Package ‘taxize’

May 8, 2026

Title Taxonomic Information from Around the Web

Description Interacts with a suite of web application programming interfaces (API) for taxonomic tasks, such as getting database specific taxonomic identifiers, verifying species names, getting taxonomic hierarchies, fetching downstream and upstream taxonomic names, getting taxonomic synonyms, converting scientific to common names and vice versa, and more. Some of the services supported include 'NCBI E-utilities' (<`https://www.ncbi.nlm.nih.gov/books/NBK25501/`>), 'Encyclopedia of Life' (<`https://eol.org/docs/what-is-eol/data-services`>), 'Global Biodiversity Information Facility' (<`https://techdocs.gbif.org/en/openapi/`>), and many more. Links to the API documentation for other supported services are available in the documentation for their respective functions in this package.

Version 0.10.1

License MIT + file LICENSE

URL `https://docs.ropensci.org/taxize/` (website), `https://github.com/ropensci/taxize` (devel)

BugReports `https://github.com/ropensci/taxize/issues`

LazyLoad yes

LazyData true

Encoding UTF-8

Language en-US

Depends R(>= 3.2.1)

Imports graphics, methods, stats, utils, crul (>= 0.7.0), xml2 (>= 1.2.0), jsonlite, ape, zoo, data.table, tibble (>= 1.2), rredlist, rotl (>= 3.0.0), ritis (>= 0.7.6), worrms (>= 0.4.0), natserv (>= 1.0.0), wikitaxa (>= 0.3.0), R6, crayon, cli, phangorn, lifecycle, curl, stringi

Suggests testthat, vegan, vcr

RoxygenNote 7.3.3

X-schema.org-applicationCategory Taxonomy

X-schema.org-keywords taxonomy, biology, nomenclature, JSON, API, web, api-client, identifiers, species, names

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut],
Eduard Szoecs [aut],
Zachary Foster [aut, cre],
Zebulun Arendsee [aut],
Carl Boettiger [ctb],
Karthik Ram [ctb],
Ignasi Bartomeus [ctb],
John Baumgartner [ctb],
James O'Donnell [ctb],
Jari Oksanen [ctb],
Bastian Greshake Tzovaras [ctb],
Philippe Marchand [ctb],
Vinh Tran [ctb],
Maëlle Salmon [ctb],
Gaopeng Li [ctb],
Matthias Grenié [ctb],
rOpenSci [fnd]

Maintainer Zachary Foster `<zacharyfoster1989@gmail.com>`

Repository CRAN

Date/Publication 2026-02-14 14:40:02 UTC

Contents

`taxize-package` *Taxonomic Information from Around the Web*

Description

This package interacts with a suite of web 'APIs' for taxonomic tasks, such as verifying species names, getting taxonomic hierarchies, and verifying name spelling.

About

Allows users to search over many websites for species names (scientific and common) and download up- and downstream taxonomic hierarchical information - and many other things.

The functions in the package that hit a specific API have a prefix and suffix separated by an underscore. They follow the format of `service_whatitdoes`. For example, `gnr_resolve` uses the Global Names Resolver API to resolve species names.

General functions in the package that don't hit a specific API don't have two words separated by an underscore, e.g., `classification`

You need API keys for some data sources. See taxize-authentication for more information.

Currently supported APIs

API	prefix	SOAP?
Encyclopedia of Life (EOL)	eol	FALSE
Integrated Taxonomic Information Service (ITIS)	itis	FALSE
Global Names Resolver (from EOL/GBIF)	gnr	FALSE
Global Names Index (from EOL/GBIF)	gni	FALSE
IUCN Red List	iucn	FALSE
Tropicos (from Missouri Botanical Garden)	tp	FALSE
Theplantlist.org	tpl	FALSE
National Center for Biotechnology Information	ncbi	FALSE
CANADENSYS Vascan name search API	vascan	FALSE
International Plant Names Index (IPNI)	ipni	FALSE
World Register of Marine Species (WoRMS)	worms	TRUE
Barcode of Life Data Systems (BOLD)	bold	FALSE
Pan-European Species directories Infrastructure (PESI)	pesi	TRUE
Mycobank	myco	TRUE
National Biodiversity Network (UK)	nbn	FALSE
Index Fungorum	fg	FALSE
EU BON	eubon	FALSE
Index of Names (ION)	ion	FALSE
Open Tree of Life (TOL)	tol	FALSE
World Register of Marine Species (WoRMS)	worms	FALSE
NatureServe	natserv	FALSE

If the source above has a `TRUE` in the `SOAP?` column, it is not available in this package. They are available from a different package called **taxizesoap**. See the GitHub repo for how to install https://github.com/ropensci/taxizesoap

Catalogue of Life (COL)

COL introduced rate limiting recently in 2019 - which has made the API essentially unusable - CoL+ is coming soon and we'll incorporate it here when it's stable. See https://github.com/ropensci/colpluz for the R implementation for CoL+

Author(s)

Scott Chamberlain

Eduard Szoecs <`eduardszoecs@gmail.com`>

Zachary Foster <`zacharyfoster1989@gmail.com`>

Carl Boettiger <`cboettig@gmail.com`>

Karthik Ram <`karthik@ropensci.org`>

Ignasi Bartomeus <`nacho.bartomeus@gmail.com`>

John Baumgartner <`johnbb@student.unimelb.edu.au`>

James O'Donnell <`jodonnellbio@gmail.com`>

`apg` *Get APG names*

Description

Generic names and their replacements from the Angiosperm Phylogeny Group III system of flowering plant classification.

Usage

```
apgOrders(...)

apgFamilies(...)
```

Arguments

`...`	Curl args passed on to crul::verb-GET

References

http://www.mobot.org/MOBOT/research/APweb/

Examples

```
## Not run:
head(apgOrders())
head(apgFamilies())

## End(Not run)
```

`apg_families` *MOBOT family names*

Description

Family names and their replacements from the Angiosperm Phylogeny Website system of flowering plant classification.

Format

A data frame with 1705 rows and 6 variables:

- `family`: family name
- `synonym`: if `accepted=FALSE`, this is the accepted name; if `accepted=TRUE`, this is `NA`, and the name in `family` is accepted
- `order`: order name for the family
- `accepted`: logical, if name in `family` column is accepted or not
- `original`: original data record from APG website, mapping name in `family` column to a new name, if there is one
- `accepted_name`: accepted name. accepted names, combining those that are accepted from `family` column, with the new name from `synonym` if applicable

Details

This dataset is from Version 14, incorporated on 2020-06-03, generated using `apgFamilies()`

(update script in inst/ignore/apg_script.R)

Source

http://www.mobot.org/MOBOT/research/APweb/

`apg_lookup` *Lookup in the APGIII taxonomy and replace family names*

Description

Lookup in the APGIII taxonomy and replace family names

Usage

```
apg_lookup(taxa, rank = "family")
```

Arguments

`taxa`	(character) Taxonomic name to lookup a synonym for in APGIII taxonomy.
`rank`	(character) Taxonomic rank to lookup a synonym for. One of family or order.

Details

Internally in this function, we use the datasets apg_families and apg_orders - see their descriptions for the data in them. The functions `apgOrders()` `apgFamilies()` are for scraping current content from the http://www.mobot.org/MOBOT/research/APweb/ website

The datasets used in this function are from the most recent version of APGIII, Version 14 (http://www.mobot.org/MOBOT/rese

Value

A APGIII family or order name, the original name if the name is the same as APG has, or NA if no match found

Examples

```
# New name found
apg_lookup(taxa = "Hyacinthaceae", rank = "family")
# Name is the same
apg_lookup(taxa = "Poaceae", rank = "family")
apg_lookup(taxa = "Asteraceae", rank = "family")
# Name not found
apg_lookup(taxa = "Foobar", rank = "family")

# New name found
apg_lookup(taxa = "Acerales", rank = "order")
# Name is the same
apg_lookup(taxa = "Acorales", rank = "order")
# Name not found
apg_lookup(taxa = "Foobar", rank = "order")
```

`apg_orders` *MOBOT order names*

Description

Order names and their replacements from the Angiosperm Phylogeny Website system of flowering plant classification.

Format

A data frame with 576 rows and 5 variables:

- `order`: order name
- `synonym`: if `accepted=FALSE`, this is the accepted name; if `accepted=TRUE`, this is `NA`, and the name in `order` is accepted
- `accepted`: logical, if name in `order` column is accepted or not
- `original`: original data record from APG website, mapping name in `order` column to a new name, if there is one
- `accepted_name`: accepted name. accepted names, combining those that are accepted from `order` column, with the new name from `synonym` if applicable

Details

This dataset is from Version 14, incorporated on 2020-06-03, generated using `apgOrders()`

(update script in inst/ignore/apg_script.R)

Source

http://www.mobot.org/MOBOT/research/APweb/

`bold_downstream` *Retrieve all taxa names downstream in hierarchy for BOLD*

Description

Retrieve all taxa names downstream in hierarchy for BOLD

Usage

```
bold_downstream(id, downto, intermediate = FALSE, ...)
```

Arguments

`id`	(integer) One or more BOLD taxonomic identifiers
`downto`	(character) The taxonomic level you want to go down to. See examples below. The taxonomic level IS case sensitive, and you do have to spell it correctly. See `data(rank_ref)` for spelling.
`intermediate`	(logical) If `TRUE`, return a list of length two with target taxon rank names, with additional list of data.frame's of intermediate taxonomic groups. Default: `FALSE`
`...`	crul options passed on to crul::verb-GET

Details

BEWARE: This function scrapes the BOLD website, so may be unstable. That is, one day it may work, and the next it may fail. Open an issue if you encounter an error: https://github.com/ropensci/taxize/issues

Value

data.frame of taxonomic information downstream to family from e.g., Order, Class, etc., or if `intermediated=TRUE`, list of length two, with target taxon rank names, and intermediate names.

Examples

```
## Not run:
## the genus Gadus
bold_downstream(id = 3451, downto="species")

bold_downstream(id = 443, downto="genus")
bold_downstream(id = 443, downto="genus", intermediate=TRUE)

## End(Not run)
```

`bold_search` *Search Barcode of Life for taxonomic IDs*

Description

Search Barcode of Life for taxonomic IDs

Usage

```
bold_search(
  sci = NULL,
  id = NULL,
  fuzzy = FALSE,
  dataTypes = "basic",
  includeTree = FALSE,
  response = FALSE,
  name = NULL,
  ...
)
```

Arguments

`sci`	(character) One or more scientific names.
`id`	(integer) One or more BOLD taxonomic identifiers.
`fuzzy`	(logical) Whether to use fuzzy search or not (default: `FALSE`). Only used if `name` passed.
`dataTypes`	(character) Specifies the datatypes that will be returned. See Details for options. This variable is ignored if `name` parameter is passed, but is used if the `id` parameter is passed.
`includeTree`	(logical) If TRUE (default: FALSE), returns a list containing information for parent taxa as well as the specified taxon. Only used if `id` passed.
`response`	(logical) Note that response is the object that returns from the curl call, useful for debugging, and getting detailed info on the API call.
`name`	Deprecated, see `sci`
`...`	named curl options passed on to crul::verb-GET

Details

You must provide one of `sci` or `id` to this function. The other parameters are optional. Note that when passing in `sci`, `fuzzy` can be used as well, while if `id` is passed, then `fuzzy` is ignored, and `dataTypes` `includeTree` can be used.

Options for `dataTypes` parameter:

- all returns all data
- basic returns basic taxon information
- images returns specimen image. Includes copyright information, image URL, image metadata.
- stats Returns specimen and sequence statistics. Includes public species count, public BIN count, public marker counts, public record count, specimen count, sequenced specimen count, barcode specimen count, species count, barcode species count.
- geo Returns collection site information. Includes country, collection site map.
- sequencinglabs Returns sequencing labs. Includes lab name, record count.
- depository Returns specimen depositories. Includes depository name, record count.
- thirdparty Returns information from third parties. Includes wikipedia summary, wikipedia URL, GBIF map.

Value

A list of data.frame's.

References

http://www.boldsystems.org/index.php/resources/api

Examples

```
## Not run:
# A basic example
bold_search(sci="Apis")
bold_search(sci="Agapostemon")
bold_search(sci="Poa")

# Fuzzy search
head(bold_search(sci="Po", fuzzy=TRUE))
head(bold_search(sci="Aga", fuzzy=TRUE))

# Many names
bold_search(sci=c("Apis","Puma concolor"))
nms <- names_list('species')
bold_search(sci=nms)

# Searching by ID - dataTypes can be used, and includeTree can be used
bold_search(id=88899)
bold_search(id=88899, dataTypes="stats")
bold_search(id=88899, dataTypes="geo")
bold_search(id=88899, dataTypes="basic")
bold_search(id=88899, includeTree=TRUE)

## End(Not run)
```

`children` *Retrieve immediate children taxa for a given taxon name or ID.*

Description

This function is different from `downstream()` in that it only collects immediate taxonomic children, while `downstream()` collects taxonomic names down to a specified taxonomic rank, e.g., getting all species in a family.

Usage

```
children(...)

## Default S3 method:
children(sci_id, db = NULL, rows = NA, x = NULL, ...)

## S3 method for class 'tsn'
children(sci_id, db = NULL, ...)

## S3 method for class 'wormsid'
children(sci_id, db = NULL, ...)

## S3 method for class 'ids'
```

```
children(sci_id, db = NULL, ...)

## S3 method for class 'uid'
children(sci_id, db = NULL, ...)

## S3 method for class 'boldid'
children(sci_id, db = NULL, ...)
```

Arguments

`...`	Further args passed on to `ritis::hierarchy_down()`, `ncbi_children()`, `worrms::wm_children()`, `bold_children()` See those functions for what parameters can be passed on.
`sci_id`	Vector of taxa names (character) or IDs (character or numeric) to query.
`db`	character; database to query. One or more of `itis`, `ncbi`, `worms`, or `bold`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong db value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using ncbi, we recommend getting an API key; see taxize-authentication
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are considered. Note that this parameter is ignored if you pass in a taxonomic id of any of the acceptable classes: tsn. NCBI has a method for this function but rows doesn't work.
`x`	Deprecated, see `sci_id`

Value

A named list of data.frames with the children names of every supplied taxa. You get an NA if there was no match in the database.

ncbi

note that with `db = "ncbi"`, we set `ambiguous = TRUE`; that is, children taxa with words like "unclassified", "unknown", "uncultured", "sp." are NOT removed

bold

BEWARE: `db="bold"` scrapes the BOLD website, so may be unstable. That is, one day it may work, and the next it may fail. Open an issue if you encounter an error: https://github.com/ropensci/taxize/issues

Examples

```
## Not run:
# Plug in taxonomic IDs
children(161994, db = "itis")
children(8028, db = "ncbi")
## works with numeric if as character as well
children(161994, db = "itis")
children(88899, db = "bold")
children(as.boldid(88899))
```

```
# Plug in taxon names
children("Salmo", db = 'itis')
children("Salmo", db = 'ncbi')
children("Salmo", db = 'worms')
children("Salmo", db = 'bold')

# Plug in IDs
(id <- get_wormsid("Gadus"))
children(id)

# Many taxa
sp <- c("Tragia", "Schistocarpha", "Encalypta")
children(sp, db = 'itis')

# Two data sources
(ids <- get_ids("Apis", db = c('ncbi','itis')))
children(ids)
## same result
children(get_ids("Apis", db = c('ncbi','itis')))

# Use the rows parameter
children("Poa", db = 'itis')
children("Poa", db = 'itis', rows=1)

# use curl options
res <- children("Poa", db = 'itis', rows=1, verbose = TRUE)

## End(Not run)
```

`class2tree` *Convert a list of classifications to a tree.*

Description

This function converts a list of hierarchies for individual species into a single species by taxonomic level matrix, then calculates a distance matrix based on taxonomy alone, and outputs either a phylo or dist object. See details for more information.

Usage

```
class2tree(input, varstep = TRUE, check = TRUE, remove_shared = FALSE, ...)

## S3 method for class 'classtree'
plot(x, ...)

## S3 method for class 'classtree'
print(x, ...)
```

Arguments

`input`	List of classification data.frame's from the function `classification()`
`varstep`	Vary step lengths between successive levels relative to proportional loss of the number of distinct classes.
`check`	If TRUE, remove all redundant levels which are different for all rows or constant for all rows and regard each row as a different basal taxon (species). If FALSE all levels are retained and basal taxa (species) also must be coded as variables (columns). You will get a warning if species are not coded, but you can ignore this if that was your intention.
`remove_shared`	If `TRUE`, remove any taxa that are coarser ranks present in other taxa, such as both a genus and a species in that genus in the same tree.
`...`	Further arguments passed on to hclust.
`x`	Input object to print or plot - output from class2tree function.

Details

See `vegan::taxa2dist()`. Thanks to Jari Oksanen for making the taxa2dist function and pointing it out, and Clarke & Warwick (1998, 2001), which taxa2dist was based on. The taxonomy tree created is not only based on the clustering of the taxonomy ranks (e.g. strain, species, genus, ...), but it also utilizes the actual taxon clades (e.g. mammals, plants or reptiles, etc.). The process of this function is as following: First, all possible taxonomy ranks and their corresponding IDs for each given taxon will be collected from the input. Then, the rank vectors of all taxa will be aligned, so that they together will become a matrix where columns are ordered taxonomy ranks of all taxa and rows are the rank vectors of those taxa. After that, the rank matrix will be converted into taxonomy ID matrix, any missing rank will have a pseudo ID from the previous rank. Finally, this taxonomy ID matrix will be used to cluster taxa that have similar taxonomy hierarchy together.

Value

An object of class "classtree" with slots:

- phylo - The resulting object, a phylo object
- classification - The classification data.frame, with taxa as rows, and different classification levels as columns
- distmat - Distance matrix
- names - The names of the tips of the phylogeny

Note that when you execute the resulting object, you only get the phylo object. You can get to the other 3 slots by calling them directly, like output$names, etc.

Examples

```
## Not run:
spnames <- c('Quercus robur', 'Iris oratoria', 'Arachis paraguariensis',
 'Helianthus annuus','Madia elegans','Lupinus albicaulis',
 'Pinus lambertiana')
out <- classification(spnames, db='itis')
```

```
tr <- class2tree(out)
plot(tr)

spnames <- c('Klattia flava', 'Trollius sibiricus',
 'Arachis paraguariensis',
 'Tanacetum boreale', 'Gentiana yakushimensis','Sesamum schinzianum',
 'Pilea verrucosa','Tibouchina striphnocalyx','Lycium dasystemum',
 'Berkheya echinacea','Androcymbium villosum',
 'Helianthus annuus','Madia elegans','Lupinus albicaulis',
 'Pinus lambertiana')
out <- classification(spnames, db='ncbi')
tr <- class2tree(out)
plot(tr)

## End(Not run)
```

`classification` *Retrieve the taxonomic hierarchy for a given taxon ID.*

Description

Retrieve the taxonomic hierarchy for a given taxon ID.

Usage

```
classification(...)

## Default S3 method:
classification(
  sci_id,
  db = NULL,
  callopts = list(),
  return_id = TRUE,
  rows = NA,
  x = NULL,
  ...
)

## S3 method for class 'tsn'
classification(id, return_id = TRUE, ...)

## S3 method for class 'uid'
classification(
  id,
  callopts = list(),
  return_id = TRUE,
  batch_size = 50,
```

```
  max_tries = 3,
  ...
)

## S3 method for class 'eolid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'tpsid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'gbifid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'nbnid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'tolid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'wormsid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'natservid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'boldid'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'wiki'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'pow'
classification(id, callopts = list(), return_id = TRUE, ...)

## S3 method for class 'ids'
classification(id, ...)

## S3 method for class 'classification'
cbind(...)

## S3 method for class 'classification'
rbind(...)

## S3 method for class 'classification_ids'
cbind(...)

## S3 method for class 'classification_ids'
rbind(...)
```

Arguments

`...`	For `classification`: other arguments passed to `get_tsn()`, `get_uid()`, `get_eolid()`, `get_tpsid()`, `get_gbifid()`, `get_wormsid()`, `get_natservid()`, `get_wormsid()`, `get_wiki()`, `get_pow()`. For `rbind.classification` and `cbind.classification`: one or more objects of class `classification`
`sci_id`	Vector of taxa names (character) or IDs (character or numeric) to query. For `db = "eol"`, EOL expects you to pass it a taxon id, called `eolid` in the output of `get_eolid()`.
`db`	character; database to query. either `ncbi`, `itis`, `eol`, `tropicos`, `gbif`, `nbn`, `worms`, `natserv`, `bold`, `wiki`, or `pow`. Note that each taxonomic data source has, their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using ncbi, and/or tropicos, we recommend getting an API key; see taxize-authentication
`callopts`	Curl options passed on to crul::verb-GET
`return_id`	(logical) If `TRUE` (default), return the taxon id as well as the name and rank of taxa in the lineage returned. Ignored for natserv as they don't return IDs in their taxonomic classification data.
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are considered. Note that this parameter is ignored if you pass in a taxonomic id instead of a name of class character.
`x`	Deprecated, see `sci_id`
`id`	character; identifiers, returned by `get_tsn()`, `get_uid()`, `get_eolid()`, `get_tpsid()`, `get_gbifid()`, `get_tolid()`, `get_wormsid()`, `get_natservid()`, `get_wormsid()`, `get_wiki()`, `get_pow()`
`batch_size`	(numeric) For NCBI queries, specify the number of IDs to lookup for each query.
`max_tries`	(numeric) For NCBI queries, the number of times a particular query will be attempted, assuming the first does not work.

Details

If IDs are supplied directly (not from the `get_*` functions) you must specify the type of ID. There is a timeout of 1/3 seconds between queries to NCBI.

BEWARE: Right now, NBN doesn't return the queried taxon in the classification. But you can attach it yourself quite easily of course. This behavior is different from the other data sources.

