modified BCDM data (seq_align)
bold.export(
  bold_df = seq_align,
  export_type = "msa",
```

```
  export = "file_path_with_intended_name.fas"
) #'

## End(Not run)
```

---

bold.fetch *Retrieve data from the BOLD database*

Description

Retrieves public and private user data based on different parameter (processid, sampleid, dataset or project codes & bin_uris) input.

Usage

```
bold.fetch(
  get_by,
  identifiers,
  cols = NULL,
  export = NULL,
  na.rm = FALSE,
  filt_taxonomy = NULL,
  filt_geography = NULL,
  filt_latitude = NULL,
  filt_longitude = NULL,
  filt_shapefile = NULL,
  filt_institutes = NULL,
  filt_identified.by = NULL,
  filt_seq_source = NULL,
  filt_marker = NULL,
  filt_collection_period = NULL,
  filt_basecount = NULL,
  filt_altitude = NULL,
  filt_depth = NULL
)
```

Arguments

get_by	A character string specifying the parameter used to fetch data (“processid”, “sampleid”, "bin_uris", "dataset_codes" or "project_codes")
identifiers	A vector (or a data frame column) pointing to the get_by parameter specified.
cols	A single or multiple character vector specifying columns needed in the final dataframe. Default value is NULL.
export	A character string specifying the data path where the file should be exported locally along with the name of the file with extension (csv or tsv). Default value is NULL.

`na.rm` A logical value specifying whether NA values should be removed from the BCDM dataframe. Default value is FALSE.

`filt_taxonomy` A single or multiple character vector of taxonomic names at any hierarchical level. Default value is NULL.

`filt_geography` A single or multiple character vector specifying any of the country/province/state/region/sector/site names/codes. Default value is NULL.

`filt_latitude` A single or a vector of two numbers specifying the latitudinal range in decimal degrees. Values should be separated by a comma. Default value is NULL.

`filt_longitude` A single or a vector of two numbers specifying the longitudinal range in decimal degrees. Values should be separated by a comma. Default value is NULL.

`filt_shapefile` A file path pointing to a shapefile. Default value is NULL.

`filt_institutes`
A single or multiple character vector specifying names of institutes. Default value is NULL.

`filt_identified.by`
A single or multiple character vector specifying names of people responsible for identifying the organism. Default value is NULL.

`filt_seq_source`
A single or multiple character vector specifying the data portals from where the (sequence) data was mined. Default value is NULL.

`filt_marker` A single or multiple character vector specifying gene names. Default value is NULL.

`filt_collection_period`
A single or a vector of two date values specifying the collection period range (start, end). Values should be separated by a comma. Default value is NULL.

`filt_basecount` A single or a vector of two numbers specifying range of number of basepairs. Val- ues should be separated by a comma. Default value is NULL.

`filt_altitude` A single or a vector of two numbers specifying the altitude range in meters. Values should be separated by a comma. Default value is NULL.

`filt_depth` A single or a vector of two numbers specifying the depth range. Values should be separated by a comma. Default value is NULL.

Details

`bold.fetch` retrieves both public as well as private user data, where private data refers to data that the user has permission to access. The data is downloaded in the Barcode Core Data Model (BCDM) format. It supports effective download data in bulk using search parameters like 'processid', 'sampleid', 'bin_uris', 'dataset_codes' and 'project_codes' through the `get_by` argument. Users must specify only one of the parameters at a time for retrieval. Multi-parameter searches combining fields like 'processid'+ 'sampleid' + 'bin_uris' are not supported, regardless of the parameters available. Data input is via the `identifier` argument and it can either be a single or multiple character vector containing data for one of the parameters. A dataframe column can be used as an input using the '$' operator (e.g., df$column_name). It is important to correctly match the `get_by` and `identifiers` arguments to avoid getting any errors. The `filt_` or filter parameter arguments provide further data sorting by which a specific user defined data can be obtained. Note that any/all

filt_argument names must be written explicitly to avoid any errors (Ex. filt_institutes = 'CBG' instead of just 'CBG'). Using the cols argument allows users to select specific columns for inclusion in the final data frame. If this argument is left as NULL all columns will be downloaded. Providing a data path for the export argument will save the data locally. Data path with the name of the output file with the corresponding file extension (csv or tsv) should be provided (Ex. 'C:/Users/xyz/Desktop/fetch_data_output.csv' for Windows). There is a hard limit of 1 million records that can be downloaded in a single instance. Download speeds for very large requests for bin_uris, dataset_codes and project_codes will be throttled, resulting in more time for fetching the data. Download speed would also depend on the user's internet connection and computer specifications. Downloaded data includes information (wherever available) for the columns given in the field column of the bold.fields.info() in the BCDM format. Metadata on the columns fetched in the downloaded data can also be obtained using bold.fields.info().

Important Note: bold.apikey() should be run prior to running bold.fetch to setup the apikey which is needed for the latter.

Value

A data frame containing all the information related to the processids/sampleids and the filters applied (if/any).

Examples