Value

A named list of data.frames with the taxonomic classification of every supplied taxa.

Lots of results

It may happen sometimes that you get more results back from your query than will show in the data.frame on screen. Our advice is to refine your query in those cases. On a data source basis we can attempt to help make it easier to refine queries, whether it be with the data provider (unlikely to happen), or in the code in this package (more likely) - let us know if you run into too many results problem and we'll see what we can do.

Authentication

See taxize-authentication

EOL

EOL does not have very good failure behavior. For example, if you submit an ID that does not exist they'll return a 500 HTTP error, which is not an appropriate error; it's probably that that ID does not exist in their database, but we can't know for sure. Isn't that fun?

NCBI Rate limits

In case you run into NCBI errors due to your rate limit being exceeded, see `taxize_options()`, where you can set `ncbi_sleep`.

HTTP version for NCBI requests

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

See Also

`get_tsn()`, `get_uid()`, `get_eolid()`, `get_tpsid()`, `get_gbifid()`, `get_wormsid()`, `get_natservid()`, `get_boldid()`, `get_wiki()`, `get_pow()`

Examples

```
## Not run:
# Plug in taxon IDs
classification(9606, db = 'ncbi')
classification(c(9606, 55062), db = 'ncbi')
classification(129313, db = 'itis')
classification(6985636, db = 'eol')
classification(126436, db = 'worms')
classification('Helianthus annuus', db = 'pow')
classification('Helianthus', db = 'pow')
classification('Asteraceae', db = 'pow')
classification("134717", db = 'natserv')
classification(c(2704179, 6162875, 8286319, 2441175, 731), db = 'gbif')
classification(25509881, db = 'tropicos')
classification("NBNSYS0000004786", db = 'nbn')
classification(as.nbnid("NBNSYS0000004786"), db = 'nbn')
classification(3930798, db = 'tol')

## works the same if IDs are in class character
classification(c("2704179", "2441176"), db = 'gbif')
classification("Agapostemon", db = "bold")

# wikispecies
classification("Malus domestica", db = "wiki")
classification("Pinus contorta", db = "wiki")
classification("Pinus contorta", db = "wiki", wiki_site = "commons")
```

```
classification("Pinus contorta", db = "wiki", wiki_site = "pedia")
classification("Pinus contorta", db = "wiki", wiki_site = "pedia",
  wiki = "fr")

classification(get_wiki("Malus domestica", "commons"))
classification(get_wiki("Malus domestica", "species"))
classification(c("Pinus contorta", "Malus domestica"), db = "wiki")

# Plug in taxon names
## in this case, we use get_*() fxns internally to first get taxon IDs
classification("Oncorhynchus mykiss", db = "eol")
classification(c("Chironomus riparius", "aaa vva"), db = 'ncbi')
classification(c("Chironomus riparius", "aaa vva"), db = 'ncbi',
  messages=FALSE)
classification(c("Chironomus riparius", "aaa vva"), db = 'itis')
classification(c("Chironomus riparius", "aaa vva"), db = 'itis',
  messages=FALSE)
classification(c("Chironomus riparius", "aaa vva"), db = 'eol')
classification("Alopias vulpinus", db = 'nbn')
classification('Gadus morhua', db = 'worms')
classification('Aquila chrysaetos', db = 'natserv')
classification('Gadus morhua', db = 'natserv')
classification('Pomatomus saltatrix', db = 'natserv')
classification('Aquila chrysaetos', db = 'natserv')
classification(c("Chironomus riparius", "asdfasdfsfdfsd"), db = 'gbif')
classification("Chironomus", db = 'tol')
classification("Poa annua", db = 'tropicos')

# Use methods for get_uid, get_tsn, get_eolid, get_tpsid
classification(get_uid(c("Chironomus riparius", "Puma concolor")))

classification(get_uid(c("Chironomus riparius", "aaa vva")))
classification(get_tsn(c("Chironomus riparius", "aaa vva")))
classification(get_tsn(c("Chironomus riparius", "aaa vva"),
  messages = FALSE))
classification(get_eolid(c("Chironomus riparius", "aaa vva")))
classification(get_tpsid(c("Poa annua", "aaa vva")))
classification(get_gbifid(c("Poa annua", "Bison bison")))

# Pass many ids from class "ids"
(out <- get_ids("Puma concolor", db = c('ncbi','gbif')))
(cl <- classification(out))

# Bind width-wise from class classification_ids
cbind(cl)

# Bind length-wise
rbind(cl)

# Many names to get_ids
(out <- get_ids(c("Puma concolor","Accipiter striatus"),
  db = c('ncbi','itis')))
(cl <- classification(out))
```

```
rbind(cl)
## cbind with so many names results in some messy data
cbind(cl)
## so you can turn off return_id
cbind( classification(out, return_id=FALSE) )

(cl_uid <- classification(get_uid(c("Puma concolor",
  "Accipiter striatus")), return_id=FALSE))
rbind(cl_uid)
cbind(cl_uid)
## cbind works a bit odd when there are lots of ranks without names
(cl_uid <- classification(get_uid(c("Puma concolor","Accipiter striatus")),
  return_id=TRUE))
cbind(cl_uid)

(cl_tsn <- classification(get_tsn(c("Puma concolor","Accipiter striatus"))))
rbind(cl_tsn)
cbind(cl_tsn)

(tsns <- get_tsn(c("Puma concolor","Accipiter striatus")))
(cl_tsns <- classification(tsns))
cbind(cl_tsns)

# NBN data
(res <- classification(c("Alopias vulpinus","Pinus sylvestris"),
  db = 'nbn'))
rbind(res)
cbind(res)

# Return taxonomic IDs
## the return_id parameter is logical, and you can turn it on or off.
## It's TRUE by default
classification(c("Alopias vulpinus","Pinus sylvestris"), db = 'ncbi',
  return_id = TRUE)
classification(c("Alopias vulpinus","Pinus sylvestris"), db = 'ncbi',
  return_id = FALSE)

# Use rows parameter to select certain
classification('Poa annua', db = 'tropicos')
classification('Poa annua', db = 'tropicos', rows=1:4)
classification('Poa annua', db = 'tropicos', rows=1)
classification('Poa annua', db = 'tropicos', rows=6)

# Queries of many IDs are processed in batches for NCBI
ids <- c("13083", "2650392", "1547764", "230054", "353934", "656984",
"271789", "126272", "184644", "73213", "662816", "1161803", "1239353",
"59420", "665675", "866969", "1091219", "1431218", "1471898",
"864321", "251768", "2486276", "2068772", "1825808", "2006532",
"128287", "1195738", "1084683", "1886461", "508296", "377247",
"1489665", "329325", "219243", "1176946", "339893", "197933",
"174510", "1704048", "212897", "154842", "1239280", "260135",
"405735", "1566412", "2083462", "651348", "983204", "165380",
"2338856", "2068760", "167262", "34229", "1213340", "478939",
```

```
"1933585", "49951", "1277794", "1671089", "1502538", "362355",
"746473", "242879", "158219", "313664", "2093188", "1541232",
"584742", "1331091", "147639", "284492", "75642", "1412882",
"391782", "1406855", "434506", "2053357", "217315", "1444328",
"329249", "2294004", "84942", "324458", "538247", "69452", "49170",
"1993845", "261646", "127633", "228146", "1420004", "1629772",
"577055", "697062", "231660", "648380", "554953", "746496", "2602969")
result <- classification(ids, db = 'ncbi')

## End(Not run)
```

`comm2sci` *Get scientific names from common names.*

Description

Get scientific names from common names.

Usage

```
comm2sci(...)

## Default S3 method:
comm2sci(
  com,
  db = "ncbi",
  itisby = "search",
  simplify = TRUE,
  commnames = NULL,
  ...
)

## S3 method for class 'tsn'
comm2sci(id, db = "ncbi", itisby = "search", simplify = TRUE, ...)

## S3 method for class 'uid'
comm2sci(id, db = "ncbi", itisby = "search", simplify = TRUE, ...)
```

Arguments

`...`	Further arguments passed on to internal methods.
`com`	One or more common names or partial names.
`db`	Data source, one of *"ncbi"* (default), *"itis"*, *"tropicos"*, *"eol"*, or *"worms"*. If using ncbi, we recommend getting an API key; see taxize-authentication
`itisby`	Search for common names across entire names (search, default), at beginning of names (begin), or at end of names (end).

`simplify`	(logical) If `TRUE`, simplify output to a vector of names. If `FALSE`, return variable formats from different sources, usually a data.frame.
`commnames`	Deprecated, see `com`
`id`	taxon identifiers, as returned by `get_tsn()` or `get_uid()`

Details

For data sources ITIS and NCBI you can pass in common names directly, and use `get_uid()` or `get_tsn()` to get ids first, then pass in to this fxn.

For the other data sources, you can only pass in common names directly.

Value

If `simplify=TRUE`, a list of scientific names, with list labeled by your input names. If `simplify=FALSE`, a data.frame with columns that vary by data source. `character(0)` on no match

Authentication

See taxize-authentication for help on authentication

HTTP version for NCBI requests

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Rate limits

In case you run into errors due to your rate limit being exceeded, see `taxize_options()`, where you can set `ncbi_sleep`.

Author(s)

Scott Chamberlain

See Also

`sci2comm()`

Examples

```
## Not run:
comm2sci(com='american black bear')
comm2sci(com='american black bear', simplify = FALSE)
comm2sci(com='black bear', db='itis')
comm2sci(com='american black bear', db='itis')
comm2sci(com='annual blue grass', db='tropicos')
comm2sci(com=c('annual blue grass','tree of heaven'), db='tropicos')
comm2sci('blue whale', db = "worms")
comm2sci(c('blue whale', 'dwarf surfclam'), db = "worms")
```

```
# ncbi: pass in uid's from get_uid() directly
x <- get_uid("western capercaillie", modifier = "Common Name")
comm2sci(x)
# itis: pass in tsn's from get_tsn() directly
x <- get_tsn(c("Louisiana black bear", "american crow"),
  searchtype = "common")
comm2sci(x)

## End(Not run)
```

`downstream` *Retrieve the downstream taxa for a given taxon name or ID.*

Description

This function uses a while loop to continually collect children taxa down to the taxonomic rank that you specify in the `downto` parameter. You can get data from ITIS (itis), GBIF (gbif), NCBI (ncbi), WORMS (worms), or BOLD (bold). There is no method exposed by these four services for getting taxa at a specific taxonomic rank, so we do it ourselves here.

Usage

```
downstream(...)

## Default S3 method:
downstream(
  sci_id,
  db = NULL,
  downto = NULL,
  intermediate = FALSE,
  rows = NA,
  x = NULL,
  ...
)

## S3 method for class 'tsn'
downstream(sci_id, db = NULL, downto = NULL, intermediate = FALSE, ...)

## S3 method for class 'gbifid'
downstream(
  sci_id,
  db = NULL,
  downto = NULL,
  intermediate = FALSE,
  limit = 100,
  start = NULL,
  ...
)
```

```
## S3 method for class 'uid'
downstream(sci_id, db = NULL, downto = NULL, intermediate = FALSE, ...)

## S3 method for class 'wormsid'
downstream(sci_id, db = NULL, downto = NULL, intermediate = FALSE, ...)

## S3 method for class 'boldid'
downstream(sci_id, db = NULL, downto = NULL, intermediate = FALSE, ...)

## S3 method for class 'ids'
downstream(sci_id, db = NULL, downto = NULL, intermediate = FALSE, ...)
```

Arguments

`...`	Further args passed on to `itis_downstream()`, `gbif_downstream()`, `ncbi_downstream()`, `worms_downstream()`, or `bold_downstream()`
`sci_id`	Vector of taxa names (character) or IDs (character or numeric) to query.
`db`	character; database to query. One or more of `itis`, `gbif`, `ncbi`, `worms`, or `bold`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using ncbi, we recommend getting an API key; see taxize-authentication
`downto`	What taxonomic rank to go down to. One of: ’superkingdom’, ’kingdom’, ’subkingdom’,’infrakingdom’,’phylum’,’division’,’subphylum’, ’subdivision’,’infradivision’, ’superclass’,’class’,’subclass’,’infraclass’, ’superorder’,’order’,’suborder’,’infraorder’,’superfamily’,’famil ’subfamily’,’tribe’,’subtribe’,’genus’,’subgenus’,’section’,’subsection’, ’species group’,’species’,’subspecie ’stirp’, ’morph’,’aberration’,’subform’, ’unspecified’, ’no rank’
`intermediate`	(logical) If `TRUE`, return a list of length two with target taxon rank names, with additional list of data.frame’s of intermediate taxonomic groups. Default: `FALSE`
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are considered. Note that this parameter is ignored if you pass in a taxonomic id of any of the acceptable classes: tsn.
`x`	Deprecated, see `sci_id`
`limit`	Number of records to return. Applies to gbif only. default: 100. max: 1000. use in combination with the `start` parameter
`start`	Record number to start at. Applies to gbif only. default: 0. use in combination with the `limit` parameter

Value

A named list of data.frames with the downstream names of every supplied taxa. You get an NA if there was no match in the database.

Authentication

See taxize-authentication for help on authentication

bold

BEWARE: `db="bold"` scrapes the BOLD website, so may be unstable. That is, one day it may work, and the next it may fail. Open an issue if you encounter an error: https://github.com/ropensci/taxize/issues

Examples

```
## Not run:
# Plug in taxon IDs
downstream(125732, db = 'worms', downto = 'species')
downstream(3451, db = 'bold', downto = 'species')

if (interactive()) {

# Plug in taxon names
downstream("Apis", db = 'ncbi', downto = 'species')
downstream("Apis", db = 'itis', downto = 'species')
downstream("Apis", db = 'bold', downto = 'species')
downstream("Gadus", db = 'worms', downto = 'species')
downstream(c("Apis","Epeoloides"), db = 'itis', downto = 'species')
downstream("Ursus", db = 'gbif', downto = 'species')
downstream(get_gbifid("Ursus"), db = 'gbif', downto = 'species')

# Many taxa
sp <- names_list("genus", 3)
downstream(sp, db = 'itis', downto = 'species')
downstream(sp, db = 'gbif', downto = 'species')

# Both data sources
ids <- get_ids("Apis", db = c('gbif','itis'))
downstream(ids, downto = 'species')
## same result
downstream(get_ids("Apis", db = c('gbif','itis')), downto = 'species')

# Collect intermediate names
## itis
downstream('Bangiophyceae', db="itis", downto="genus")
downstream('Bangiophyceae', db="itis", downto="genus", intermediate=TRUE)
downstream(get_tsn('Bangiophyceae'), downto="genus")
downstream(get_tsn('Bangiophyceae'), downto="genus", intermediate=TRUE)

# Use the rows parameter
## note how in the second function call you don't get the prompt
downstream("Poa", db = 'gbif', downto="species")
downstream("Poa", db = 'gbif', downto="species", rows=1)

# use curl options
res <- downstream("Apis", db = 'gbif', downto = 'species', verbose = TRUE)

# Pagination
# GBIF limits queries to a maximum of 1000 records per request, so if
# there's more than 1000, use the start parameter
# Piper, taxonKey = 3075433
```

```
z1 <- downstream(3075433, db = 'gbif', downto = "species", limit=1000)
z2 <- downstream(3075433, db = 'gbif', downto = "species", limit=1000,
  start=1000)
z3 <- downstream(3075433, db = 'gbif', downto = "species", limit=1000,
  start=2000)
z4 <- downstream(3075433, db = 'gbif', downto = "species", limit=1000,
  start=3000)
NROW(rbind(z1[[1]], z2[[1]], z3[[1]], z4[[1]]))
}
## End(Not run)
```

`eol_dataobjects` *Given the identifier for a data object, return all metadata about the object*

Description

Given the identifier for a data object, return all metadata about the object

Usage

```
eol_dataobjects(id, taxonomy = TRUE, language = NULL, ...)
```

Arguments

`id`	(character) The EOL data object identifier
`taxonomy`	(logical) Whether to return any taxonomy details from different taxon hierarchy providers, in an array named `taxonconcepts`
`language`	(character) provides the results in the specified language. one of ms, de, en, es, fr, gl, it, nl, nb, oc, pt-BR, sv, tl, mk, sr, uk, ar, zh-Hans, zh-Hant, ko
`...`	Curl options passed on to crul::HttpClient

Details

It's possible to return JSON or XML with the EOL API. However, this function only returns JSON for now.

Value

A list, optionally with a data.frame if `taxonomy=TRUE`

Examples

```
## Not run:
eol_dataobjects(id = 7561533)

# curl options
eol_dataobjects(id = 7561533, verbose = TRUE)

## End(Not run)
```

`eol_pages` *Search for pages in EOL database using a taxonconceptID.*

Description

Search for pages in EOL database using a taxonconceptID.

Usage

```
eol_pages(
  taxonconceptID,
  images_per_page = NULL,
  images_page = NULL,
  videos_per_page = NULL,
  videos_page = NULL,
  sounds_per_page = NULL,
  sounds_page = NULL,
  maps_per_page = NULL,
  maps_page = NULL,
  texts_per_page = NULL,
  texts_page = NULL,
  subjects = "overview",
  licenses = "all",
  details = FALSE,
  common_names = FALSE,
  synonyms = FALSE,
  references = FALSE,
  taxonomy = TRUE,
  vetted = 0,
  cache_ttl = NULL,
  ...
)
```

Arguments

`taxonconceptID`	(numeric) a taxonconceptID, which is also the page number
`images_per_page`	(integer) number of returned image objects (0-75)
`images_page`	(integer) images page
`videos_per_page`	(integer) number of returned video objects (0-75)
`videos_page`	(integer) videos page
`sounds_per_page`	(integer) number of returned sound objects (0-75)
`sounds_page`	(integer) sounds page

`maps_per_page`	(integer) number of returned map objects (0-75)
`maps_page`	(integer) maps page
`texts_per_page`	(integer) number of returned text objects (0-75)
`texts_page`	(integer) texts page
`subjects`	'overview' (default) to return the overview text (if exists), a pipe | delimited list of subject names from the list of EOL accepted subjects (e.g. TaxonBiology, FossilHistory), or 'all' to get text in any subject. Always returns an overview text as a first result (if one exists in the given context).
`licenses`	A pipe | delimited list of licenses or 'all' (default) to get objects under any license. Licenses abbreviated cc- are all Creative Commons licenses. Visit their site for more information on the various licenses they offer.
`details`	Include all metadata for data objects. (Default: `FALSE`)
`common_names`	Return all common names for the page's taxon (Default: `FALSE`)
`synonyms`	Return all synonyms for the page's taxon (Default: `FALSE`)
`references`	Return all references for the page's taxon (Default: `FALSE`)
`taxonomy`	(logical) Whether to return any taxonomy details from different taxon hierarchy providers, in an array named `taxonconcepts` (Default: `TRUE`)
`vetted`	If 'vetted' is given a value of '1', then only trusted content will be returned. If 'vetted' is '2', then only trusted and unreviewed content will be returned (untrusted content will not be returned). The default is to return all content. (Default: `FALSE`)
`cache_ttl`	The number of seconds you wish to have the response cached.
`...`	Curl options passed on to crul::HttpClient

Details

It's possible to return JSON or XML with the EOL API. However, this function only returns JSON for now.

Value

JSON list object, or data.frame.

Examples

```
## Not run:
(pageid <- eol_search('Pomatomus')$pageid[1])
x <- eol_pages(taxonconceptID = pageid)
x
x$scinames

z <- eol_pages(taxonconceptID = pageid, synonyms = TRUE)
z$synonyms

z <- eol_pages(taxonconceptID = pageid, common_names = TRUE)
z$vernacular

## End(Not run)
```

`eol_search` *Search for terms in EOL database.*

Description

Search for terms in EOL database.

Usage

```
eol_search(
  sci,
  page = 1,
  exact = NULL,
  filter_tid = NULL,
  filter_heid = NULL,
  filter_by_string = NULL,
  cache_ttl = NULL,
  terms = NULL,
  ...
)
```

Arguments

`sci`	(character) scientific name
`page`	A maximum of 30 results are returned per page. This parameter allows you to fetch more pages of results if there are more than 30 matches (Default 1)
`exact`	Will find taxon pages if the preferred name or any synonym or common name exactly matches the search term.
`filter_tid`	Given an EOL page ID, search results will be limited to members of that taxonomic group
`filter_heid`	Given a Hierarchy Entry ID, search results will be limited to members of that taxonomic group
`filter_by_string`	Given a search term, an exact search will be made and that matching page will be used as the taxonomic group against which to filter search results
`cache_ttl`	The number of seconds you wish to have the response cached.
`terms`	Deprecated, see `sci`
`...`	Curl options passed on to crul::HttpClient

Details

It's possible to return JSON or XML with the EOL API. However, this function only returns JSON for now.

Value

A data frame with four columns:

- pageid: pageid, this is the same as the eolid you can get from `get_eolid()`
- name: taxonomic name, may or may not contain the taxonomic authority
- link: URL for the taxon in question
- content: a string of semi-colon separated names. it's not clear to us what these represent exactly, but figured why not give it to users in case some may find it useful

Examples

```
## Not run:
eol_search(sci='Homo')
eol_search(sci='Salix', verbose = TRUE)
eol_search(sci='Ursus americanus')
eol_search('Pinus contorta')

## End(Not run)
```

`eubon_capabilities` *EUBON capabilities*

Description

EUBON capabilities

Usage

```
eubon_capabilities(...)
```

Arguments

`...`	Curl options passed on to crul::verb-GET

References

https://cybertaxonomy.eu/eu-bon/utis/1.3/doc.html

See Also

Other eubon-methods: `eubon_children`(), `eubon_hierarchy`(), `eubon_search`()

Examples

```
## Not run:
eubon_capabilities()

## End(Not run)
```

`eubon_children` *EUBON children*

Description

EUBON children

Usage

```
eubon_children(id, providers = NULL, timeout = 0, ...)
```

Arguments

`id`	(character) identifier for the taxon. (LSID, DOI, URI, or any other identifier used by the checklist provider)
`providers`	(character) A list of provider id strings concatenated by comma characters. The default : "pesi,bgbm-cdm-server[col] will be used if this parameter is not set. A list of all available provider ids can be obtained from the '/capabilities' service end point. Providers can be nested, that is a parent provider can have sub providers. If the id of the parent provider is supplied all subproviders will be queried. The query can also be restricted to one or more subproviders by using the following syntax: parent-id[sub-id-1,sub-id2,...]
`timeout`	(numeric) The maximum of milliseconds to wait for responses from any of the providers. If the timeout is exceeded the service will just return the responses that have been received so far. The default timeout is 0 ms (wait for ever)
`...`	Curl options passed on to crul::verb-GET

Value

a data.frame or an empty list if no results found

Note

There is no pagination in this method, so you may or may not be getting all the results for a search. Sorry, out of our control

References

https://cybertaxonomy.eu/eu-bon/utis/1.3/doc.html

See Also

Other eubon-methods: `eubon_capabilities()`, `eubon_hierarchy()`, `eubon_search()`

Examples

```
## Not run:
x <- eubon_children(id = "urn:lsid:marinespecies.org:taxname:126141",
  providers = 'worms')
head(x)

## End(Not run)
```

`eubon_hierarchy` *EUBON hierarchy*

Description

EUBON hierarchy

Usage

```
eubon_hierarchy(id, providers = "pesi", timeout = 0, ...)
```

Arguments

`id`	(character) identifier for the taxon. (LSID, DOI, URI, or any other identifier used by the checklist provider)
`providers`	(character) A list of provider id strings concatenated by comma characters. The default : "pesi,bgbm-cdm-server[col]" will be used if this parameter is not set. A list of all available provider ids can be obtained from the '/capabilities' service end point. Providers can be nested, that is a parent provider can have sub providers. If the id of the parent provider is supplied all subproviders will be queried. The query can also be restricted to one or more subproviders by using the following syntax: parent-id[sub-id-1,sub-id2,...]
`timeout`	(numeric) The maximum of milliseconds to wait for responses from any of the providers. If the timeout is exceeded the service will just return the responses that have been received so far. The default timeout is 0 ms (wait for ever)
`...`	Curl options passed on to crul::verb-GET

Note

There is no pagination in this method, so you may or may not be getting all the results for a search. Sorry, out of our control

References

https://cybertaxonomy.eu/eu-bon/utis/1.3/doc.html

See Also

Other eubon-methods: `eubon_capabilities()`, `eubon_children()`, `eubon_search()`

Examples

```
## Not run:
eubon_hierarchy(id = "urn:lsid:marinespecies.org:taxname:126141", 'worms')
eubon_hierarchy(id = "urn:lsid:marinespecies.org:taxname:274350", 'worms')

## End(Not run)
```

`eubon_search` *EUBON taxonomy search*

Description

EUBON taxonomy search

Usage

```
eubon_search(
  query,
  providers = "pesi",
  searchMode = "scientificNameExact",
  addSynonymy = FALSE,
  addParentTaxon = FALSE,
  timeout = 0,
  dedup = NULL,
  limit = 20,
  page = 1,
  ...
)
```

Arguments

`query`	(character) The scientific name to search for. For example: "Bellis perennis", "Prionus" or "Bolinus brandaris". This is an exact search so wildcard characters are not supported
`providers`	(character) A list of provider id strings concatenated by comma characters. The default : "pesi,bgbm-cdm-server[col]" will be used if this parameter is not set. A list of all available provider ids can be obtained from the '/capabilities' service end point. Providers can be nested, that is a parent provider can have sub providers. If the id of the parent provider is supplied all subproviders will be queried. The query can also be restricted to one or more subproviders by using the following syntax: parent-id[sub-id-1,sub-id2,...]
`searchMode`	(character) Specifies the searchMode. Possible search modes are: `scientificNameExact`, `scientificNameLike` (begins with), `vernacularNameExact`, `vernacularNameLike` (contains), `findByIdentifier`. If the a provider does not support the chosen searchMode it will be skipped and the status message in the tnrClientStatus will be set to 'unsupported search mode' in this case.

`addSynonymy`	(logical) Indicates whether the synonymy of the accepted taxon should be included into the response. Turning this option on may cause an increased response time. Default: `FALSE`
`addParentTaxon`	(logical) Indicates whether the the parent taxon of the accepted taxon should be included into the response. Turning this option on may cause a slightly increased response time. Default: `FALSE`
`timeout`	(numeric) The maximum of milliseconds to wait for responses from any of the providers. If the timeout is exceeded the service will just return the responses that have been received so far. The default timeout is 0 ms (wait for ever)
`dedup`	(character) Allows to deduplicate the results by making use of a deduplication strategy. The deduplication is done by comparing specific properties of the taxon: • id: compares 'taxon.identifier' • id_name: compares 'taxon.identifier' AND 'taxon.taxonName.scientificName' • name: compares 'taxon.taxonName.scientificName' Using the pure 'name' strategy is not recommended.
`limit`	(numeric/integer) number of records to retrieve. default: 20. This only affects the search mode `scientificNameLike` and `vernacularNameLike`; other search modes are expected to return only one record per check list
`page`	(numeric/integer) page to retrieve. default: 1. This only affects the search mode `scientificNameLike` and `vernacularNameLike`; other search modes are expected to return only one record per check list
`...`	Curl options passed on to crul::verb-GET

References

https://cybertaxonomy.eu/eu-bon/utis/1.3/doc.html

See Also

Other eubon-methods: `eubon_capabilities()`, `eubon_children()`, `eubon_hierarchy()`

Examples

```
## Not run:
eubon_search("Prionus")
eubon_search("Salmo", "pesi")
eubon_search("Salmo", c("pesi", "worms"))
eubon_search("Salmo", "worms", "scientificNameLike")
eubon_search("Salmo", "worms", "scientificNameLike", limit = 3)
eubon_search("Salmo", "worms", "scientificNameLike", limit = 20, page = 2)
eubon_search("Salmo", "worms", addSynonymy = TRUE)
eubon_search("Salmo", "worms", addParentTaxon = TRUE)

## End(Not run)
```

`fungorum` *Index Fungorum*

Description

Search for taxonomic names in Index Fungorum

Usage

```
fg_name_search(q, anywhere = TRUE, limit = 10, ...)

fg_author_search(q, anywhere = TRUE, limit = 10, ...)

fg_epithet_search(q, anywhere = TRUE, limit = 10, ...)

fg_name_by_key(key, ...)

fg_name_full_by_lsid(lsid, ...)

fg_all_updated_names(date, ...)

fg_deprecated_names(date, ...)
```

Arguments

`q`	(character) Query term
`anywhere`	(logical) Default: `TRUE`
`limit`	(integer) Number of results to return. max limit value appears to be 6000, not positive about that though
`...`	Curl options passed on to crul::verb-GET
`key`	(character) A IndexFungorum taxon key
`lsid`	(character) an LSID, e.,g. "urn:lsid:indexfungorum.org:names:81085"
`date`	(character) Date, of the form YYYMMDD

Value

A `data.frame`, or `NULL` if no results

References

http://www.indexfungorum.org/, API docs: http://www.indexfungorum.org/ixfwebservice/fungus.asmx

Examples

```
## Not run:
# NameSearch
fg_name_search(q = "Gymnopus", limit = 2, verbose = TRUE)
fg_name_search(q = "Gymnopus")

# EpithetSearch
fg_epithet_search(q = "phalloides")

# NameByKey
fg_name_by_key(17703)

# NameFullByKey
fg_name_full_by_lsid("urn:lsid:indexfungorum.org:names:81085")

# AllUpdatedNames
fg_all_updated_names(date = gsub("-", "", Sys.Date() - 2))

# DeprecatedNames
fg_deprecated_names(date=20151001)

# AuthorSearch
fg_author_search(q = "Fayod", limit = 2)

## End(Not run)
```

`gbif_downstream` *Retrieve all taxonomic names downstream in hierarchy for GBIF*

Description

Retrieve all taxonomic names downstream in hierarchy for GBIF

Usage

```
gbif_downstream(
  id,
  downto,
  intermediate = FALSE,
  limit = 100,
  start = NULL,
  key = NULL,
  ...
)
```

Arguments

`id`	A taxonomic serial number.

`downto`	The taxonomic level you want to go down to. See examples below. The taxonomic level IS case sensitive, and you do have to spell it correctly. See `data(rank_ref)` for spelling.
`intermediate`	(logical) If TRUE, return a list of length two with target taxon rank names, with additional list of data.frame's of intermediate taxonomic groups. Default: FALSE
`limit`	Number of records to return. default: 100. max: 1000. use in combination with the `start` parameter
`start`	Record number to start at. default: 0. use in combination with the `limit` parameter
`key`	Deprecated, see `id`
`...`	Further args passed on to `gbif_name_usage()`

Details

Sometimes records don't have a `canonicalName` entry which is what we look for. In that case we grab the `scientificName` entry. You can see the type of name colleceted in the column `name_type`

Value

data.frame of taxonomic information downstream to family from e.g., Order, Class, etc., or if `intermediated=TRUE`, list of length two, with target taxon rank names, and intermediate names.

Author(s)

Scott Chamberlain

Examples

```
## Not run:
## the plant class Bangiophyceae
gbif_downstream(id = 198, downto="genus")
gbif_downstream(id = 198, downto="genus", intermediate=TRUE)

# families downstream from the family Strepsiptera (twisted wing parasites)
gbif_downstream(id = 1227, "family")
## here, intermediate leads to the same result as the target
gbif_downstream(id = 1227, "family", intermediate=TRUE)

if (interactive()) {
# Lepidoptera
gbif_downstream(id = 797, "family")

# get species downstream from the genus Ursus
gbif_downstream(id = 2433406, "species")

# get tribes down from the family Apidae
gbif_downstream(id = 7799978, downto="species")
gbif_downstream(id = 7799978, downto="species", intermediate=TRUE)
```

```
# names that don't have canonicalname entries for some results
# Myosotis: key 2925668
key <- 2925668
res <- gbif_downstream(key, downto = "species")
res2 <- downstream(key, db = "gbif", downto = "species")

# Pagination
# GBIF limits queries to a maximum of 1000 records per request, so if
# there's more than 1000, use the start parameter
# Piper, taxonKey = 3075433
x1 <- gbif_downstream(id = 3075433, downto = "species", limit=1000)
x2 <- gbif_downstream(id = 3075433, downto = "species", limit=1000,
  start=1000)
x3 <- gbif_downstream(id = 3075433, downto = "species", limit=1000,
  start=2000)
x4 <- gbif_downstream(id = 3075433, downto = "species", limit=1000,
  start=3000)
rbind(x1, x2, x3, x4)
}

## End(Not run)
```

`gbif_name_usage` *Lookup details for specific names in all taxonomies in GBIF.*

Description

This is a taxize version of the same function in the `rgbif` package so as to not have to import rgbif and thus require GDAL binary installation.

Usage

```
gbif_name_usage(
  key = NULL,
  name = NULL,
  data = "all",
  language = NULL,
  datasetKey = NULL,
  uuid = NULL,
  sourceId = NULL,
  rank = NULL,
  shortname = NULL,
  start = NULL,
  limit = 20,
  ...
)
```

Arguments

`key`	(numeric) A GBIF key for a taxon
`name`	(character) Filters by a case insensitive, canonical namestring, e.g. 'Puma concolor'
`data`	(character) Specify an option to select what data is returned. See Description below.
`language`	(character) Language, default is english
`datasetKey`	(character) Filters by the dataset's key (a uuid)
`uuid`	(character) A uuid for a dataset. Should give exact same results as datasetKey.
`sourceId`	(numeric) Filters by the source identifier. Not used right now.
`rank`	(character) Taxonomic rank. Filters by taxonomic rank as one of: CLASS, CULTIVAR, CULTIVAR_GROUP, DOMAIN, FAMILY, FORM, GENUS, INFORMAL, INFRAGENERIC_NAME, INFRAORDER, INFRASPECIFIC_NAME, INFRASUBSPECIFIC_NAME, KINGDOM, ORDER, PHYLUM, SECTION, SERIES, SPECIES, STRAIN, SUBCLASS, SUBFAMILY, SUBFORM, SUBGENUS, SUBKINGDOM, SUBORDER, SUBPHYLUM, SUBSECTION, SUBSERIES, SUBSPECIES, SUBTRIBE, SUBVARIETY, SUPERCLASS, SUPERFAMILY, SUPERORDER, SUPERPHYLUM, SUPRAGENERIC_NAME, TRIBE, UNRANKED, VARIETY
`shortname`	(character) A short name..need more info on this?
`start`	Record number to start at
`limit`	Number of records to return
`...`	Curl options passed on to crul::HttpClient

Value

A list of length two. The first element is metadata. The second is either a data.frame (verbose=FALSE, default) or a list (verbose=TRUE)

References

https://www.gbif.org/developer/summary

`gbif_parse` *Parse taxon names using the GBIF name parser.*

Description

Parse taxon names using the GBIF name parser.

Usage

```
gbif_parse(scientificname, ...)
```

Arguments

`scientificname`	(character) scientific names
`...`	Further args passed on to crul::verb-POST

Value

A `data.frame` containing fields extracted from parsed taxon names. Fields returned are the union of fields extracted from all species names in `scientificname`.

Author(s)

John Baumgartner `<johnbb@student.unimelb.edu.au>`

References

https://www.gbif.org/tools/name-parser/about

See Also

`gni_parse()`, `gna_parse()`

Examples

```
## Not run:
gbif_parse(scientificname='x Agropogon littoralis')
gbif_parse(c('Arrhenatherum elatius var. elatius',
             'Secale cereale subsp. cereale', 'Secale cereale ssp. cereale',
             'Vanessa atalanta (Linnaeus, 1758)'))

## End(Not run)
```

`genbank2uid` *Get NCBI taxonomy UID from GenBankID*

Description

Get NCBI taxonomy UID from GenBankID

Usage

```
genbank2uid(id, batch_size = 100, key = NULL, ...)
```

Arguments

`id`	A GenBank accession alphanumeric string, or a gi numeric string.
`batch_size`	The number of queries to submit at a time.
`key`	(character) NCBI Entrez API key. optional. See Details.
`...`	Curl args passed on to crul::HttpClient

Details

See https://www.ncbi.nlm.nih.gov/Sitemap/sequenceIDs.html for help on why there are two identifiers, and the difference between them.

Value

one or more NCBI taxonomic IDs

Authentication

See taxize-authentication for help on authentication. We recommend getting an API key.

HTTP version

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Rate limits

In case you run into errors due to your rate limit being exceeded, see `taxize_options()`, where you can set `ncbi_sleep`.

Examples

```
## Not run:
# with accession numbers
genbank2uid(id = 'AJ748748')
genbank2uid(id = 'Y13155')
genbank2uid(id = 'X78312')
genbank2uid(id = 'KM495596')

# with gi numbers
genbank2uid(id = 62689767)
genbank2uid(id = 22775511)
genbank2uid(id = 156446673)

# pass in many accession or gi numbers
genbank2uid(c(62689767,156446673))
genbank2uid(c('X78312','KM495596'))
genbank2uid(list('X78312',156446673))

# curl options
res <- genbank2uid(id = 156446673, verbose = TRUE)

## End(Not run)
```

getkey *Function to get API key.*

Description

Checks first to get key from your .Rprofile or .Renviron (or similar) file

Usage

```
getkey(x = NULL, service)
```

Arguments

x	(character) An API key, defaults to `NULL`
service	(character) The API data provider, used to match to default guest key (for Tropicos; there's no guest key for NCBI or IUCN, for which you have to get your own)

Examples

```
## Not run:
getkey(service="tropicos")
getkey(service="iucn")
getkey(service="entrez")

## End(Not run)
```

get_boldid *Get the BOLD (Barcode of Life) code for a search term.*

Description

Get the BOLD (Barcode of Life) code for a search term.

Usage

```
get_boldid(
  sci,
  fuzzy = FALSE,
  dataTypes = "basic",
  includeTree = FALSE,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  rank = NULL,
  division = NULL,
```

```
  parent = NULL,
  searchterm = NULL,
  ...
)

as.boldid(x, check = TRUE)

## S3 method for class 'boldid'
as.boldid(x, check = TRUE)

## S3 method for class 'character'
as.boldid(x, check = TRUE)

## S3 method for class 'list'
as.boldid(x, check = TRUE)

## S3 method for class 'numeric'
as.boldid(x, check = TRUE)

## S3 method for class 'data.frame'
as.boldid(x, check = TRUE)

## S3 method for class 'boldid'
as.data.frame(x, ...)

get_boldid_(
  sci,
  messages = TRUE,
  fuzzy = FALSE,
  dataTypes = "basic",
  includeTree = FALSE,
  rows = NA,
  searchterm = NULL,
  ...
)
```

Arguments

`sci`	character; A vector of scientific names. Or, a `taxon_state` object (see taxon-state)
`fuzzy`	(logical) Whether to use fuzzy search or not (default: FALSE).
`dataTypes`	(character) Specifies the datatypes that will be returned. See `bold_search()` for options.
`includeTree`	(logical) If TRUE (default: FALSE), returns a list containing information for parent taxa as well as the specified taxon.
`ask`	logical; should get_tsn be run in interactive mode? If TRUE and more than one TSN is found for teh species, the user is asked for input. If FALSE NA is returned for multiple matches.

`messages`	logical; should progress be printed?
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a boldid class object with one to many identifiers. See `get_boldid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`rank`	(character) A taxonomic rank name. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Filtering` below.
`division`	(character) A division (aka phylum) name. Optional. See `Filtering` below.
`parent`	(character) A parent name (i.e., the parent of the target search taxon). Optional. See `Filtering` below.
`searchterm`	Deprecated, see `sci`
`...`	Curl options passed on to `crul::verb-GET`
`x`	Input to `as.boldid()`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.boldid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

Filtering

The parameters `division`, `parent`, and `rank` are not used in the search to the data provider, but are used in filtering the data down to a subset that is closer to the target you want. For all these parameters, you can use regex strings since we use `grep()` internally to match. Filtering narrows down to the set that matches your query, and removes the rest.

See Also

`classification()`

Other taxonomic-ids: `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_boldid(sci = "Agapostemon")
get_boldid(sci = "Chironomus riparius")
get_boldid(c("Chironomus riparius","Quercus douglasii"))
splist <- names_list('species')
get_boldid(splist, messages=FALSE)
```

```
# Fuzzy searching
get_boldid(sci="Osmi", fuzzy=TRUE)

# Get back a subset
get_boldid(sci="Osmi", fuzzy=TRUE, rows = 1)
get_boldid(sci="Osmi", fuzzy=TRUE, rows = 1:10)
get_boldid(sci=c("Osmi","Aga"), fuzzy=TRUE, rows = 1)
get_boldid(sci=c("Osmi","Aga"), fuzzy=TRUE, rows = 1:3)

# found
get_boldid('Epicordulia princeps')
get_boldid('Arigomphus furcifer')

# When not found
get_boldid("howdy")
get_boldid(c("Chironomus riparius", "howdy"))
get_boldid("Cordulegaster erronea")
get_boldid("Nasiaeshna pentacantha")

# Narrow down results to a division or rank, or both
## Satyrium example
### Results w/o narrowing
get_boldid("Satyrium")
### w/ phylum
get_boldid("Satyrium", division = "Plantae")
get_boldid("Satyrium", division = "Animalia")

## Rank example
get_boldid("Osmia", fuzzy = TRUE)
get_boldid("Osmia", fuzzy = TRUE, rank = "genus")

# Fuzzy filter on any filtering fields
## uses grep on the inside
get_boldid("Satyrium", division = "anim")
get_boldid("Aga", fuzzy = TRUE, parent = "*idae")

# Convert a boldid without class information to a boldid class
as.boldid(get_boldid("Agapostemon")) # already a boldid, returns the same
as.boldid(get_boldid(c("Agapostemon","Quercus douglasii"))) # same
as.boldid(1973) # numeric
as.boldid(c(1973,101009,98597)) # numeric vector, length > 1
as.boldid("1973") # character
as.boldid(c("1973","101009","98597")) # character vector, length > 1
as.boldid(list("1973","101009","98597")) # list, either numeric or character
## dont check, much faster
as.boldid("1973", check=FALSE)
as.boldid(1973, check=FALSE)
as.boldid(c("1973","101009","98597"), check=FALSE)
as.boldid(list("1973","101009","98597"), check=FALSE)

(out <- as.boldid(c(1973,101009,98597)))
data.frame(out)
as.boldid( data.frame(out) )
```

```
# Get all data back
get_boldid_("Osmia", fuzzy=TRUE, rows=1:5)
get_boldid_("Osmia", fuzzy=TRUE, rows=1)
get_boldid_(c("Osmi","Aga"), fuzzy=TRUE, rows = 1:3)

## End(Not run)
```

`get_eolid` *Get the EOL ID from Encyclopedia of Life from taxonomic names.*

Description

Note that EOL doesn't expose an API endpoint for directly querying for EOL taxon ID's, so we first use the function `eol_search()` to find pages that deal with the species of interest, then use `eol_pages()` to find the actual taxon IDs.

Usage

```
get_eolid(
  sci_com,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  rank = NULL,
  data_source = NULL,
  sciname = NULL,
  ...
)

as.eolid(x, check = TRUE)

## S3 method for class 'eolid'
as.eolid(x, check = TRUE)

## S3 method for class 'character'
as.eolid(x, check = TRUE)

## S3 method for class 'list'
as.eolid(x, check = TRUE)

## S3 method for class 'numeric'
as.eolid(x, check = TRUE)

## S3 method for class 'data.frame'
as.eolid(x, check = TRUE)
```

```
## S3 method for class 'eolid'
as.data.frame(x, ...)