```
## Not run:
# Test data with processids
data(test.data)

# Fetch the data using the ids.
# 1. api_key must be obtained from BOLD support before using `bold.fetch()` function.
# 2. Use the `bold.apikey()` function  to set the apikey in the global env.

bold.apikey("apikey")

# With processids
res <- bold.fetch(
  get_by = "processid",
  identifiers = test.data$processid
)


# With sampleids
res <- bold.fetch(
  get_by = "sampleid",
  identifiers = test.data$sampleid
)

# With datasets (publicly available dataset provided)
res <- bold.fetch(
  get_by = "dataset_codes",
  identifiers = "DS-IBOLR24"
)
```

```
## Using filters

# Geography
res <- bold.fetch(
  get_by = "processid",
  identifiers = test.data$processid,
  filt_geography = "Churchill"
)

# Sequence length
res <- bold.fetch(
  get_by = "processid",
  identifiers = test.data$processid,
  filt_basecount = c(500, 600)
)

# Gene marker & sequence length
res <- bold.fetch(
  get_by = "processid",
  identifiers = test.data$processid,
  filt_marker = "COI-5P",
  filt_basecount = c(500, 600)
)

## End(Not run)
```

`bold.fields.info` *Retrieve metadata of the BOLD data fields*

Description

Provides information on the field (column) names and their respective data type, all of which are compliant with the Barcode Core Data Model (BCDM), the latest data model of the BOLD database.

Usage

```
bold.fields.info(print.output = FALSE)
```

Arguments

`print.output`	Whether the output should be printed in the console. Default is FALSE.

Details

The function downloads the latest field (column) meta data (file type and brief description) which is currently available for download from BOLD.print,output = TRUE will print the information in the console.

Value

A data frame containing information on all fields (columns).

Examples

```
bold.field.data <- bold.fields.info()
head(bold.field.data, 10)
```

`bold.full.search` *Search user based (private) and publicly available data on the BOLD database*

Description

Retrieves record ids accessible to the particular user along with publicly available data based on taxonomy, geography, markers and collection dates

Usage