get_eolid_(sci_com, messages = TRUE, rows = NA, sciname = NULL, ...)
```

Arguments

`sci_com`	character; one or more scientific or common names. Or, a `taxon_state` object (see taxon-state)
`ask`	logical; should get_eolid be run in interactive mode? If TRUE and more than one ID is found for the species, the user is asked for input. If FALSE NA is returned for multiple matches.
`messages`	logical; If `TRUE` the actual taxon queried is printed on the console.
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a eolid class object with one to many identifiers. See `get_eolid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`rank`	(character) A taxonomic rank name. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Filtering` below.
`data_source`	(character) A data source inside of EOL. These are longish names like e.g., "Barcode of Life Data Systems" or "USDA PLANTS images". Optional. See `Filtering` below.
`sciname`	Deprecated, see `sci_com`
`...`	Further args passed on to `eol_search()`
`x`	Input to `as.eolid()`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.eolid()`

Details

EOL is a bit odd in that they have page IDs for each taxon, but then within that, they have taxon ids for various taxa within that page (e.g., GBIF and NCBI each have a taxon they refer to within the page [i.e., taxon]). And we need the taxon ids from a particular data provider (e.g, NCBI) to do other things, like get a higher classification tree. However, humans want the page id, not the taxon id. So, the id returned from this function is the taxon id, not the page id. You can get the page id for a taxon by using `eol_search()` and ‘`eol_pages()`, and the URI returned in the attributes for a taxon will lead you to the taxon page, and the ID in the URL is the page id.

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can’t pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

Filtering

The parameters `rank` and `data_source` are not used in the search to the data provider, but are used in filtering the data down to a subset that is closer to the target you want. For all these parameters, you can use regex strings since we use `grep()` internally to match. Filtering narrows down to the set that matches your query, and removes the rest.

Author(s)

Scott Chamberlain

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_eolid(sci_com='Pinus contorta')
get_eolid(sci_com='Puma concolor')

get_eolid(c("Puma concolor", "Pinus contorta"))

# specify rows to limit choices available
get_eolid('Poa annua')
get_eolid('Poa annua', rows=1)
get_eolid('Poa annua', rows=2)
get_eolid('Poa annua', rows=1:2)

# When not found
get_eolid(sci_com="uaudnadndj")
get_eolid(c("Chironomus riparius", "uaudnadndj"))

# filter results to a rank or data source, or both
get_eolid("Satyrium")
get_eolid("Satyrium", rank = "genus")
get_eolid("Satyrium", data_source = "INAT")
get_eolid("Satyrium", rank = "genus",
  data_source = "North Pacific Species List")

# Convert a eolid without class information to a eolid class
# already a eolid, returns the same
as.eolid(get_eolid("Chironomus riparius"))
# same
as.eolid(get_eolid(c("Chironomus riparius","Pinus contorta")))
# numeric
as.eolid(10247706)
# numeric vector, length > 1
as.eolid(c(6985636,12188704,10247706))
```

```
# character
as.eolid("6985636")
# character vector, length > 1
as.eolid(c("6985636","12188704","10247706"))
# list, either numeric or character
as.eolid(list("6985636","12188704","10247706"))
## dont check, much faster
as.eolid("6985636", check=FALSE)
as.eolid(6985636, check=FALSE)
as.eolid(c("6985636","12188704","10247706"), check=FALSE)
as.eolid(list("6985636","12188704","10247706"), check=FALSE)

(out <- as.eolid(c(6985636,12188704,10247706)))
data.frame(out)
as.eolid( data.frame(out) )

# Get all data back
get_eolid_("Poa annua")
get_eolid_("Poa annua", rows=2)
get_eolid_("Poa annua", rows=1:2)
get_eolid_(c("asdfadfasd", "Pinus contorta"))

## End(Not run)
```

`get_gbifid` *Get the GBIF backbone taxon ID from taxonomic names.*

Description

Get the GBIF backbone taxon ID from taxonomic names.

Usage

```
get_gbifid(
  sci,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  phylum = NULL,
  class = NULL,
  order = NULL,
  family = NULL,
  rank = NULL,
  method = "backbone",
  sciname = NULL,
  ...
)

as.gbifid(x, check = FALSE)
```

```
## S3 method for class 'gbifid'
as.gbifid(x, check = FALSE)

## S3 method for class 'character'
as.gbifid(x, check = TRUE)

## S3 method for class 'list'
as.gbifid(x, check = TRUE)

## S3 method for class 'numeric'
as.gbifid(x, check = TRUE)

## S3 method for class 'data.frame'
as.gbifid(x, check = TRUE)

## S3 method for class 'gbifid'
as.data.frame(x, ...)

get_gbifid_(
  sci,
  messages = TRUE,
  rows = NA,
  method = "backbone",
  sciname = NULL
)
```

Arguments

`sci`	(character) one or more scientific names. Or, a `taxon_state` object (see taxon-state)
`ask`	logical; should get_gbifid be run in interactive mode? If TRUE and more than one ID is found for the species, the user is asked for input. If FALSE NA is returned for multiple matches.
`messages`	logical; If TRUE the actual taxon queried is printed on the console.
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a gbifid class object with one to many identifiers. See `get_gbifid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`phylum`	(character) A phylum (aka division) name. Optional. See `Filtering` below.
`class`	(character) A class name. Optional. See `Filtering` below.
`order`	(character) An order name. Optional. See `Filtering` below.
`family`	(character) A family name. Optional. See `Filtering` below.
`rank`	(character) A taxonomic rank name. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Filtering` below.

`method`	(character) one of "backbone" or "lookup". See Details.
`sciname`	Deprecated, see `sci`
`...`	Ignored
`x`	Input to `as.gbifid()`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.gbifid()`

Details

Internally in this function we use a function to search GBIF's taxonomy, and if we find an exact match we return the ID for that match. If there isn't an exact match we return the options to you to pick from.

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

method parameter

"backbone" uses the `/species/match` GBIF API route, matching against their backbone taxonomy. We turn on fuzzy matching by default, as the search without fuzzy against backbone is quite narrow. "lookup" uses the `/species/search` GBIF API route, doing a full text search of name usages covering scientific and vernacular named, species descriptions, distributions and the entire classification.

Filtering

The parameters `phylum`, `class`, `order`, `family`, and `rank` are not used in the search to the data provider, but are used in filtering the data down to a subset that is closer to the target you want. For all these parameters, you can use regex strings since we use `grep()` internally to match. Filtering narrows down to the set that matches your query, and removes the rest.

Author(s)

Scott Chamberlain,

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_gbifid(sci='Poa annua')
get_gbifid(sci='Pinus contorta')
get_gbifid(sci='Puma concolor')

#lots of queries
spp <- names_list("species", 10)
res <- get_gbifid(spp)
res
xx <- taxon_last()
xx

# multiple names
get_gbifid(c("Poa annua", "Pinus contorta"))

# specify rows to limit choices available
get_gbifid(sci='Pinus')
get_gbifid(sci='Pinus', rows=10)
get_gbifid(sci='Pinus', rows=1:3)

# When not found, NA given
get_gbifid(sci="uaudnadndj")
get_gbifid(c("Chironomus riparius", "uaudnadndj"))

# Narrow down results to a division or rank, or both
## Satyrium example
### Results w/o narrowing
get_gbifid("Satyrium")
### w/ phylum
get_gbifid("Satyrium", phylum = "Tracheophyta")
get_gbifid("Satyrium", phylum = "Arthropoda")
### w/ phylum & rank
get_gbifid("Satyrium", phylum = "Arthropoda", rank = "genus")

## Rank example
get_gbifid("Poa", method = "lookup")
get_gbifid("Poa", method = "lookup", rank = "genus")
get_gbifid("Poa", method = "lookup", family = "Thripidae")

# Fuzzy filter on any filtering fields
## uses grep on the inside
get_gbifid("Satyrium", phylum = "arthropoda")
get_gbifid("A*", method = "lookup", order = "*tera")
get_gbifid("A*", method = "lookup", order = "*ales")

# Convert a uid without class information to a uid class
as.gbifid(get_gbifid("Poa annua")) # already a uid, returns the same
as.gbifid(get_gbifid(c("Poa annua","Puma concolor"))) # same
as.gbifid(2704179) # numeric
as.gbifid(c(2704179,2435099,3171445)) # numeric vector, length > 1
as.gbifid("2704179") # character
```

```
as.gbifid(c("2704179","2435099","3171445")) # character vector, length > 1
as.gbifid(list("2704179","2435099","3171445")) # list, either numeric or character
## dont check, much faster
as.gbifid("2704179", check=FALSE)
as.gbifid(2704179, check=FALSE)
as.gbifid(2704179, check=FALSE)
as.gbifid(c("2704179","2435099","3171445"), check=FALSE)
as.gbifid(list("2704179","2435099","3171445"), check=FALSE)

(out <- as.gbifid(c(2704179,2435099,3171445)))
data.frame(out)
as.uid( data.frame(out) )

# Get all data back
get_gbifid_("Puma concolor")
get_gbifid_(c("Pinus", "uaudnadndj"))
get_gbifid_(c("Pinus", "Puma"), rows=5)
get_gbifid_(c("Pinus", "Puma"), rows=1:5)

# use curl options
invisible(get_gbifid("Quercus douglasii", verbose = TRUE))

## End(Not run)
```

`get_ids` *Retrieve taxonomic identifiers for a given taxon name.*

Description

This is a convenience function to get identifiers across all data sources. You can use other `get_*` functions to get identifiers from specific sources if you like.

Usage

```
get_ids(
  sci_com,
  db = c("itis", "ncbi", "eol", "tropicos", "gbif", "nbn", "pow"),
  suppress = FALSE,
  names = NULL,
  ...
)

get_ids_(
  sci_com,
  db = get_ids_dbs,
  rows = NA,
  suppress = FALSE,
  names = NULL,
  ...
)
```

Arguments

`sci_com`	(character) Taxonomic name to query.
`db`	(character) database to query. One or more of `ncbi`, `itis`, `eol`, `tropicos`, `gbif`, `nbn`, or `pow`. By default db is set to search all data sources. Note that each taxonomic data source has their own identifiers, so that if you give the wrong db value for the identifier you could get a result, it will likely be wrong (not what you were expecting). If using ncbi and/or tropicos we recommend getting API keys; see taxize-authentication
`suppress`	(logical) suppress **cli** separators with the database name being queried. default: `FALSE`
`names`	Deprecated, see `sci_com`
`...`	Other arguments passed to `get_tsn()`, `get_uid()`, `get_eolid()`, `get_tpsid()`, `get_gbifid()`, `get_nbnid()`.
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are returned. When used in `get_ids` this function still only gives back a ids class object with one to many identifiers. See `get_ids_` to get back all, or a subset, of the raw data that you are presented during the ask process.

Value

A vector of taxonomic identifiers, each retaining their respective S3 classes so that each element can be passed on to another function (see e.g.'s).

Authentication

See taxize-authentication for help on authentication

Note

There is a timeout of 1/3 seconds between queries to NCBI.

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
# Plug in taxon names directly
# specify rows to limit choices available
get_ids("Poa annua", db="eol", rows=1)
get_ids("Poa annua", db="eol", rows=1:2)

## Or you can specify which source you want via the db parameter
get_ids("Chironomus riparius", db = 'ncbi')
```

```
get_ids("Salvelinus fontinalis", db = 'nbn')

get_ids(c("Chironomus riparius", "Pinus contorta"), db = 'ncbi')
get_ids("Pinus contorta", db = c('ncbi','eol','tropicos'))
get_ids("ava avvva", db = c('ncbi','eol','tropicos'))

# Pass on to other functions
out <- get_ids("Pinus contorta", db = c('ncbi','eol','tropicos'))
classification(out$ncbi)

# Get all data back
get_ids_(c("Chironomus riparius", "Pinus contorta"), db = 'nbn',
  rows=1:10)
get_ids_(c("Chironomus riparius", "Pinus contorta"), db = c('nbn','gbif'),
  rows=1:10)

# use curl options
get_ids("Agapostemon", db = "ncbi", verbose = TRUE)

## End(Not run)
```

`get_id_details` *Details on* `get_*()` *functions*

Description

Including outputs from `get_*()` functions, as well as their attributes, and all exception behaviors.

Details

This document applies to the following functions:

- `get_boldid()`
- `get_eolid()`
- `get_gbifid()`
- `get_ids()`
- `get_iucn()`
- `get_natservid()`
- `get_nbnid()`
- `get_tolid()`
- `get_tpsid()`
- `get_tsn()`
- `get_ubioid()`
- `get_uid()`
- `get_wiki()`
- `get_wormsid()`

attributes

Each output from `get_*()` functions have the following attributes:

- *match* (character) - the reason for NA, either 'not found', 'found' or if `ask = FALSE` then 'NA due to ask=FALSE')
- *multiple_matches* (logical) - Whether multiple matches were returned by the data source. This can be `TRUE`, even if you get 1 name back because we try to pattern match the name to see if there's any direct matches. So sometimes this attribute is `TRUE`, as well as `pattern_match`, which then returns 1 resulting name without user prompt.
- *pattern_match* (logical) - Whether a pattern match was made. If `TRUE` then`multiple_matches` must be `TRUE`, and we found a perfect match to your name, ignoring case. If `FALSE`, there wasn't a direct match, and likely you need to pick from many choices or further parameters can be used to limit results
- *uri* (character) - The URI where more information can be read on the taxon

- includes the taxonomic identifier in the URL somewhere. This may be missing if the value returned is `NA`

exceptions

The following are the various ways in which `get_*()` functions behave:

- success - the value returned is a character string or numeric
- no matches found - you'll get an NA, refine your search or possible the taxon searched for does not exist in the database you're using
- more than on match and `ask = FALSE` - if there's more than one matching result, and you have set `ask = FALSE`, then we can't determine the single match to return, so we give back `NA`. However, in this case we do set the `match` attribute to say `NA due to ask=FALSE & > 1 result` so it's very clear what happened - and you can even programatically check this as well
- NA due to some other reason - some `get_*()` functions have additional parameters for filtering taxa. It's possible that even though there's results (that is, `found` will say `TRUE`), you can get back an NA. This is most likely if the parameter filters taxa after they are returned from the data provider and the value passed to the parameter leads to no matches.

`get_iucn` *Get a IUCN Redlist taxon*

Description

Get a IUCN Redlist taxon

Usage

```
get_iucn(sci, messages = TRUE, key = NULL, x = NULL, ...)

as.iucn(x, check = TRUE, key = NULL)

## S3 method for class 'iucn'
as.iucn(x, check = TRUE, key = NULL)

## S3 method for class 'character'
as.iucn(x, check = TRUE, key = NULL)

## S3 method for class 'list'
as.iucn(x, check = TRUE, key = NULL)

## S3 method for class 'numeric'
as.iucn(x, check = TRUE, key = NULL)

## S3 method for class 'data.frame'
as.iucn(x, check = TRUE, key = NULL)

## S3 method for class 'iucn'
as.data.frame(x, ...)
```

Arguments

`sci`	(character) A vector of scientific names. Or, a `taxon_state` object (see taxon-state)
`messages`	logical; should progress be printed?
`key`	(character) required. you IUCN Redlist API key. See rredlist::rredlist-package for help on authenticating with IUCN Redlist
`x`	For `get_iucn()`: Deprecated, see `sci`. For `as.iucn()`, various, see examples
`...`	Ignored
`check`	(logical) Check if ID matches any existing on the DB, only used in `as.iucn()`

Details

There is no underscore method, because there's no real search for IUCN, that is, where you search for a string, and get back a bunch of results due to fuzzy matching. If that exists in the future we'll add an underscore method here.

IUCN ids only work with `synonyms()` and `sci2comm()` methods.

Value

A vector of taxonomic identifiers as an S3 class.

Comes with the following attributes:

- *match* (character) - the reason for NA, either 'not found', 'found' or if `ask = FALSE` then 'NA due to ask=FALSE')
- *name* (character) - the taxonomic name, which is needed in `synonyms()` and `sci2comm()` methods since they internally use **rredlist** functions which require the taxonomic name, and not the taxonomic identifier
- *ri* (character) - The URI where more information can be read on the taxon - includes the taxonomic identifier in the URL somewhere

multiple_matches and *pattern_match* do not apply here as in other `get_*` methods since there is no IUCN Redlist search, so you either get a match or you do not get a match.

See Also

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_iucn("Branta canadensis")
get_iucn("Branta bernicla")
get_iucn("Panthera uncia")


## End(Not run)
```

`get_natservid` *Get NatureServe taxonomic ID for a taxon name*

Description

Get NatureServe taxonomic ID for a taxon name

Usage

```
get_natservid(
  sci_com,
  searchtype = "scientific",
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  query = NULL,
  ...
)

as.natservid(x, check = TRUE)
```

```
## S3 method for class 'natservid'
as.natservid(x, check = TRUE)

## S3 method for class 'character'
as.natservid(x, check = TRUE)

## S3 method for class 'list'
as.natservid(x, check = TRUE)

## S3 method for class 'numeric'
as.natservid(x, check = TRUE)

## S3 method for class 'data.frame'
as.natservid(x, check = TRUE)

## S3 method for class 'natservid'
as.data.frame(x, ...)

get_natservid_(
  sci_com,
  searchtype = "scientific",
  messages = TRUE,
  rows = NA,
  query = NULL,
  ...
)
```

Arguments

`sci_com`	character; A vector of common or scientific names. Or, a `taxon_state` object (see taxon-state)
`searchtype`	character; One of 'scientific' (default) or 'common'. This doesn't affect the query to NatureServe - but rather affects what column of data is targeted in name filtering post data request.
`ask`	logical; should get_natservid be run in interactive mode? If `TRUE` and more than one wormsid is found for the species, the user is asked for input. If `FALSE` NA is returned for multiple matches. default: `TRUE`
`messages`	logical; should progress be printed? default: `TRUE`
`rows`	numeric; Any number from 1 to infinity. If the default NaN, all rows are considered. Note that this function still only gives back a natservid class object with one to many identifiers. See `get_natservid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`query`	Deprecated, see `sci_com`
`...`	curl options passed on to crul::verb-POST
`x`	Input to `as.natservid`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.natservid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

Note

Authentication no longer required

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
(x <- get_natservid("Helianthus annuus", verbose = TRUE))
attributes(x)
attr(x, "match")
attr(x, "multiple_matches")
attr(x, "pattern_match")
attr(x, "uri")

get_natservid('Gadus morhua')
get_natservid(c("Helianthus annuus", 'Gadus morhua'))

# specify rows to limit choices available
get_natservid('Ruby Quaker Moth', 'common')
get_natservid('Ruby*', 'common')
get_natservid('Ruby*', 'common', rows=1)
get_natservid('Ruby*', 'common', rows=1:2)

# When not found
get_natservid("howdy")
get_natservid(c('Gadus morhua', "howdy"))

# Convert a natservid without class information to a natservid class
# already a natservid, returns the same
as.natservid(get_natservid('Pomatomus saltatrix'))
# same
as.natservid(get_natservid(c('Gadus morhua', 'Pomatomus saltatrix')))
# character
as.natservid(101905)
# character vector, length > 1
as.natservid(c(101905, 101998))
```

```
# list, either numeric or character
as.natservid(list(101905, 101998))
## dont check, much faster
as.natservid(101905, check = FALSE)
as.natservid(c(101905, 101998), check = FALSE)
as.natservid(list(101905, 101998), check = FALSE)

(out <- as.natservid(c(101905, 101998), check = FALSE))
data.frame(out)
as.natservid( data.frame(out) )

# Get all data back
get_natservid_("Helianthus")
get_natservid_("Ruby*", searchtype = "common")
get_natservid_("Ruby*", searchtype = "common", rows=1:3)

## End(Not run)
```

`get_nbnid` *Get the UK National Biodiversity Network ID from taxonomic names.*

Description

Get the UK National Biodiversity Network ID from taxonomic names.

Usage

```
get_nbnid(
  sci_com,
  ask = TRUE,
  messages = TRUE,
  rec_only = FALSE,
  rank = NULL,
  rows = NA,
  name = NULL,
  ...
)

as.nbnid(x, check = TRUE)

## S3 method for class 'nbnid'
as.nbnid(x, check = TRUE)

## S3 method for class 'character'
as.nbnid(x, check = TRUE)

## S3 method for class 'list'
as.nbnid(x, check = TRUE)
```

```
## S3 method for class 'data.frame'
as.nbnid(x, check = TRUE)

## S3 method for class 'nbnid'
as.data.frame(x, ...)

get_nbnid_(
  sci_com,
  messages = TRUE,
  rec_only = FALSE,
  rank = NULL,
  rows = NA,
  name = NULL,
  ...
)
```

Arguments

`sci_com`	character; a vector of common or scientific names. Or, a `taxon_state` object (see taxon-state)
`ask`	logical; should get_nbnid be run in interactive mode? If TRUE and more than one ID is found for the species, the user is asked for input. If FALSE NA is returned for multiple matches.
`messages`	logical; If TRUE the actual taxon queried is printed on the console.
`rec_only`	(logical) If `TRUE` ids of recommended names are returned (i.e. synonyms are removed). Defaults to `FALSE`. Remember, the id of a synonym is a taxa with 'recommended' name status.
`rank`	(character) If given, we attempt to limit the results to those taxa with the matching rank.
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a nbnid class object with one to many identifiers. See `get_nbnid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`name`	Deprecated, see `sci_com`
`...`	Further args passed on to `nbn_search`
`x`	Input to `as.nbnid()`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.nbnid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

an object of class nbnid, a light wrapper around a character string that is the taxonomic ID - includes attributes with relavant metadata

Author(s)

Scott Chamberlain,

References

https://api.nbnatlas.org/

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Other nbn: `nbn_classification()`, `nbn_search()`, `nbn_synonyms()`

Examples

```
## Not run:
get_nbnid(sci_com='Poa annua')
get_nbnid(sci_com='Poa annua', rec_only=TRUE)
get_nbnid(sci_com='Poa annua', rank='Species')
get_nbnid(sci_com='Poa annua', rec_only=TRUE, rank='Species')
get_nbnid(sci_com='Pinus contorta')

# The NBN service handles common names too
get_nbnid(sci_com='red-winged blackbird')

# specify rows to limit choices available
get_nbnid('Poa ann')
get_nbnid('Poa ann', rows=1)
get_nbnid('Poa ann', rows=25)
get_nbnid('Poa ann', rows=1:2)

# When not found
get_nbnid(sci_com="uaudnadndj")
get_nbnid(c("Zootoca vivipara", "uaudnadndj"))
get_nbnid(c("Zootoca vivipara","Chironomus riparius", "uaudnadndj"))

# Convert an nbnid without class information to a nbnid class
as.nbnid(get_nbnid("Zootoca vivipara")) # already a nbnid, returns the same
as.nbnid(get_nbnid(c("Zootoca vivipara","Pinus contorta"))) # same
as.nbnid('NHMSYS0001706186') # character
# character vector, length > 1
as.nbnid(c("NHMSYS0001706186","NHMSYS0000494848","NBNSYS0000010867"))
# list
as.nbnid(list("NHMSYS0001706186","NHMSYS0000494848","NBNSYS0000010867"))
## dont check, much faster
```

```
as.nbnid('NHMSYS0001706186', check=FALSE)
as.nbnid(list("NHMSYS0001706186","NHMSYS0000494848","NBNSYS0000010867"),
  check=FALSE)

(out <- as.nbnid(c("NHMSYS0001706186","NHMSYS0000494848",
  "NBNSYS0000010867")))
data.frame(out)
as.nbnid( data.frame(out) )

# Get all data back
get_nbnid_("Zootoca vivipara")
get_nbnid_("Poa annua", rows=2)
get_nbnid_("Poa annua", rows=1:2)
get_nbnid_(c("asdfadfasd","Pinus contorta"), rows=1:5)

# use curl options
invisible(get_nbnid("Quercus douglasii", verbose = TRUE))

## End(Not run)
```

`get_pow` *Get Kew's Plants of the World code for a taxon*

Description

Get Kew's Plants of the World code for a taxon

Usage

```
get_pow(
  sci_com,
  accepted = FALSE,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  family_filter = NULL,
  rank_filter = NULL,
  x = NULL,
  ...
)

as.pow(x, check = TRUE)

## S3 method for class 'pow'
as.pow(x, check = TRUE)

## S3 method for class 'character'
as.pow(x, check = TRUE)
```

```
## S3 method for class 'list'
as.pow(x, check = TRUE)

## S3 method for class 'data.frame'
as.pow(x, check = TRUE)

## S3 method for class 'pow'
as.data.frame(x, ...)

get_pow_(sci_com, messages = TRUE, rows = NA, x = NULL, ...)
```

Arguments

`sci_com`	character; A vector of common or scientific names. Or, a `taxon_state` object (see taxon-state)
`accepted`	logical; If TRUE, removes names that are not accepted valid names by ITIS. Set to `FALSE` (default) to give back both accepted and unaccepted names.
`ask`	logical; should `get_pow` be run in interactive mode? If TRUE and more than one pow is found for teh species, the user is asked for input. If FALSE NA is returned for multiple matches.
`messages`	logical; should progress be printed?
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a pow class object with one to many identifiers. See `get_pow_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`family_filter`	(character) A division (aka phylum) name to filter data after retrieved from NCBI. Optional. See `Filtering` below.
`rank_filter`	(character) A taxonomic rank name to filter data after retrieved from NCBI. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Filtering` below.
`x`	For `get_pow()`: deprecated, see `sci_com`. For `as.pow`, various, see examples
`...`	Curl options passed on to crul::HttpClient
`check`	logical; Check if ID matches any existing on the DB, only used in `as.pow()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

Filtering

The parameters `family_filter` an`rank_filter`er are not used in the search to the data provider, but are used in filtering the data down to a subset that is closer to the target you want. For these two parameters, you can use regex strings since we use `grep()` internally to match. Filtering narrows down to the set that matches your query, and removes the rest.

Rate-limits

As of February 2019, KEW was limiting to 5 requests per second. Note that they may change that number in the future.

If you get errors that contain `429` you are hitting the rate limit, and you can get around it by doing requests with `Sys.sleep` in between requests.

See Also

`classification()`

Other pow: `pow_lookup()`, `pow_search()`, `pow_synonyms()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_pow(sci_com="Helianthus")
get_pow(c("Helianthus","Quercus douglasii"))

# Get back a subset
get_pow(sci_com="Helianthus", rows = 1)
get_pow(sci_com="Helianthus", rows = 1:10)

# When not found
get_pow("howdy")
get_pow(c("Helianthus annuus", "howdy"))

# Narrow down results
# to accepted names
get_pow("Helianthus", accepted = TRUE)
# to a kingom
get_pow("Helianthus", rank_filter = "genus")
# to accepted names and rank
get_pow("Helianthus annuus", accepted = TRUE, rank_filter = "species")
# to a family
get_pow("flower", family_filter = "Acanthaceae")

# Convert a pow without class information to a pow class
z <- get_pow("Helianthus annuus", accepted = TRUE, rank_filter = "species")
# already a pow, returns the same
as.pow(z)
as.pow("urn:lsid:ipni.org:names:119003-2")
```

```
# character vector, length > 1
ids <- c("urn:lsid:ipni.org:names:119003-2","urn:lsid:ipni.org:names:328247-2")
as.pow(ids)
# list, with character strings
as.pow(as.list(ids))
## dont check, much faster
as.pow("urn:lsid:ipni.org:names:119003-2", check=FALSE)
as.pow(ids, check=FALSE)
as.pow(as.list(ids), check=FALSE)

(out <- as.pow(ids))
data.frame(out)
as.pow( data.frame(out) )

# Get all data back
get_pow_("Quercus", rows=1:5)
get_pow_("Quercus", rows=1)
get_pow_(c("Pinus", "Abies"), rows = 1:3)

## End(Not run)
```

`get_tolid` *Get the OTT id for a search term*

Description

Retrieve the Open Tree of Life Taxonomy (OTT) id of a taxon from OpenTreeOfLife

Usage

```
get_tolid(sci, ask = TRUE, messages = TRUE, rows = NA, sciname = NULL, ...)

as.tolid(x, check = TRUE)

## S3 method for class 'tolid'
as.tolid(x, check = TRUE)

## S3 method for class 'character'
as.tolid(x, check = TRUE)

## S3 method for class 'list'
as.tolid(x, check = TRUE)

## S3 method for class 'numeric'
as.tolid(x, check = TRUE)

## S3 method for class 'data.frame'
as.tolid(x, check = TRUE)
```

```
## S3 method for class 'tolid'
as.data.frame(x, ...)

get_tolid_(sci, messages = TRUE, rows = NA, sciname = NULL)
```

Arguments

`sci`	character; one or more scientific names. Or, a `taxon_state` object (see taxon-state)
`ask`	logical; should `get_tolid` be run in interactive mode? If `TRUE` and more than one TOL is found for the species, the user is asked for input. If `FALSE` NA is returned for multiple matches.
`messages`	logical; should progress be printed?
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a tol class object with one to many identifiers. See `get_tolid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`sciname`	Deprecated, see `sci`
`...`	Ignored
`x`	Input to `as.tolid`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.tolid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_tolid(sci = "Quercus douglasii")
get_tolid(sci = "Chironomus riparius")
get_tolid(c("Chironomus riparius","Quercus douglasii"))
splist <- c("annona cherimola", 'annona muricata', "quercus robur",
"shorea robusta", "pandanus patina", "oryza sativa", "durio zibethinus")
get_tolid(splist, messages=FALSE)
```

```
# specify rows to limit choices available
get_tolid('Arni')
get_tolid('Arni', rows=1)
get_tolid('Arni', rows=1:2)

# When not found
get_tolid("howdy")
get_tolid(c("Chironomus riparius", "howdy"))

# Convert a tol without class information to a tol class
as.tolid(get_tolid("Quercus douglasii")) # already a tol, returns the same
as.tolid(get_tolid(c("Chironomus riparius","Pinus contorta"))) # same
as.tolid(5907893) # numeric
as.tolid(c(3930798,515712,872577)) # numeric vector, length > 1
as.tolid("3930798") # character
as.tolid(c("3930798","515712","872577")) # character vector, length > 1
as.tolid(list("3930798","515712","872577")) # list, either numeric or character
## dont check, much faster
as.tolid("3930798", check=FALSE)
as.tolid(3930798, check=FALSE)
as.tolid(c("3930798","515712","872577"), check=FALSE)
as.tolid(list("3930798","515712","872577"), check=FALSE)

(out <- as.tolid(c(3930798,515712,872577)))
data.frame(out)
as.tolid( data.frame(out) )

# Get all data back
get_tolid_("Arni")
get_tolid_("Arni", rows=1)
get_tolid_("Arni", rows=1:2)
get_tolid_(c("asdfadfasd","Pinus contorta"))

## End(Not run)
```

`get_tpsid` *Get the NameID codes from Tropicos for taxonomic names.*

Description

Get the NameID codes from Tropicos for taxonomic names.

Usage

```
get_tpsid(
  sci,
  ask = TRUE,
  messages = TRUE,
  key = NULL,
  rows = NA,
```

```
  family = NULL,
  rank = NULL,
  sciname = NULL,
  ...
)

as.tpsid(x, check = TRUE)

## S3 method for class 'tpsid'
as.tpsid(x, check = TRUE)

## S3 method for class 'character'
as.tpsid(x, check = TRUE)

## S3 method for class 'list'
as.tpsid(x, check = TRUE)

## S3 method for class 'numeric'
as.tpsid(x, check = TRUE)

## S3 method for class 'data.frame'
as.tpsid(x, check = TRUE)

## S3 method for class 'tpsid'
as.data.frame(x, ...)

get_tpsid_(sci, messages = TRUE, key = NULL, rows = NA, sciname = NULL, ...)
```

Arguments

`sci`	(character) One or more scientific name's as a vector or list. Or, a `taxon_state` object (see taxon-state)
`ask`	logical; should get_tpsid be run in interactive mode? If TRUE and more than one ID is found for the species, the user is asked for input. If FALSE NA is returned for multiple matches.
`messages`	logical; If TRUE the actual taxon queried is printed on the console.
`key`	Your API key; see taxize-authentication
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a tpsid class object with one to many identifiers. See `get_tpsid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`family`	(character) A family name. Optional. See `Filtering` below.
`rank`	(character) A taxonomic rank name. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Filtering` below.
`sciname`	Deprecated, see `sci`

`...`	Other arguments passed to `tp_search()`.
`x`	Input to `as.tpsid()`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.tpsid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

Filtering

The parameters `family` anranknk are not used in the search to the data provider, but are used in filtering the data down to a subset that is closer to the target you want. For all these parameters, you can use regex strings since we use `grep()` internally to match. Filtering narrows down to the set that matches your query, and removes the rest.

Author(s)

Scott Chamberlain,

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tsn()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_tpsid(sci='Poa annua')
get_tpsid(sci='Pinus contorta')

get_tpsid(c("Poa annua", "Pinus contorta"))

# specify rows to limit choices available
get_tpsid('Poa ann')
get_tpsid('Poa ann', rows=1)
get_tpsid('Poa ann', rows=25)
get_tpsid('Poa ann', rows=1:2)

# When not found, NA given (howdy is not a species name, and Chrinomus is a fly)
get_tpsid("howdy")
get_tpsid(c("Chironomus riparius", "howdy"))

# Narrow down results to a division or rank, or both
## Satyrium example
```

```
### Results w/o narrowing
get_tpsid("Satyrium")
### w/ rank
get_tpsid("Satyrium", rank = "var.")
get_tpsid("Satyrium", rank = "sp.")

## w/ family
get_tpsid("Poa")
get_tpsid("Poa", family = "Iridaceae")
get_tpsid("Poa", family = "Orchidaceae")
get_tpsid("Poa", family = "Orchidaceae", rank = "gen.")

# Fuzzy filter on any filtering fields
## uses grep on the inside
get_tpsid("Poa", family = "orchidaceae")
get_tpsid("Aga", fuzzy = TRUE, parent = "*idae")

# pass to classification function to get a taxonomic hierarchy
classification(get_tpsid(sci='Poa annua'))

# Convert a tpsid without class information to a tpsid class
as.tpsid(get_tpsid("Pinus contorta")) # already a tpsid, returns the same
as.tpsid(get_tpsid(c("Chironomus riparius","Pinus contorta"))) # same
as.tpsid(24900183) # numeric
as.tpsid(c(24900183,50150089,50079838)) # numeric vector, length > 1
as.tpsid("24900183") # character
as.tpsid(c("24900183","50150089","50079838")) # character vector, length > 1
as.tpsid(list("24900183","50150089","50079838")) # list, either numeric or character
## dont check, much faster
as.tpsid("24900183", check=FALSE)
as.tpsid(24900183, check=FALSE)
as.tpsid(c("24900183","50150089","50079838"), check=FALSE)
as.tpsid(list("24900183","50150089","50079838"), check=FALSE)

(out <- as.tpsid(c(24900183,50150089,50079838)))
data.frame(out)
as.tpsid( data.frame(out) )

# Get all data back
get_tpsid_("Poa annua")
get_tpsid_("Poa annua", rows=2)
get_tpsid_("Poa annua", rows=1:2)
get_tpsid_(c("asdfadfasd","Pinus contorta"), rows=1:5)

# use curl options
invisible(get_tpsid("Quercus douglasii", messages = TRUE))

## End(Not run)
```

`get_tsn` *Get the TSN code for a search term.*

Description

Retrieve the taxonomic serial numbers (TSN) of a taxon from ITIS.

Usage

```
get_tsn(
  sci_com,
  searchtype = "scientific",
  accepted = FALSE,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  searchterm = NULL,
  ...
)

as.tsn(x, check = TRUE)

## S3 method for class 'tsn'
as.tsn(x, check = TRUE)

## S3 method for class 'character'
as.tsn(x, check = TRUE)

## S3 method for class 'list'
as.tsn(x, check = TRUE)

## S3 method for class 'numeric'
as.tsn(x, check = TRUE)

## S3 method for class 'data.frame'
as.tsn(x, check = TRUE)

## S3 method for class 'tsn'
as.data.frame(x, ...)

get_tsn_(
  sci_com,
  messages = TRUE,
  searchtype = "scientific",
  accepted = TRUE,
  rows = NA,
  searchterm = NULL,
  ...
)
```

Arguments

`sci_com`	character; A vector of common or scientific names. Or, a `taxon_state` object (see taxon-state)
`searchtype`	character; One of 'scientific' or 'common', or any unique abbreviation
`accepted`	logical; If TRUE, removes names that are not accepted valid names by ITIS. Set to `FALSE` (default) to give back both accepted and unaccepted names.
`ask`	logical; should get_tsn be run in interactive mode? If `TRUE` and more than one TSN is found for the species, the user is asked for input. If `FALSE` NA is returned for multiple matches.
`messages`	logical; should progress be printed?
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a tsn class object with one to many identifiers. See `get_tsn_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`searchterm`	Deprecated, see `sci_com`
`...`	Ignored
`x`	Input to as.tsn
`check`	logical; Check if ID matches any existing on the DB, only used in `as.tsn()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_uid()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_tsn("Quercus douglasii")
get_tsn("Chironomus riparius")
get_tsn(c("Chironomus riparius","Quercus douglasii"))
splist <- c("annona cherimola", 'annona muricata', "quercus robur",
"shorea robusta", "pandanus patina", "oryza sativa", "durio zibethinus")
get_tsn(splist, messages=FALSE)

# specify rows to limit choices available
get_tsn('Arni')
```

```
get_tsn('Arni', rows=1)
get_tsn('Arni', rows=1:2)

# When not found
get_tsn("howdy")
get_tsn(c("Chironomus riparius", "howdy"))

# Using common names
get_tsn("black bear", searchtype="common")

# Convert a tsn without class information to a tsn class
as.tsn(get_tsn("Quercus douglasii")) # already a tsn, returns the same
as.tsn(get_tsn(c("Chironomus riparius","Pinus contorta"))) # same
as.tsn(19322) # numeric
as.tsn(c(19322,129313,506198)) # numeric vector, length > 1
as.tsn("19322") # character
as.tsn(c("19322","129313","506198")) # character vector, length > 1
as.tsn(list("19322","129313","506198")) # list, either numeric or character
## dont check, much faster
as.tsn("19322", check=FALSE)
as.tsn(19322, check=FALSE)
as.tsn(c("19322","129313","506198"), check=FALSE)
as.tsn(list("19322","129313","506198"), check=FALSE)

(out <- as.tsn(c(19322,129313,506198)))
data.frame(out)
as.tsn( data.frame(out) )

# Get all data back
get_tsn_("Arni")
get_tsn_("Arni", rows=1)
get_tsn_("Arni", rows=1:2)
get_tsn_(c("asdfadfasd","Pinus contorta"), rows=1:5)

## End(Not run)
```

`get_uid` *Get the UID codes from NCBI for taxonomic names.*

Description

Retrieve the Unique Identifier (UID) of a taxon from NCBI taxonomy browser.

Usage

```
get_uid(
  sci_com,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
```

```
  modifier = NULL,
  rank_query = NULL,
  division_filter = NULL,
  rank_filter = NULL,
  key = NULL,
  sciname = NULL,
  ...
)

as.uid(x, check = TRUE)

## S3 method for class 'uid'
as.uid(x, check = TRUE)

## S3 method for class 'character'
as.uid(x, check = TRUE)

## S3 method for class 'list'
as.uid(x, check = TRUE)

## S3 method for class 'numeric'
as.uid(x, check = TRUE)

## S3 method for class 'data.frame'
as.uid(x, check = TRUE)

## S3 method for class 'uid'
as.data.frame(x, ...)

get_uid_(sci_com, messages = TRUE, rows = NA, key = NULL, sciname = NULL, ...)
```

Arguments

`sci_com`	character; scientific or common name. Or, a `taxon_state` object (see taxon-state)
`ask`	logical; should get_uid be run in interactive mode? If TRUE and more than one TSN is found for the species, the user is asked for input. If FALSE NA is returned for multiple matches.
`messages`	logical; If `TRUE` (default) the actual taxon queried is printed on the console.
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a uid class object with one to many identifiers. See `get_uid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`modifier`	(character) A modifier to the `sci_com` given. Options include: Organism, Scientific Name, Common Name, All Names, Division, Filter, Lineage, GC, MGC, Name Tokens, Next Level, PGC, Properties, Rank, Subtree, Synonym, Text Word. These are not checked, so make sure they are entered correctly, as is.

`rank_query`	(character) A taxonomic rank name to modify the query sent to NCBI. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Querying` below.
`division_filter`	(character) A division (aka phylum) name to filter data after retrieved from NCBI. Optional. See `Filtering` below.
`rank_filter`	(character) A taxonomic rank name to filter data after retrieved from NCBI. See rank_ref for possible options. Though note that some data sources use atypical ranks, so inspect the data itself for options. Optional. See `Filtering` below.
`key`	(character) NCBI Entrez API key. optional. See Details.
`sciname`	Deprecated, see `sci_com`
`...`	Ignored
`x`	Input to `as.uid()`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.uid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

Rate limits

In case you run into errors due to your rate limit being exceeded, see `taxize_options()`, where you can set `ncbi_sleep`.

Querying

The parameter `rank_query` is used in the search sent to NCBI, whereas `rank_filter` filters data after it comes back. The parameter `modifier` adds modifiers to the name. For example, `modifier="Organism"` adds that to the name, giving e.g., `Helianthus[Organism]`.

Filtering

The parameters `division_filter` and `rank_filter` are not used in the search to the data provider, but are used in filtering the data down to a subset that is closer to the target you want. For all these parameters, you can use regex strings since we use `grep()` internally to match. Filtering narrows down to the set that matches your query, and removes the rest.

Beware

NCBI does funny things sometimes. E.g., if you search on Fringella morel, a slight misspelling of the genus name, and a non-existent epithet, NCBI gives back a morel fungal species. In addition, NCBI doesn't really do fuzzy searching very well, so if there is a slight mis-spelling in your names, you likely won't get what you are expecting. The lesson: clean your names before using this function. Other data sources are better about fuzzy matching.

Authentication

See taxize-authentication for help on authentication

Note that even though you can't pass in your key to `as.uid` functions, we still use your Entrez API key if you have it saved as an R option or environment variable.