```
bold.full.search(
  taxonomy = NULL,
  geography = NULL,
  marker = NULL,
  marker_min_length = NULL,
  marker_max_length = NULL,
  collection_start_date = NULL,
  collection_end_date = NULL,
  institutes = NULL
)
```

Arguments

`taxonomy`	A list of single or multiple character strings specifying the taxonomic names at any hierarchical level. Default value is NULL.
`geography`	A list of single or multiple character strings any of the country/province/state/region/sector codes. Default value is NULL.
`marker`	A character string specifying the particular marker gene. Default value is NULL.
`marker_min_length`	A numerical value of the minimum length of the specified marker gene. Default value is NULL.
`marker_max_length`	A numerical value of the maximum length of the specified marker gene. Default value is NULL.
`collection_start_date`	A date value specifying the start date of a date range. Default value is NULL.

collection_end_date
: A date value specifying the end date of a date range. Default value is NULL.

institutes
: A list of single or multiple character strings specifying the institute names. Default value is NULL.

Details

bold.full.search() searches user accessible and publicly available data on BOLD, retrieving proccessid and their respective marker codes. All the BCDM data can then be retrieved using the processids as inputs for the bold.fetch() function. Search parameters can include one or a combination of taxonomy, geography, institutes, markers, marker lengths and collection dates. Taxonomy, geography and institutes inputs are provided as lists if provided as text values directly (Ex. taxonomy = list("Panthera","Poecilia")). A dataframe column can also be used as an input for these 3 parameters using the '$' operator (e.g., df$column_name). If this is the case (i.e. df$colum_name), as.list should be used instead of just list (Ex. taxonomy = as.list (df$column_name), geography = as.list(df$column_name)) in order to preserve the query structure for internal operation. The function will return NULL output unless this is adhered to. Marker is given as a character value. Names of the markers can be found on the BOLD webpage. Marker length and collection dates are provided as numeric and character values respectively. Marker lengths without specifying the marker will generate an error but just specifying the marker will provide all the data for that marker available based on the user access. In such a case, the function has a default minimum length of 5 and a maximum length of 2000 base pairs. If one of the two lengths are provided, default value of the other will be used (Example: if marker_min_length is given by the user, the default marker_maximum_length will be 2000). Collection dates work in a similar way with a default start date of '2000-01-01' and a end date of '2075-01-01'.Any misspellings (Ex. Canadaa),incorrect placements of search terms (Ex. taxa names in geography) or terms having no data on BOLD at the time when the function is executed will result in a NULL output. There is a hard limit of 1 million record downloads for each search. Download speed would depend on the user's internet connection and computer specifications.

Value

A data frame containing all the processids and marker codes related to the query search.. *Important Note*: bold.apikey() should be run prior to running bold.full.search() to setup the apikey which is needed for the latter.

Examples

```
## Not run:
# Taxonomy
bold.data.tax <- bold.full.search(taxonomy = list("Panthera leo"))

# Result
head(bold.data.tax, 10)

# Taxonomy and Geography
bold.data.taxo.geo <- bold.full.search(
  taxonomy = list("Panthera uncia"),
  geography = list("India")
)
```

```
# Result
head(bold.data.taxo.geo, 10)

# Taxonomy, Geography and marker
bold.data.taxo.geo.marker <- bold.full.search(
  taxonomy = list("Poecilia reticulata"),
 geography = list("India"), marker = "COI-5P", marker_min_length = 300, marker_max_length = 700
)

# Result
bold.data.taxo.geo.marker

# Input as a dataframe column
df_test <- data.frame(
  taxon_name = c("Panthera leo", "Panthera uncia"),
  locations = c("India", "Sri Lanka")
)

# Result (correct way)
bold.data.taxo.geo.df.col <- bold.full.search(
  taxonomy = as.list(df_test$taxon_name),
  geography = as.list(df_test$locations)
)

# Incorrect way

bold.data.taxo.geo.df.col <- bold.full.search(
  taxonomy = list(df_test$taxon_name),
  geography = list(df_test$locations)
)

## End(Not run)
```

`bold.public.search` *Search publicly available data on the BOLD database*

Description

Retrieves record ids for publicly available data based on taxonomy, geography, institutes, bin_uris or datasets/project codes search.

Usage

```
bold.public.search(
  taxonomy = NULL,
  geography = NULL,
  bins = NULL,
  institutes = NULL,
```