HTTP version

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Author(s)

Eduard Szoecs, <`eduardszoecs@gmail.com`>

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_wiki()`, `get_wormsid()`

Examples

```
## Not run:
get_uid(c("Chironomus riparius", "Chaetopteryx"))
get_uid(c("Chironomus riparius", "aaa vva"))

# When not found
get_uid("howdy")
get_uid(c("Chironomus riparius", "howdy"))

# Narrow down results to a division or rank, or both
## By modifying the query
### w/ modifiers to the name
get_uid(sci_com = "Aratinga acuticauda", modifier = "Organism")
get_uid(sci_com = "bear", modifier = "Common Name")

### w/ rank query
get_uid(sci_com = "Pinus", rank_query = "genus")
get_uid(sci_com = "Pinus", rank_query = "subgenus")
### division query doesn't really work, for unknown reasons, so not available

## By filtering the result
## Echinacea example
### Results w/o narrowing
get_uid("Echinacea")
### w/ division
get_uid(sci_com = "Echinacea", division_filter = "eudicots")
get_uid(sci_com = "Echinacea", division_filter = "sea urchins")
```

```
## Satyrium example
### Results w/o narrowing
get_uid(sci_com = "Satyrium")
### w/ division
get_uid(sci_com = "Satyrium", division_filter = "monocots")
get_uid(sci_com = "Satyrium", division_filter = "butterflies")

## Rank example
get_uid(sci_com = "Pinus")
get_uid(sci_com = "Pinus", rank_filter = "genus")
get_uid(sci_com = "Pinus", rank_filter = "subgenus")

# Fuzzy filter on any filtering fields
## uses grep on the inside
get_uid("Satyrium", division_filter = "m")

# specify rows to limit choices available
get_uid('Dugesia') # user prompt needed
get_uid('Dugesia', rows=1) # 2 choices, so returns only 1 row, so no choices
get_uid('Dugesia', ask = FALSE) # returns NA for multiple matches

# Go to a website with more info on the taxon
res <- get_uid("Chironomus riparius")
browseURL(attr(res, "uri"))

# Convert a uid without class information to a uid class
as.uid(get_uid("Chironomus riparius")) # already a uid, returns the same
as.uid(get_uid(c("Chironomus riparius","Pinus contorta"))) # same
as.uid(315567) # numeric
as.uid(c(315567,3339,9696)) # numeric vector, length > 1
as.uid("315567") # character
as.uid(c("315567","3339","9696")) # character vector, length > 1
as.uid(list("315567","3339","9696")) # list, either numeric or character
## dont check, much faster
as.uid("315567", check=FALSE)
as.uid(315567, check=FALSE)
as.uid(c("315567","3339","9696"), check=FALSE)
as.uid(list("315567","3339","9696"), check=FALSE)

(out <- as.uid(c(315567,3339,9696)))
data.frame(out)
as.uid( data.frame(out) )

# Get all data back
get_uid_("Puma concolor")
get_uid_("Dugesia")
get_uid_("Dugesia", rows=2)
get_uid_("Dugesia", rows=1:2)
get_uid_(c("asdfadfasd","Pinus contorta"))

# use curl options
get_uid("Quercus douglasii", verbose = TRUE)
```

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

---

get_wiki *Get the page name for a Wiki taxon*

Description

Get the page name for a Wiki taxon

Usage

```
get_wiki(
  sci_com,
  wiki_site = "species",
  wiki = "en",
  ask = TRUE,
  messages = TRUE,
  limit = 100,
  rows = NA,
  x = NULL,
  ...
)

as.wiki(x, check = TRUE, wiki_site = "species", wiki = "en")

## S3 method for class 'wiki'
as.wiki(x, check = TRUE, wiki_site = "species", wiki = "en")

## S3 method for class 'character'
as.wiki(x, check = TRUE, wiki_site = "species", wiki = "en")

## S3 method for class 'list'
as.wiki(x, check = TRUE, wiki_site = "species", wiki = "en")

## S3 method for class 'numeric'
as.wiki(x, check = TRUE, wiki_site = "species", wiki = "en")

## S3 method for class 'data.frame'
as.wiki(x, check = TRUE, wiki_site = "species", wiki = "en")

## S3 method for class 'wiki'
as.data.frame(x, ...)

get_wiki_(
  x,
  messages = TRUE,
  wiki_site = "species",
```

```
  wiki = "en",
  limit = 100,
  rows = NA,
  ...
)
```

Arguments

`sci_com`	(character) A vector of common or scientific names. Or, a `taxon_state` object (see taxon-state)
`wiki_site`	(character) Wiki site. One of species (default), pedia, commons
`wiki`	(character) language. Default: en
`ask`	logical; should get_wiki be run in interactive mode? If `TRUE` and more than one wiki is found for the species, the user is asked for input. If `FALSE` NA is returned for multiple matches.
`messages`	logical; should progress be printed?
`limit`	(integer) number of records to return
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. Note that this function still only gives back a wiki class object with one to many identifiers. See `get_wiki_()` to get back all, or a subset, of the raw data that you are presented during the ask process.
`x`	For `get_wiki()`: deprecated, see `sci_com`. For `as.wiki`, various, see examples
`...`	Ignored
`check`	logical; Check if ID matches any existing on the DB, only used in `as.wiki()`

Details

For `wiki_site = "pedia"` , we use the english language site by default. Set the `wiki` parameter for a different language site.

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wormsid()`

Examples

```
## Not run:
get_wiki(sci_com = "Quercus douglasii")
get_wiki(sci_com = "Quercu")
get_wiki(sci_com = "Quercu", "pedia")
get_wiki(sci_com = "Quercu", "commons")

# diff. wikis with wikipedia
get_wiki("Malus domestica", "pedia")
get_wiki("Malus domestica", "pedia", "fr")

# as coercion
as.wiki("Malus_domestica")
as.wiki("Malus_domestica", wiki_site = "commons")
as.wiki("Malus_domestica", wiki_site = "pedia")
as.wiki("Malus_domestica", wiki_site = "pedia", wiki = "fr")
as.wiki("Malus_domestica", wiki_site = "pedia", wiki = "da")

## End(Not run)
```

`get_wormsid` *Get Worms ID for a taxon name*

Description

Retrieve Worms ID of a taxon from World Register of Marine Species (WORMS).

Usage

```
get_wormsid(
  sci_com,
  searchtype = "scientific",
  marine_only = TRUE,
  fuzzy = NULL,
  accepted = FALSE,
  ask = TRUE,
  messages = TRUE,
  rows = NA,
  query = NULL,
  ...
)

as.wormsid(x, check = TRUE)

## S3 method for class 'wormsid'
as.wormsid(x, check = TRUE)

## S3 method for class 'character'
```

```
as.wormsid(x, check = TRUE)

## S3 method for class 'list'
as.wormsid(x, check = TRUE)

## S3 method for class 'numeric'
as.wormsid(x, check = TRUE)

## S3 method for class 'data.frame'
as.wormsid(x, check = TRUE)

## S3 method for class 'wormsid'
as.data.frame(x, ...)

get_wormsid_(
  sci_com,
  messages = TRUE,
  searchtype = "scientific",
  marine_only = TRUE,
  fuzzy = NULL,
  accepted = TRUE,
  rows = NA,
  query = NULL,
  ...
)
```

Arguments

`sci_com`	character; A vector of common or scientific names. Or, a `taxon_state` object (see taxon-state)
`searchtype`	character; One of 'scientific' or 'common', or any unique abbreviation
`marine_only`	logical; marine only? default: `TRUE` (only used when `searchtype="scientific"`); passed on to `worrms::wm_records_name()`
`fuzzy`	logical; fuzzy search. default: `NULL` (`TRUE` for `searchtype="scientific"` and `FALSE` for `searchtype="common"` to match the default values for those parameters in **worrms** package); passed on to `worrms::wm_records_name()` or `worrms::wm_records_common()`
`accepted`	logical; If TRUE, removes names that are not accepted valid names by WORMS. Set to `FALSE` (default) to give back both accepted and unaccepted names.
`ask`	logical; should get_wormsid be run in interactive mode? If `TRUE` and more than one wormsid is found for the species, the user is asked for input. If `FALSE` NA is returned for multiple matches.
`messages`	logical; should progress be printed?
`rows`	numeric; Any number from 1 to infinity. If the default NaN, all rows are considered. Note that this function still only gives back a wormsid class object with one to many identifiers. See `get_wormsid_()` to get back all, or a subset, of the raw data that you are presented during the ask process.

`query`	Deprecated, see `sci_com`
`...`	Ignored
`x`	Input to `as.wormsid`
`check`	logical; Check if ID matches any existing on the DB, only used in `as.wormsid()`

Value

A vector of taxonomic identifiers as an S3 class. If a taxon is not found an `NA` is given. If more than one identifier is found the function asks for user input if `ask = TRUE`, otherwise returns `NA`. If `ask=FALSE` and `rows` does not equal `NA`, then a data.frame is given back, but not of the uid class, which you can't pass on to other functions as you normally can.

See `get_id_details` for further details including attributes and exceptions

See Also

`classification()`

Other taxonomic-ids: `get_boldid()`, `get_eolid()`, `get_gbifid()`, `get_ids()`, `get_iucn()`, `get_natservid()`, `get_nbnid()`, `get_pow()`, `get_tolid()`, `get_tpsid()`, `get_tsn()`, `get_uid()`, `get_wiki()`

Examples

```
## Not run:
(x <- get_wormsid('Gadus morhua'))
attributes(x)
attr(x, "match")
attr(x, "multiple_matches")
attr(x, "pattern_match")
attr(x, "uri")

get_wormsid('Pomatomus saltatrix')
get_wormsid(c("Gadus morhua", "Lichenopora neapolitana"))

# marine_only
get_wormsid("Apedinella", marine_only=TRUE)
get_wormsid("Apedinella", marine_only=FALSE)

# fuzzy
## searchtype="scientific": fuzzy is TRUE by default
get_wormsid("Platypro", searchtype="scientific", fuzzy=TRUE)
get_wormsid("Platypro", searchtype="scientific", fuzzy=FALSE)
## searchtype="common": fuzzy is FALSE by default
get_wormsid("clam", searchtype="common", fuzzy=FALSE)
get_wormsid("clam", searchtype="common", fuzzy=TRUE)

# by common name
get_wormsid("dolphin", 'common')
get_wormsid("clam", 'common')

# specify rows to limit choices available
```

```
get_wormsid('Plat')
get_wormsid('Plat', rows=1)
get_wormsid('Plat', rows=1:2)

# When not found
get_wormsid("howdy")
get_wormsid(c('Gadus morhua', "howdy"))

# Convert a wormsid without class information to a wormsid class
# already a wormsid, returns the same
as.wormsid(get_wormsid('Gadus morhua'))
# same
as.wormsid(get_wormsid(c('Gadus morhua', 'Pomatomus saltatrix')))
# numeric
as.wormsid(126436)
# numeric vector, length > 1
as.wormsid(c(126436,151482))
# character
as.wormsid("126436")
# character vector, length > 1
as.wormsid(c("126436","151482"))
# list, either numeric or character
as.wormsid(list("126436","151482"))
## dont check, much faster
as.wormsid("126436", check=FALSE)
as.wormsid(126436, check=FALSE)
as.wormsid(c("126436","151482"), check=FALSE)
as.wormsid(list("126436","151482"), check=FALSE)

(out <- as.wormsid(c(126436,151482)))
data.frame(out)
as.wormsid( data.frame(out) )

# Get all data back
get_wormsid_("Plat")
get_wormsid_("Plat", rows=1)
get_wormsid_("Plat", rows=1:2)
get_wormsid_("Plat", rows=1:75)

## End(Not run)
```

`gna_data_sources` *Get metadata about GNA data sources*

Description

Downloads metadata about Global Names Architecture (GNA) data sources available to be used in other GNA functions.

Usage

```
gna_data_sources(output_type = "table", ...)
```

Arguments

`output_type`	What format of output to return. Either `'json'`, `'list'`, or `'table'`.
`...`	Passed to crul::HttpClient.

Author(s)

Zachary S.L. Foster

Examples

```
## Not run:

gna_data_sources()

## End(Not run)
```

`gna_parse` *Parse scientific names using Global Names Parser*

Description

Parse scientific names using Global Names Parser

Usage

```
gna_parse(names, ...)
```

Arguments

`names`	A vector of length 1 or more taxonomic names
`...`	Curl options passed on to crul::verb-GET

Value

A data.frame with results, the submitted names, and the parsed names with additional information.

References

http://gni.globalnames.org/

See Also

`gbif_parse()`, `gni_parse()`

Examples

```
## Not run:
gna_parse("Cyanistes caeruleus")
gna_parse("Plantago minor")
gna_parse("Plantago minor minor")
gna_parse(c("Plantago minor minor","Helianthus annuus texanus"))

# if > 20 names, uses an HTTP POST request
x <- names_list("species", size = 30)
gna_parse(x)

# pass on curl options
gna_parse("Cyanistes caeruleus", verbose = TRUE)

## End(Not run)
```

`gna_search` *Search for taxonomic names using the Global Names Architecture*

Description

Uses the Global Names Index, see http://gni.globalnames.org

Usage

```
gna_search(sci, justtotal = FALSE, parse_names = FALSE, ...)
```

Arguments

`sci` (character) required. Name pattern you want to search for. WARNING: Does not work for common names. Search term may include following options:

- n: A shortcut that allows to put together several elements (e.g., `n:B. bubo Linn. 1750-1800`)
- g: a genus name. (e.g. `g:B.`, `g:Bub.`, `g:Bubo`)
- isp: an infraspecies name (e.g. `sp:bubo`, `sp:gallop.`)
- asp: either species or infraspecies (all sp) (e.g. `asp:bubo`)
- ds: data-sources IDs (e.g., `ds:1,2,3`)
- tx: parent taxon . Uses classification of the first data-source from ds. If data-sources are not set, uses Catalogue of Life. (e.g. `tx:Aves`)
- au: author - Search by author word (e.g. `au:Linnaeus`, `au:Linn.`)
- y: year - Search by year (e.g. `y:2005`)

`justtotal` Return only the total results found.

`parse_names` If `TRUE` use `gni_parse()` on the outputs.

`...` Curl options passed on to crul::verb-GET

Value

data.frame of results.

Author(s)

Scott Chamberlain, Zachary Foster

References

http://gni.globalnames.org/ https://apidoc.globalnames.org/gnames

See Also

`gnr_datasources()`, `gna_search()`

Examples

```
## Not run:
gna_search('n:B. bubo ds:1,2 au:Linn. y:1700-')

## End(Not run)
```

`gna_verifier` *Verify a list of scientific names against biodiversity data-sources.*

Description

This service parses incoming names, executes exact or fuzzy matching as required, and returns the best-scored result. Optionally, it can also return matches from data-sources selected by a user.

Usage

```
gna_verifier(
  names,
  data_sources = c(1, 12),
  all_matches = FALSE,
  capitalize = FALSE,
  species_group = FALSE,
  fuzzy_uninomial = FALSE,
  stats = FALSE,
  main_taxon_threshold = 0.5,
  output_type = "table",
  ...
)
```

Arguments

`names`	A `character` vector of taxon names to verify.
`data_sources`	A `character` or `integer` vector with numbers corresponding to data sources. See the Global Names Architecture documentation for a list of available options.

`all_matches`	When `TRUE`, return all found matches, not only the best one. Multiple results are returned in results. These results are sorted by matching quality, the first result is the same as bestResult.
`capitalize`	When `TRUE`, capitalize the first letter of a name-string.
`species_group`	When `TRUE`, expands the search to species group where applicable.
`fuzzy_uninomial`	When `TRUE`, allows fuzzy matching for uninomial names.
`stats`	When `TRUE`, finds out a kingdom and a taxon (main taxon) that contain most names. It only takes in account the names matched to the Catalogue of Life entries. This option is ignored, if the Catalogue of Life is not included in data-sources.
`main_taxon_threshold`	A `numeric` vector from 0.5 to 1. This sets the minimal percentage for the main taxon discovery.
`output_type`	A `character` vector of length 1, either `table` or `list`, indicating the format of the output. The tabular output only contains values that consistently appear in all results, so `list` output can have additional information. For `list` and `json` outputs, only values for unique taxon names are returned, but the `table` output has rows that correspond 1-1 with the input data.
`...`	Curl options passed on to crul::HttpClient

Value

Depends on the value of the `output_type` option

Author(s)

Zachary S.L. Foster

Examples

```
## Not run:
gna_verifier(c("Helianthus annuus", "Homo saapiens"))
gna_verifier(c("Helianthus annuus", "Homo saapiens"), all_matches = TRUE)

## End(Not run)
```

`gni_details` *Search for taxonomic name details using the Global Names Index*

Description

Uses the Global Names Index, see http://gni.globalnames.org/

Usage

```
gni_details(id, all_records = 1, ...)
```

Arguments

`id`	Name id. Required.
`all_records`	If all_records is 1, GNI returns all records from all repositories for the name string (takes 0, or 1 [default]).
`...`	Curl options passed on to crul::verb-GET

Value

Data.frame of results.

Author(s)

Scott Chamberlain

See Also

`gnr_datasources()`, `gna_search()`.

Examples

```
## Not run:
gni_details(id = 17802847)

# pass on curl options
gni_details(id = 17802847, verbose = TRUE)

## End(Not run)
```

`gnr_datasources` *Global Names Resolver Data Sources*

Description

Retrieve data sources used in the Global Names Resolver

Usage

```
gnr_datasources(..., todf)
```

Arguments

`...`	Curl options passed on to crul::HttpClient
`todf`	defunct, always get a data.frame back now

Value

data.frame/tibble

References

https://resolver.globalnames.org/data_sources

See Also

`gnr_resolve()`, `gna_search()`

Examples

```
## Not run:
# all data sources
gnr_datasources()

# give me the id for EOL
out <- gnr_datasources()
out[out$title == "EOL", "id"]

# Fuzzy search for sources with the word zoo
out <- gnr_datasources()
out[agrep("zoo", out$title, ignore.case = TRUE), ]

## End(Not run)
```

`gnr_resolve` *Resolve names using Global Names Resolver*

Description

NOTE: this function is depreciated and will be removed in a future version. The service this function interacts with is no longer maintained and has been replaced by GNA Verifier, which can be used with the `gna_verifier()` function.

Usage

```
gnr_resolve(
  sci,
  data_source_ids = NULL,
  resolve_once = FALSE,
  with_context = FALSE,
  canonical = FALSE,
  highestscore = TRUE,
  best_match_only = FALSE,
  preferred_data_sources = NULL,
  with_canonical_ranks = FALSE,
  http = "get",
  cap_first = TRUE,
  fields = "minimal",
  names = NULL,
```

```
    ...
  )
```

Arguments

`sci`	character; taxonomic names to be resolved. Doesn't work for vernacular/common names.
`data_source_ids`	character; IDs to specify what data source is searched. See `gnr_datasources()`.
`resolve_once`	logical; Find the first available match instead of matches across all data sources with all possible renderings of a name. When `TRUE`, response is rapid but incomplete.
`with_context`	logical; Reduce the likelihood of matches to taxonomic homonyms. When `TRUE` a common taxonomic context is calculated for all supplied names from matches in data sources that have classification tree paths. Names out of determined context are penalized during score calculation.
`canonical`	logical; If `FALSE` (default), gives back names with taxonomic authorities. If `TRUE`, returns canocial names (without tax. authorities and abbreviations).
`highestscore`	logical; Return those names with the highest score for each searched name? Defunct
`best_match_only`	(logical) If `TRUE`, best match only returned. Default: `FALSE`
`preferred_data_sources`	(character) A vector of one or more data source IDs.
`with_canonical_ranks`	(logical) Returns names with infraspecific ranks, if present. If `TRUE`, we force `canonical=TRUE`, otherwise this parameter would have no effect. Default: `FALSE`
`http`	The HTTP method to use, one of "get" or "post". Default: "get". Use `http="post"` with large queries. Queries with > 300 records use "post" automatically because "get" would fail
`cap_first`	(logical) For each name, fix so that the first name part is capitalized, while others are not. This web service is sensitive to capitalization, so you'll get different results depending on capitalization. First name capitalized is likely what you'll want and is the default. If `FALSE`, names are not modified. Default: `TRUE`
`fields`	(character) One of minimal (default) or all. Minimal gives back just four fields, whereas all gives all fields back.
`names`	Deprecated, see `sci`
`...`	Curl options passed on to crul::HttpClient

Details

See section **Age of datasets in the Global Names Resolver**

Value

A data.frame with one attribute `not_known`: a character vector of taxa unknown to the Global Names Index. Access like `attr(output, "not_known")`, or `attributes(output)$not_known`.

Columns of the output data.frame:

- user_supplied_name (character) - the name you passed in to the `names` parameter, unchanged.
- submitted_name (character) - the actual name submitted to the GNR service
- data_source_id (integer/numeric) - data source ID
- data_source_title (character) - data source name
- gni_uuid (character) - Global Names Index UUID (aka identifier)
- matched_name (character) - the matched name in the GNR service
- matched_name2 (character) - returned if `canonical=TRUE`, in which case **matched_name** is not returned
- classification_path (character) - names of the taxonomic classification tree, with names separated by pipes (`|`)
- classification_path_ranks (character) - ranks of the taxonomic classification tree, with names separated by pipes (`|`)
- classification_path_ids (character) - identifiers of the taxonomic classification tree, with names separated by pipes (`|`)
- taxon_id (character) - taxon identifier
- edit_distance (integer/numeric) - edit distance
- imported_at (character) - date imported
- match_type (integer/numeric) - match type
- match_value (character) - description of match type
- prescore (character) - pre score
- score (numeric) - score
- local_id (character) - local identifier
- url (character) - URL for taxon
- global_id (character) - global identifier
- current_taxon_id (character) - current taxon id
- current_name_string (character) - current name string

  Note that names (i.e. rows) are dropped that are NA, are zero length strings, are not character vectors, or are not found by the API.

Age of datasets in the Global Names Resolver

IMPORTANT: Datasets used in the Global Names Resolver vary in how recently they've been updated. See the `updated_at` field in the output of `gnr_datasources()` for dates when each dataset was last updated.

preferred_data_sources

If `preferred_data_sources` is used, only the preferred data is returned - if it has any results.

Author(s)

Scott Chamberlain

References

http://gnrd.globalnames.org/api http://gnrd.globalnames.org/

See Also

`gnr_datasources()`

Examples

```
## Not run:
gnr_resolve(sci = c("Helianthus annuus", "Homo sapiens"))
gnr_resolve(sci = c("Asteraceae", "Plantae"))

# Using data source 12 (Encyclopedia of Life)
sources <- gnr_datasources()
sources
eol <- sources$id[sources$title == 'EOL']
gnr_resolve(names=c("Helianthos annuus","Homo sapians"), data_source_ids=eol)

# Two species in the NE Brazil catalogue
sps <- c('Justicia brasiliana','Schinopsis brasiliensis')
gnr_resolve(sci = sps, data_source_ids = 145)

# Best match only, compare the two
gnr_resolve(sci = "Helianthus annuus", best_match_only = FALSE)
gnr_resolve(sci = "Helianthus annuus", best_match_only = TRUE)

# Preferred data source
gnr_resolve(sci = "Helianthus annuus", preferred_data_sources = c(3,4))

# Return canonical names - default is canonical=FALSE
head(gnr_resolve(sci = "Helianthus annuus"))
head(gnr_resolve(sci = "Helianthus annuus", canonical=TRUE))

# Return canonical names with authority stripped but
# ranks still present
gnr_resolve("Scorzonera hispanica L. subsp. asphodeloides Wallr.")
## vs.
gnr_resolve("Scorzonera hispanica L. subsp. asphodeloides Wallr.",
   with_canonical_ranks = TRUE)

## End(Not run)
```

`id2name` *Taxonomic IDs to taxonomic names*

Description

Taxonomic IDs to taxonomic names

Usage

```
id2name(id, db = NULL, x = NULL, ...)

## Default S3 method:
id2name(id, db = NULL, x = NULL, ...)

## S3 method for class 'tolid'
id2name(id, ...)

## S3 method for class 'tsn'
id2name(id, ...)

## S3 method for class 'uid'
id2name(id, ...)

## S3 method for class 'wormsid'
id2name(id, ...)

## S3 method for class 'gbifid'
id2name(id, ...)

## S3 method for class 'boldid'
id2name(id, ...)
```

Arguments

`id`	vector of taxonomic IDs (character or numeric)
`db`	(character) database to query. One or more of `tol`, `itis`, `ncbi`, `worms`, `gbif`, or `bold`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using ncbi we recommend getting API keys; see taxize-authentication
`x`	Deprecated, see `id`
`...`	Further args passed on to `tol_id2name` or itis_getrecord, or other internal functions. See those functions for what parameters can be passed on.

Value

A named list of data.frames, named by the input taxonomic ids

HTTP version for NCBI requests

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Examples

```
## Not run:
# ITIS
id2name(19322, db = "itis")

# TOL
id2name(515698, db = "tol")
# get NCBI ID and pass to classification()
x <- id2name(515698, db = "tol")
classification(as.uid(x[[1]]$tax_sources_ncbi))

# NCBI
id2name(315567, db = "ncbi")
id2name(3339, db = "ncbi")
id2name(9696, db = "ncbi")
id2name(c(9695, 9696), db = "ncbi")

# WORMS
id2name(105706, db = "worms")

# GBIF
id2name(2441176, db = "gbif")

# BOLD
id2name(88899, db = "bold")

## End(Not run)
```

`ion` *ION - Index to Organism Names*

Description

ION - Index to Organism Names

Usage

```
ion(x, ...)
```

Arguments

`x`	An LSID number. Required.
`...`	Curl options passed on to crul::verb-GET

Value

A data.frame

References

http://www.organismnames.com

Examples

```
## Not run:
ion(155166)
ion(298678)
ion(4796748) # ursus americanus
ion(1280626) # puma concolor

## End(Not run)
```

`iplant_resolve` *iPlant name resolution*

Description

iPlant name resolution

Usage

```
iplant_resolve(sci, retrieve = "all", query = NULL, ...)
```

Arguments

`sci`	Vector of one or more taxonomic names (no common names)
`retrieve`	Specifies whether to retrieve all matches for the names submitted. One of 'best' (retrieves only the single best match for each name submitted) or 'all' (retrieves all matches)
`query`	Deprecated, see `sci`
`...`	Curl options passed on to crul::verb-GET

Value

A data.frame

Examples

```
## Not run:
iplant_resolve(sci=c("Helianthus annuus", "Homo sapiens"))
iplant_resolve("Helianthusss")
iplant_resolve("Pooa")
iplant_resolve("Helianthusss", verbose = TRUE)

## End(Not run)
```

`ipni_search` *Search for names in the International Plant Names Index (IPNI).*

Description

Note: This data source is also provided in the Global Names Index (GNI) (http://gni.globalnames.org/data_sources). The interface to the data is different among the two services though.

Usage

```
ipni_search(
  family = NULL,
  infrafamily = NULL,
  genus = NULL,
  infragenus = NULL,
  species = NULL,
  infraspecies = NULL,
  publicationtitle = NULL,
  authorabbrev = NULL,
  includepublicationauthors = NULL,
  includebasionymauthors = NULL,
  geounit = NULL,
  addedsince = NULL,
  modifiedsince = NULL,
  isapnirecord = NULL,
  isgcirecord = NULL,
  isikrecord = NULL,
  ranktoreturn = NULL,
  output = "minimal",
  ...
)
```

Arguments

`family`	Family name to search on (Optional)
`infrafamily`	Infrafamilial name to search on (Optional)
`genus`	Genus name to search on (Optional)

`infragenus`	Infrageneric name to search on (Optional)
`species`	Species name to search on (Optional) - Note, this is the epithet, not the full genus - epithet name combination.
`infraspecies`	Infraspecies name to search on (Optional)
`publicationtitle`	Publication name or abbreviation to search on. Again, replace any spaces with a '+' (e.g. 'J.+Bot.') (Optional)
`authorabbrev`	Author standard form to search on (publishing author, basionym author or both - see below) (Optional)
`includepublicationauthors`	TRUE (default) to include the taxon author in the search or FALSE to exclude it
`includebasionymauthors`	TRUE (default) to include the basionum author in the search or FALSE to exclude it
`geounit`	Country name or other geographical unit to search on (see the help pages for more information and warnings about the use of this option) (Optional)
`addedsince`	Date to search on in the format 'yyyy-mm-dd', e.g. 2005-08-01 for all records added since the first of August, 2005. (see the help pages for more information and warnings about the use of this option) (Optional. If supplied must be in format YYYY-MM-DD and must be greater than or equal to 1984-01-01.)
`modifiedsince`	Date to search on in the format 'yyyy-mm-dd', e.g. 2005-08-01 for all records edited since the first of August, 2005. (See the help pages for more information about the use of this option) (Optional. If supplied must be in format YYYY-MM-DD and must be greater than or equal to 1993-01-01.)
`isapnirecord`	FALSE (default) to exclude records from the Australian Plant Name Index
`isgcirecord`	FALSE (default) to exclude records from the Gray Cards Index
`isikrecord`	FALSE (default) to exclude records from the Index Kewensis
`ranktoreturn`	One of a few options to choose the ranks returned. See details.
`output`	One of minimal (default), classic, short, or extended
`...`	Curl options passed on to crul::verb-GET (Optional). Default: returns all ranks.

Details

`ranktoreturn` options:

- "all" - all records
- "fam" - family records
- "infrafam" - infrafamilial records
- "gen" - generic records
- "infragen" - infrageneric records
- "spec" - species records
- "infraspec" - infraspecific records

Value

a tibble (data.frame)

References

https://web.archive.org/web/20190501132148/http://www.ipni.org/link_to_ipni.html

Examples

```
## Not run:
ipni_search(genus='Brintonia', isapnirecord=TRUE, isgcirecord=TRUE,
  isikrecord=TRUE)
ipni_search(genus='Ceanothus')
ipni_search(genus='Pinus', species='contorta')

# Different output formats
ipni_search(genus='Ceanothus')
ipni_search(genus='Ceanothus', output='short')
ipni_search(genus='Ceanothus', output='extended')

## End(Not run)
```

`itis_acceptname` *Retrieve accepted TSN and name*

Description

Retrieve accepted TSN and name

Usage

```
itis_acceptname(searchtsn, ...)
```

Arguments

`searchtsn`	One or more TSN for a taxon (numeric/integer)
`...`	Curl options passed on tocrul::verb-GET

Value

data.frame with with row number equal to input vector length, and with three columns:

- submittedtsn (numeric) - The submitted TSN
- acceptedname (character) - The accepted name - if the submitted TSN is the accepted TSN, then this is `NA_character_` because ITIS does not return a name along with the TSN if it's an accepted name. We could make an extra HTTP request to ITIS, but that means additional time.
- acceptedtsn (numeric) - The accepted TSN
- author (character) - taxonomic authority

Examples

```
## Not run:
# TSN accepted - good name
itis_acceptname(searchtsn = 208527)

# TSN not accepted - input TSN is old
itis_acceptname(searchtsn = 504239)

# many accepted names
ids <- c(18161, 18162, 18163, 18164, 18165, 18166, 46173, 46174,
46178, 46181, 46186, 46193, 46196, 46197, 46200, 46201, 46204,
46207, 46867, 46868)
itis_acceptname(searchtsn = ids)

# many unaccepted names
ids <- c(39087, 46208, 46973, 46976, 46978, 46980, 47295, 47445,
47448, 47512, 47515, 47527, 47546, 47622, 47783, 47786, 47787,
47788, 47835, 47839)
itis_acceptname(searchtsn = ids)

# many: mix of accepted and unaccepted names
ids <- c(18161, 18162, 47527, 47546, 47622, 46200)
itis_acceptname(searchtsn = ids)

## End(Not run)
```

`itis_downstream` *Retrieve all taxa names or TSNs downstream in hierarchy from given TSN.*

Description

Retrieve all taxa names or TSNs downstream in hierarchy from given TSN.

Usage

```
itis_downstream(id, downto, intermediate = FALSE, tsns = NULL, ...)
```

Arguments

`id`	A taxonomic serial number.
`downto`	The taxonomic level you want to go down to. See examples below. The taxonomic level IS case sensitive, and you do have to spell it correctly. See `data(rank_ref)` for spelling.
`intermediate`	(logical) If TRUE, return a list of length two with target taxon rank names, with additional list of data.frame's of intermediate taxonomic groups. Default: FALSE
`tsns`	Deprecated, see `id`
`...`	Further args passed on to `ritis::rank_name()` and `ritis::hierarchy_down()`

Value

Data.frame of taxonomic information downstream to family from e.g., Order, Class, etc., or if `intermediated=TRUE`, list of length two, with target taxon rank names, and intermediate names.

Examples

```
## Not run:
## the plant class Bangiophyceae, tsn 846509
itis_downstream(id = 846509, downto="genus")
itis_downstream(id = 846509, downto="genus", intermediate=TRUE)

# get families downstream from Acridoidea
itis_downstream(id = 650497, "family")
## here, intermediate leads to the same result as the target
itis_downstream(id = 650497, "family", intermediate=TRUE)

# get species downstream from Ursus
itis_downstream(id = 180541, "species")

# get orders down from the Division Rhodophyta (red algae)
itis_downstream(id = 660046, "order")
itis_downstream(id = 660046, "order", intermediate=TRUE)

# get tribes down from the family Apidae
itis_downstream(id = 154394, downto="tribe")
itis_downstream(id = 154394, downto="tribe", intermediate=TRUE)

## End(Not run)
```

`itis_getrecord` *Get full ITIS record for one or more ITIS TSN's or lsid's.*

Description

Get full ITIS record for one or more ITIS TSN's or lsid's.

Usage

```
itis_getrecord(values, by = "tsn", ...)
```

Arguments

`values`	(character) One or more TSN's (taxonomic serial number) or lsid's for a taxonomic group
`by`	(character) By "tsn" (default) or "lsid"
`...`	Further arguments passed on to ritis::full_record

Details

You can only enter values in tsn parameter or lsid, not both.

Examples

```
## Not run:
# by TSN
itis_getrecord(202385)
itis_getrecord(c(202385,70340))

# by lsid
itis_getrecord("urn:lsid:itis.gov:itis_tsn:202385", "lsid")

## End(Not run)
```

`itis_hierarchy` *ITIS hierarchy*

Description

Get hierarchies from TSN values, full, upstream only, or immediate downstream only

Usage

```
itis_hierarchy(tsn, what = "full", ...)
```

Arguments

`tsn`	One or more TSN's (taxonomic serial number). Required.
`what`	One of full (full hierarchy), up (immediate upstream), or down (immediate downstream)
`...`	Further arguments passed on to `ritis::hierarchy_full()` `ritis::hierarchy_up()` or `ritis::hierarchy_down()`

Details

Note that `itis_downstream()` gets taxa downstream to a particular rank, while this function only gets immediate names downstream.

See Also

`itis_downstream()`

Examples

```
## Not run:
# Get full hierarchy
itis_hierarchy(tsn=180543)

# Get hierarchy upstream
itis_hierarchy(tsn=180543, "up")

# Get hierarchy downstream
itis_hierarchy(tsn=180543, "down")

# Many tsn's
itis_hierarchy(tsn=c(180543,41074,36616))

## End(Not run)
```

`itis_kingdomnames` *Get kingdom names*

Description

Get kingdom names

Usage

```
itis_kingdomnames(tsn = NULL, ...)
```

Arguments

`tsn`	One or more TSN’s (taxonomic serial number)
`...`	Further arguments passed on to getkingdomnamefromtsn

Examples

```
## Not run:
itis_kingdomnames(202385)
itis_kingdomnames(tsn=c(202385,183833,180543))

## End(Not run)
```

`itis_lsid` *Get TSN from LSID*

Description

Get TSN from LSID

Usage

```
itis_lsid(lsid = NULL, what = "tsn", ...)
```

Arguments

`lsid`	One or more lsid's
`what`	What to retrieve. One of tsn, record, or fullrecord
`...`	Further arguments passed on to `ritis::lsid2tsn()`, `ritis::record()`, or `ritis::full_record()`

Examples

```
## Not run:
# Get TSN
itis_lsid("urn:lsid:itis.gov:itis_tsn:180543")
itis_lsid(lsid=c("urn:lsid:itis.gov:itis_tsn:180543","urn:lsid:itis.gov:itis_tsn:28726"))

# Get partial record
itis_lsid("urn:lsid:itis.gov:itis_tsn:180543", "record")

# Get full record
itis_lsid("urn:lsid:itis.gov:itis_tsn:180543", "fullrecord")

# An invalid lsid (a tsn actually)
itis_lsid(202385)

## End(Not run)
```

`itis_name` *Get taxonomic names for a given taxonomic name query.*

Description

Get taxonomic names for a given taxonomic name query.

Usage

```
itis_name(query = NULL, get = NULL)
```

Arguments

`query`	TSN number (taxonomic serial number).
`get`	The rank of the taxonomic name to get.

Value

Taxonomic name for the searched taxon.

Examples

```
## Not run:
itis_name(query="Helianthus annuus", get="family")

## End(Not run)
```

`itis_native` *Get jurisdiction data, i.e., native or not native in a region.*

Description

Get jurisdiction data, i.e., native or not native in a region.

Usage

```
itis_native(tsn = NULL, what = "bytsn", ...)
```

Arguments

`tsn`	One or more TSN's (taxonomic serial number)
`what`	One of bytsn, values, or originvalues
`...`	Further arguments passed on to `ritis::jurisdictional_origin()`, `ritis::jurisdiction_values()` or `ritis::jurisdiction_origin_values()`

Examples

```
## Not run:
# Get values
itis_native(what="values")

# Get origin values
itis_native(what="originvalues")

# Get values by tsn
itis_native(tsn=180543)
itis_native(tsn=c(180543,41074,36616))

## End(Not run)
```

`itis_refs` *Get references related to a ITIS TSN.*

Description

Get references related to a ITIS TSN.

Usage

```
itis_refs(tsn, ...)
```

Arguments

`tsn`	One or more TSN's (taxonomic serial number) for a taxonomic group (numeric)
`...`	Further arguments passed on to getpublicationsfromtsn

Examples

```
## Not run:
itis_refs(202385)
itis_refs(c(202385, 70340))

## End(Not run)
```

`itis_taxrank` *Retrieve taxonomic rank name from given TSN.*

Description

Retrieve taxonomic rank name from given TSN.

Usage

```
itis_taxrank(query = NULL, ...)
```

Arguments

`query`	TSN for a taxonomic group (numeric). If query is left as default (NULL), you get all possible rank names, and their TSN's (using function `ritis::rank_names()`. There is slightly different terminology for Monera vs. Plantae vs. Fungi vs. Animalia vs. Chromista, so there are separate terminologies for each group.
`...`	Further arguments passed on to `ritis::rank_name()`

Details

You can print messages by setting `verbose=FALSE`.

Value

Taxonomic rank names or data.frame of all ranks.

Examples

```
## Not run:
# All ranks
itis_taxrank()

# A single TSN
itis_taxrank(query=202385)

# Many TSN's
itis_taxrank(query=c(202385,183833,180543))

## End(Not run)
```

`itis_terms` *Get ITIS terms, i.e., tsn's, authors, common names, and scientific names.*

Description

Get ITIS terms, i.e., tsn's, authors, common names, and scientific names.

Usage

```
itis_terms(query, what = "both", ...)
```

Arguments

`query`	One or more common or scientific names, or partial names
`what`	One of both (search common and scientific names), common (search just common names), or scientific (search just scientific names)
`...`	Further arguments passed on to `ritis::terms()`

Examples

```
## Not run:
# Get terms searching both common and scientific names
itis_terms(query='bear')

# Get terms searching just common names
itis_terms(query='tarweed', "common")

# Get terms searching just scientific names
itis_terms(query='Poa annua', "scientific")

## End(Not run)
```

`iucn_getname` *Get any matching IUCN species names*

Description

Get any matching IUCN species names

Usage

```
iucn_getname(name, verbose = TRUE, ...)
```

Arguments

`name`	character; taxon name
`verbose`	logical; should messages be printed?
`...`	Further arguments passed on to `iucn_summary()`, note that you'll need an API key.

Details

Beware: IUCN functions can give back incorrect data. This isn't our fault. We do our best to get you the correct data quickly, but sometimes IUCN gives back the wrong data, and sometimes Global Names gives back the wrong data. We will fix these as soon as possible. In the meantime, just make sure that the data you get back is correct.

Value

Character vector of names that matched in IUCN

See Also

`iucn_summary()` `iucn_status()`

Examples

```
## Not run:
iucn_getname(name = "Cyanistes caeruleus")
iucn_getname(name = "Panthera uncia")

# not found in IUCN search
iucn_getname(name = "Acacia allenii")

## End(Not run)
```

`iucn_id` *Get an ID for a IUCN listed taxon*

Description

Get an ID for a IUCN listed taxon

Usage

```
iucn_id(sciname, key = NULL, ...)
```

Arguments

`sciname`	character; Scientific name. Should be cleaned and in the format `*<Genus> <Species>*`. One or more.
`key`	(character) required. you IUCN Redlist API key. See rredlist::rredlist-package for help on authenticating with IUCN Redlist
`...`	Curl options passed on to crul::HttpClient

Value

A named list (names are input taxa names) of one or more IUCN IDs. Taxa that aren't found are silently dropped.

Author(s)

Scott Chamberlain,

Examples

```
## Not run:
iucn_id("Branta canadensis")
iucn_id("Branta bernicla")
iucn_id("Panthera uncia")
iucn_id("Lynx lynx")

# many names
iucn_id(c("Panthera uncia", "Lynx lynx"))

# many names, some not found
iucn_id(c("Panthera uncia", "Lynx lynx", "foo bar", "Gorilla gorilla gorilla"))

# a name not found
iucn_id("Foo bar")

## End(Not run)
```

`iucn_status` *Extractor functions for* `iucn`*-class.*

Description

Extractor functions for `iucn`-class.

Usage

```
iucn_status(x, ...)
```

Arguments

`x`	an `iucn`-object as returned b`iucn_summary`ry
`...`	Currently not used

Value

A character vector with the status.

See Also

`iucn_summary()`

Examples

```
## Not run:
ia <- iucn_summary(c("Panthera uncia", "Lynx lynx"))
iucn_status(ia)
## End(Not run)
```

`iucn_summary` *Get a summary from the IUCN Red List*

Description

Get a summary from the IUCN Red List (https://www.iucnredlist.org/).

Usage

```
iucn_summary(x, distr_detail = FALSE, key = NULL, ...)
```

Arguments

`x`	character; Scientific name. Should be cleaned and in the format `*<Genus> <Species>*`.
`distr_detail`	logical; If `TRUE`, the geographic distribution is returned as a list of vectors corresponding to the different range types: native, introduced, etc.
`key`	a Redlist API key, get one from https://apiv3.iucnredlist.org/api/v3/token Required for `iucn_summary`. Defaults to `NULL` in case you have your key stored (see `Redlist Authentication` below).
`...`	curl options passed on to crul::verb-GET

Details

Beware: IUCN functions can give back incorrect data. This isn't our fault. We do our best to get you the correct data quickly, but sometimes IUCN gives back the wrong data, and sometimes Global Names gives back the wrong data. We will fix these as soon as possible. In the meantime, just make sure that the data you get back is correct.

`iucn_summary` has a default method that errors when anything's passed in that's not `character` or `iucn` class - a `iucn_summary.character` method for when you pass in taxon names - and a `iucn_summary.iucn` method so you can pass in iucn class objects as output from `get_iucn()` or `as.iucn()`. If you already have IUCN IDs, coerce them to `iucn` class via `as.iucn(..., check = FALSE)`

Value

A list (for every species one entry) of data returned by `rredlist::rl_species_latest()`.

Redlist Authentication

`iucn_summary` uses the new Redlist API for searching for a IUCN ID, so we use the `rredlist::rl_species()` function internally. This function requires an API key. Get the key at https://apiv3.iucnredlist.org/api/v3/token, and pass it to the `key` parameter, or store in your `.Renviron` file like `IUCN_REDLIST_KEY=yourkey` or in your `.Rprofile` file like `options(iucn_redlist_key="yourkey")`. We strongly encourage you to not pass the key in the function call but rather store it in one of those two files. This key will also set you up to use the **rredlist** package.

Note

Not all data types are available for every species and NA is returned. `iucn_status()` is an extractor function to easily extract status into a vector.