```
  dataset_codes = NULL,
  project_codes = NULL
)
```

Arguments

taxonomy	A list of single or multiple characters specifying the taxonomic names at any hierarchical level. Default value is NULL.
geography	A list of single or multiple characters specifying any of the country/province/state/region/sector/site names/codes. Default value is NULL.
bins	A list of single or multiple characters specifying the BIN ids. Default value is NULL.
institutes	A list of single or multiple characters specifying the institutes. Default value is NULL.
dataset_codes	A list of single or multiple characters specifying the dataset codes. Default value is NULL.
project_codes	A list of single or multiple characters specifying the project codes. Default value is NULL.

Details

`bold.public.search` searches publicly available data on BOLD, retrieving associated proccessids and marker codes. All the BCDM data can then be retrieved using the processids as inputs for the `bold.fetch` function. Search parameters can include one or a combination of taxonomy, geography, bin uris, dataset or project codes. Each input should be provided as a separate list (Ex. taxonomy = list("Panthera", "Poecilia"), geography = list("India")). A dataframe column can also be used as an input using the '$' operator (e.g., df$column_name). If this is the case (i.e. df$column_name), `as.list` should be used instead of just `list` (Ex. taxonomy = as.list (df$column_name), geography = as.list(df$column_name)). The character length of a search query should also be considered as the function wont be able to retrieve records if that exceeds the predetermined web URL character length (2048 characters). For multi-parameter searches (e.g. taxonomy + geography + bins; see the example: Taxonomy + Geography + BIN id), it's important to logically combine the parameters to ensure accurate and non-empty results. Misspelled queries or those for which no public data exists on BOLD at the time the function is executed will result in an error. This applies for any of the search parameters. There is a hard limit of 1 million record downloads for each search. Download speeds for very large requests for `bin_uris`, `dataset_codes` and `project_codes` will be throttled, resulting in more time for fetching the data. Download speed would also depend on the user's internet connection and computer specifications.

Value

A data frame containing all the processids and marker codes related to the query search.

Examples

```
## Not run:

# Taxonomy
```

```
bold.data <- bold.public.search(taxonomy = list("Panthera leo"))

# Result
head(bold.data, 10)

# Taxonomy and Geography
bold.data.taxo.geo <- bold.public.search(
  taxonomy = list("Panthera uncia"),
  geography = list("India")
)

# Result
head(bold.data.taxo.geo, 10)

# Input as a dataframe column
df_test <- data.frame(
  taxon_name = c("Panthera uncia"),
  locations = c("India", "Sri Lanka")
)

# Result
bold.data.taxo.geo.df.col <- bold.public.search(
  taxonomy = as.list(df_test$taxon_name),
  geography = as.list(df_test$locations)
)

## End(Not run)
```

`test.data` *Canadian spider data by Blagoev et al.(2015)*

Description

The test data comprises 1,336 process and sample IDs from the Salticidae (Arthropoda:Arachnida:Araneae) family, sourced from Canadian spider data published by Blagoev et al. (2015). This publication includes a DNA barcode reference library encompassing 1,018 species of Canadian spiders.

Usage

```
test.data
```

Format

A data frame with 1336 rows and 2 columns:

processid Character vector of processids

sampleid Character vector of sampleids corresponding to the processids

Source

https://onlinelibrary.wiley.com/doi/full/10.1111/1755-0998.12444

References

Blagoev, G. A., Dewaard, J. R., Ratnasingham, S., Dewaard, S. L., Lu, L., Robertson, J., ... & Hebert, P. D. (2016). Untangling taxonomy: a DNA barcode reference library for C anadian spiders. Molecular Ecology Resources, 16(1), 325-341.

`test.data2` *Example Barcode Core Data Model (BCDM) dataset*

Description

The `test.data2` object is an example of a BCDM output obtained using the `bold.fetch` function. It contains public BOLD records for the freshwater crustacean genus *Streptocephalus*, consisting of 78 records. The dataset was obtained from the BOLD Public Data Package released on June 30, 2026.

Usage

```
test.data2
```

Format

A data frame with 78 rows and 72 columns.

The returned object follows the Barcode Core Data Model (BCDM) standard and contains specimen, taxonomic, geographic, collection, and sequence metadata. Fields correspond to the official BCDM field definitions (https://github.com/boldsystems-central/BCDM)

Source

doi:10.5883/DPBOLD_Public.30Jun2026

Index

∗ **datasets**
test.data, 27
test.data2, 28

ape::dist.dna(), *11*
ape::nj(), *11*
ape::njs(), *11*
ape::plot.phylo(), *11*

bold.analyze.align, 2
bold.analyze.align(), *10*
bold.analyze.diversity, 4
bold.analyze.map, 8
bold.analyze.tree, 10
bold.apikey, 13
bold.data.summarize, 14
bold.export, 16
bold.fetch, 19
bold.fetch(), *2–4*, *9*, *13*, *14*, *17*
bold.fields.info, 22
bold.full.search, 23
bold.full.search(), *13*
bold.public.search, 25

test.data, 27
test.data2, 28