Author(s)

Eduard Szoecs, <`eduardszoecs@gmail.com`>

Philippe Marchand, <`marchand.philippe@gmail.com`>

Scott Chamberlain,

Zachary S.L. Foster

See Also

`iucn_status()`

Examples

```
## Not run:
# if you send a taxon name, an IUCN API key is required
## here, the key is being detected from a .Rprofile file
## or .Renviron file, See "Redlist Authentication" above
iucn_summary("Lutra lutra")

ia <- iucn_summary(c("Panthera uncia", "Lynx lynx"))
ia <- iucn_summary(c("Panthera uncia", "Lynx lynx", "aaa"))
iucn_summary("Muntiacus rooseveltorum/truongsonensis")
iucn_summary(c("Muntiacus rooseveltorum/truongsonensis", "Lynx lynx"))

## get detailed distribution
iac <- iucn_summary(x="Ara chloropterus", distr_detail = TRUE)
iac[[1]]$distr


# If you pass in an IUCN ID, you don't need to pass in a Redlist API Key
# extract status
iucn_status(iac)

## End(Not run)
```

`key_helpers` *Helpers to set up authentication for the different providers.*

Description

Sets up authentication to diverse providers by providing the user a detailed prompt.

Usage

```
use_tropicos()

use_entrez()

use_iucn()
```

Details

Key helpers

`use_tropicos()`

Browses to Tropicos API key request URL and provides instruction on how to store the key. After filling the form you will get the key soon, but not immediately.

`use_entrez()`

Browse NCBI Entrez to help make an API key request and provides instruction on how to store the key. There's no direct URL to request a key, one first needs to log in or register and then to generate a key from one's account.

Note that NCBI Entrez doesn't require that you use an API key, but you should get higher rate limit with a key, so do get one.

`use_iucn()`

Browse IUCN Red List API key request URL and provides instruction on how to store the key. This function wraps `rredlist::rl_use_iucn()` from the `rredlist` package. After filling the form you will get the key soon, but not immediately.

See Also

taxize-authentication

`lowest_common` *Retrieve the lowest common taxon and rank for a given taxon name or ID*

Description

Retrieve the lowest common taxon and rank for a given taxon name or ID

Usage

```
lowest_common(...)

## Default S3 method:
lowest_common(
  sci_id,
  db = NULL,
  rows = NA,
  class_list = NULL,
  low_rank = NULL,
  x = NULL,
  ...
)

## S3 method for class 'uid'
lowest_common(sci_id, class_list = NULL, low_rank = NULL, ...)

## S3 method for class 'tsn'
lowest_common(sci_id, class_list = NULL, low_rank = NULL, ...)

## S3 method for class 'gbifid'
```

```
lowest_common(sci_id, class_list = NULL, low_rank = NULL, ...)

## S3 method for class 'tolid'
lowest_common(sci_id, class_list = NULL, low_rank = NULL, ...)
```

Arguments

`...`	Other arguments passed to `get_tsn()`, `get_uid()`, `get_gbifid()`, `get_tolid()`
`sci_id`	Vector of taxa names (character) or id (character or numeric) to query.
`db`	character; database to query. either `ncbi`, `itis`, `gbif`, `tol`. If using ncbi, we recommend getting an API key; see taxize-authentication
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are considered. Note that this parameter is ignored if you pass in a taxonomic id of any of the acceptable classes: tsn, gbifid, tolid. NCBI has a method for this function but rows doesn't work.
`class_list`	(list) A list of classifications, as returned from `classification()`
`low_rank`	(character) taxonomic rank to return, of length 1
`x`	Deprecated, see `sci_id`

Value

NA when no match, or a data.frame with columns

- name
- rank
- id

Authentication

See taxize-authentication for help on authentication

Author(s)

Jimmy O'Donnell <`jodonnellbio@gmail.com`> Scott Chamberlain

Examples

```
## Not run:
id <- c("9031", "9823", "9606", "9470")
id_class <- classification(id, db = 'ncbi')
lowest_common(id[2:4], db = "ncbi")
lowest_common(id[2:4], db = "ncbi", low_rank = 'class')
lowest_common(id[2:4], db = "ncbi", low_rank = 'family')
lowest_common(id[2:4], class_list = id_class)
lowest_common(id[2:4], class_list = id_class, low_rank = 'class')
lowest_common(id[2:4], class_list = id_class, low_rank = 'family')

# TOL
```

```
taxa <- c("Angraecum sesquipedale", "Dracula vampira",
  "Masdevallia coccinea")
(cls <- classification(taxa, db = "tol"))
lowest_common(taxa, db = "tol", class_list = cls)
lowest_common(get_tolid(taxa), class_list = cls)
xx <- get_tolid(taxa)
lowest_common(xx, class_list = cls)

spp <- c("Sus scrofa", "Homo sapiens", "Nycticebus coucang")
lowest_common(spp, db = "ncbi")
lowest_common(get_uid(spp))

lowest_common(spp, db = "itis")
lowest_common(get_tsn(spp))

gbifid <- c("2704179", "3119195")
lowest_common(gbifid, db = "gbif")

spp <- c("Poa annua", "Helianthus annuus")
lowest_common(spp, db = "gbif")
lowest_common(get_gbifid(spp))

cool_orchid <- c("Angraecum sesquipedale", "Dracula vampira",
  "Masdevallia coccinea")
orchid_ncbi <- get_uid(cool_orchid)
orchid_gbif <- get_gbifid(cool_orchid)

cool_orchids2 <- c("Domingoa haematochila", "Gymnadenia conopsea",
  "Masdevallia coccinea")
orchid_itis <- get_tsn(cool_orchids2)

orchid_hier_ncbi <- classification(orchid_ncbi, db = 'ncbi')
orchid_hier_gbif <- classification(orchid_gbif, db = 'gbif')
orchid_hier_itis <- classification(orchid_itis, db = 'itis')

lowest_common(orchid_ncbi, low_rank = 'class')
lowest_common(orchid_ncbi, class_list = orchid_hier_ncbi,
  low_rank = 'class')
lowest_common(orchid_gbif, low_rank = 'class')
lowest_common(orchid_gbif, orchid_hier_gbif, low_rank = 'class')
lowest_common(get_uid(cool_orchid), low_rank = 'class')
lowest_common(get_uid(cool_orchid), low_rank = 'family')

lowest_common(orchid_ncbi, class_list = orchid_hier_ncbi,
  low_rank = 'subfamily')
lowest_common(orchid_gbif, class_list = orchid_hier_gbif,
  low_rank = 'subfamily')

lowest_common(orchid_itis, class_list = orchid_hier_itis,
  low_rank = 'class')

## Pass in sci. names
nms <- c("Angraecum sesquipedale", "Dracula vampira", "Masdevallia coccinea")
```

```
lowest_common(x = nms, db = "ncbi")
lowest_common(x = nms, db = "gbif")

## End(Not run)
```

`names_list` *Get a random vector of species names.*

Description

Family and order names come from the APG plant names list. Genus and species names come from Theplantlist.org.

Usage

```
names_list(rank = "genus", size = 10)
```

Arguments

`rank`	(character) Taxonomic rank, one of species, genus (default), family, order
`size`	(integer/numeric) Number of names to get. Maximum depends on the rank

Value

character vector of taxonomic names

Author(s)

Scott Chamberlain

Examples

```
names_list()
names_list('species')
names_list('genus')
names_list('family')
names_list('order')
names_list('order', 2)
names_list('order', 15)

# You can get a lot of genus or species names if you want
nrow(theplantlist)
names_list('genus', 500)
```

`nbn_classification` *Search UK National Biodiversity Network database for taxonomic classification*

Description

Search UK National Biodiversity Network database for taxonomic classification

Usage

```
nbn_classification(id, ...)
```

Arguments

`id`	(character) An NBN identifier.
`...`	Further args passed on to crul::verb-GET

Value

A data.frame

Author(s)

Scott Chamberlain,

References

https://api.nbnatlas.org/

See Also

Other nbn: `get_nbnid()`, `nbn_search()`, `nbn_synonyms()`

Examples

```
## Not run:
nbn_classification(id="NHMSYS0000376773")

# get id first, then pass to this fxn
id <- get_nbnid("Zootoca vivipara", rec_only = TRUE, rank = "Species")
nbn_classification(id)

nbn_classification(id="NHMSYS0000502940", verbose = TRUE)

## End(Not run)
```

`nbn_search` *Search UK National Biodiversity Network*

Description

Search UK National Biodiversity Network

Usage

```
nbn_search(
  sci_com,
  fq = NULL,
  order = NULL,
  sort = NULL,
  start = 0,
  rows = 25,
  facets = NULL,
  q = NULL,
  ...
)
```

Arguments

`sci_com`	(character) The query terms(s), a scientific or common name
`fq`	(character) Filters to be applied to the original query. These are additional params of the form fq=INDEXEDFIELD:VALUE e.g. fq=rank:kingdom. See https://species-ws.nbnatlas.org/indexFields for all the fields that are queryable.
`order`	(character) Supports "asc" or "desc"
`sort`	(character) The indexed field to sort by
`start`	(integer) Record offset, to enable paging
`rows`	(integer) Number of records to return
`facets`	(list) Comma separated list of the fields to create facets on e.g. facets=basis_of_record.
`q`	Deprecated, see `sci`
`...`	Further args passed on to crul::HttpClient.

Value

a list with slots for metadata (`meta`) with list of response attributes, and data (`data`) with a data.frame of results

Author(s)

Scott Chamberlain,

References

https://api.nbnatlas.org/

See Also

Other nbn: `get_nbnid()`, `nbn_classification()`, `nbn_synonyms()`

Examples

```
## Not run:
x <- nbn_search(sci_com = "Vulpes")
x$meta$totalRecords
x$meta$pageSize
x$meta$urlParameters
x$meta$queryTitle
head(x$data)

nbn_search(sci_com = "blackbird", start = 4)

# debug curl stuff
nbn_search(sci_com = "blackbird", verbose = TRUE)

## End(Not run)
```

`nbn_synonyms` *Return all synonyms for a taxon name with a given id from NBN*

Description

Return all synonyms for a taxon name with a given id from NBN

Usage

```
nbn_synonyms(id, ...)
```

Arguments

`id`	the taxon identifier code
`...`	Further args passed on to crul::verb-GET

Value

A data.frame

References

https://api.nbnatlas.org/

See Also

Other nbn: `get_nbnid()`, `nbn_classification()`, `nbn_search()`

Examples

```
## Not run:
nbn_synonyms(id = 'NHMSYS0001501147')
nbn_synonyms(id = 'NHMSYS0000456036')

# none
nbn_synonyms(id = 'NHMSYS0000502940')

## End(Not run)
```

`ncbi_children` *Search NCBI for children of a taxon*

Description

Search the NCBI Taxonomy database for uids of children of taxa. Taxa can be referenced by name or uid. Referencing by name is faster

In a few cases, different taxa have the same name (e.g. Satyrium; see examples). If one of these are searched for then the children of both taxa will be returned. This can be avoided by using a uid instead of the name or specifying an ancestor. If an ancestor is provided, only children of both the taxon and its ancestor are returned. This will only fail if there are two taxa with the same name and the same specified ancestor.

Usage

```
ncbi_children(
  name = NULL,
  id = NULL,
  start = 0,
  max_return = 1000,
  ancestor = NULL,
  out_type = c("summary", "uid"),
  ambiguous = FALSE,
  key = NULL,
  ...
)
```

Arguments

`name`	(`character`) The string to search for. Only exact matches found the name given will be returned. Not compatible with `id`.
`id`	(`character/numeric`) The uid to search for. Not compatible with `name`.

`start`	The first record to return. If omitted, the results are returned from the first record (start=0).
`max_return`	(`numeric; length=1`) The maximum number of children to return.
`ancestor`	(`character`) The ancestor of the taxon being searched for. This is useful if there could be more than one taxon with the same name. Has no effect if `id` is used.
`out_type`	(character) Currently either `"summary"` or `"uid"`: • `summary` The output is a list of `data.frame` with children uid, name, and rank. • `uid` A list of character vectors of children uids
`ambiguous`	`logical; length 1` If `FALSE`, children taxa with words like "unclassified", "unknown", "uncultured", or "sp." are removed from the output. NOTE: This option only applies when `out_type= "summary"`.
`key`	(character) NCBI Entrez API key. optional. See Details.
`...`	Curl options passed on to crul::HttpClient

Value

The output type depends on the value of the `out_type` parameter. Taxa that cannot be found will result in `NA`s and a lack of children results in an empty data structure.

Authentication

See `taxize-authentication()` for help on authentication. We strongly recommend getting an API key

HTTP version

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Rate limits

In case you run into errors due to your rate limit being exceeded, see `taxize_options()`, where you can set `ncbi_sleep`.

Author(s)

Zachary Foster <`zacharyfoster1989@gmail.com`>

See Also

`ncbi_get_taxon_summary()`, `children()`

Examples

```
## Not run:
ncbi_children(name="Satyrium") #Satyrium is the name of two different genera
ncbi_children(name="Satyrium", ancestor="Eumaeini") # A genus of butterflies
ncbi_children(name="Satyrium", ancestor="Orchidaceae") # A genus of orchids
ncbi_children(id="266948") #"266948" is the uid for the butterfly genus
ncbi_children(id="62858") #"62858" is the uid for the orchid genus

# use curl options
ncbi_children(name="Satyrium", ancestor="Eumaeini", verbose = TRUE)

## End(Not run)
```

`ncbi_downstream` *Retrieve all taxa names downstream in hierarchy for NCBI*

Description

Retrieve all taxa names downstream in hierarchy for NCBI

Usage

```
ncbi_downstream(id, downto, intermediate = FALSE, ...)
```

Arguments

`id`	(numeric/integer) An NCBI taxonomic identifier
`downto`	The taxonomic level you want to go down to. See examples below. The taxonomic level IS case sensitive, and you do have to spell it correctly. See `data(rank_ref)` for spelling.
`intermediate`	(logical) If `TRUE`, return a list of length two with target taxon rank names, with additional list of data.frame's of intermediate taxonomic groups. Default: `FALSE`
`...`	Further args passed on to `ncbi_children()`

Value

Data.frame of taxonomic information downstream to family from e.g., Order, Class, etc., or if `intermediate=TRUE`, list of length two, with target taxon rank names, and intermediate names.

No Rank

A sticky point with NCBI is that they can have designation for taxonomic rank of "No Rank". So we have no way of programatically knowing what to do with that taxon. Of course one can manually look at a name and perhaps know what it is, or look it up on the web - but we can't do anything programatically. So, no rank things will sometimes be missing.

Authentication

See `taxize-authentication()` for help on authentication. We strongly recommend getting an API key

Author(s)

Scott Chamberlain

Examples

```
## Not run:
## genus Apis
ncbi_downstream(id = 7459, downto="species")

## get intermediate taxa as a separate object
ncbi_downstream(id = 7459, downto="species", intermediate = TRUE)

## Lepidoptera
ncbi_downstream(id = 7088, downto="superfamily")

## families in the ferns (Moniliformopses)
(id <- get_uid("Moniliformopses"))
ncbi_downstream(id = id, downto = "order")

## End(Not run)
```

`ncbi_get_taxon_summary` *NCBI taxon information from uids*

Description

Downloads summary taxon information from the NCBI taxonomy databases for a set of taxonomy UIDs using eutils esummary.

Usage

```
ncbi_get_taxon_summary(id, key = NULL, ...)
```

Arguments

`id`	(character) NCBI taxonomy uids to retrieve information for. See Details.
`key`	(character) NCBI Entrez API key. optional. See Details.
`...`	Curl options passed on to crul::verb-GET

Details

If your input vector or list of NCBI IDs is longer than about 2500 characters (use `nchar(paste(ids, collapse = "+"))`), split the list up into chunks since at about that number of characters you will run into the HTTP 414 error "Request-URI Too Long".

Value

A `data.frame` with the following columns:

- `uid` The uid queried for
- `name` The name of the taxon; a binomial name if the taxon is of rank species
- `rank` The taxonomic rank (e.g. 'Genus')

HTTP version

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Authentication

See taxize-authentication for help on authentication. We strongly recommend getting an API key

Author(s)

Zachary Foster <`zacharyfoster1989@Sgmail.com`>

Examples

```
## Not run:
ncbi_get_taxon_summary(c(1430660, 4751))

# use curl options
ncbi_get_taxon_summary(c(1430660, 4751), verbose = TRUE)

## End(Not run)
```

`ping` *Ping an API used in taxize to see if it's working.*

Description

Ping an API used in taxize to see if it's working.

Usage

```
col_ping(what = "status", ...)

eol_ping(what = "status", ...)

itis_ping(what = "status", ...)

ncbi_ping(what = "status", key = NULL, ...)

tropicos_ping(what = "status", ...)

nbn_ping(what = "status", ...)

gbif_ping(what = "status", ...)

bold_ping(what = "status", ...)

ipni_ping(what = "status", ...)

vascan_ping(what = "status", ...)

fg_ping(what = "status", ...)
```

Arguments

`what`	(character) One of status (default), content, or an HTTP status code. If status, we just check that the HTTP status code is 200, or similar signifying the service is up. If content, we do a simple, quick check to determine if returned content matches what's expected. If an HTTP status code, it must match an appropriate code. See `status_codes()`.
`...`	Curl options passed on to crul::verb-GET
`key`	(character) NCBI Entrez API key. optional. See `get_uid()`

Details

For ITIS, see ritis::description, which provides number of scientific and common names in a character string.

Value

A logical, TRUE or FALSE

HTTP version for NCBI requests

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Examples

```
## Not run:
col_ping()
col_ping("content")
col_ping(200)
col_ping("200")
col_ping(204)

itis_ping()
eol_ping()
ncbi_ping()
tropicos_ping()
nbn_ping()

gbif_ping()
gbif_ping(200)

bold_ping()
bold_ping(200)
bold_ping("content")

ipni_ping()
ipni_ping(200)
ipni_ping("content")

vascan_ping()
vascan_ping(200)
vascan_ping("content")

# curl options
vascan_ping(verbose = TRUE)
eol_ping(500, verbose = TRUE)

## End(Not run)
```

`plantGenusNames` *Vector of plant genus names from ThePlantList*

Description

These names are from http://www.theplantlist.org, and are a randomly chosen subset of genera names for the purpose of having some names to play with for examples in this package.

Format

A vector of length 793

Source

http://www.theplantlist.org

`plantminer` *Search for taxonomy data from Plantminer.com*

Description

Search for taxonomy data from Plantminer.com

Usage

```
plantminer(plants, from = "tpl", messages = TRUE, ...)
```

Arguments

`plants`	(character) Vector of plant species names. Required.
`from`	(character) One of tpl (for theplantlist.com data), or flora (for Brazilian Flora Checklist). Required. Default: `tpl`
`messages`	(logical) informative messages or not. Default: `TRUE`
`...`	curl options passed on to crul::HttpClient

Value

data.frame of results.

Note

you used to need an API key for Plantminer; it's no longer needed

Examples

```
## Not run:
# A single taxon
plantminer("Ocotea pulchella")

# Many taxa
plants <- c("Myrcia lingua", "Myrcia bella", "Ocotea pulchella",
"Miconia", "Coffea arabica var. amarella", "Bleh")
plantminer(plants)

# By deafult, tpl is used, for Theplantlist data,
# toggle the from parameter here
plantminer("Ocotea pulchella", from = "flora")

## End(Not run)
```

`plantNames` *Vector of plant species (genus - specific epithet) names from ThePlantList*

Description

These names are from http://www.theplantlist.org, and are a randomly chosen subset of names of the form genus/specific epithet for the purpose of having some names to play with for examples in this package.

Format

A vector of length 1182

Source

http://www.theplantlist.org

`pow_lookup` *Lookup taxa in Kew's Plants of the World*

Description

Lookup taxa in Kew's Plants of the World

Usage

```
pow_lookup(id, include = NULL, ...)
```

Arguments

`id`	(character) taxon id. required
`include`	(character) vector of additional fields to include in results. options include 'distribution' and 'descriptions'. optional
`...`	Further args passed on to crul::HttpClient.

See Also

Other pow: `get_pow()`, `pow_search()`, `pow_synonyms()`

Examples

```
## Not run:
pow_lookup(id = 'urn:lsid:ipni.org:names:320035-2')
pow_lookup(id = 'urn:lsid:ipni.org:names:320035-2',
  include = "distribution")
pow_lookup(id = 'urn:lsid:ipni.org:names:320035-2',
  include = c("distribution", "descriptions"))

## End(Not run)
```

`pow_search` *Search Kew's Plants of the World*

Description

Search Kew's Plants of the World

Usage

```
pow_search(sci_com, limit = 100, cursor = "*", sort = NULL, q = NULL, ...)
```

Arguments

`sci_com`	(character) query terms, scientific or common name
`limit`	(integer) Number of records to return. default: 100
`cursor`	(character) cursor string
`sort`	(character) The field to sort by and sort order separted with underscore, e.g., `sort="name_desc"`
`q`	Deprecated, see `sci_com`
`...`	Further args passed on to crul::HttpClient.

Value

a list with slots for metadata (`meta`) with list of response attributes, and data (`data`) with a data.frame of results

Author(s)

Scott Chamberlain,

References

https://powo.science.kew.org/

See Also

Other pow: `get_pow()`, `pow_lookup()`, `pow_synonyms()`

Examples

```
## Not run:
x <- pow_search(sci_com = "Quercus")
x$meta
x$meta$totalResults
x$meta$perPage
x$meta$totalPages
x$meta$page
x$meta$cursor
head(x$data)

# pagination
pow_search(sci_com = "sunflower", limit = 2)

# debug curl stuff
invisible(pow_search(sci_com = "Helianthus annuus", verbose = TRUE))

# sort
desc <- pow_search(sci_com = "Helianthus", sort = "name_desc")
desc$data$name
asc <- pow_search(sci_com = "Helianthus", sort = "name_asc")
asc$data$name

## End(Not run)
```

`pow_synonyms` *Lookup synonyms in Kew's Plants of the World*

Description

Lookup synonyms in Kew's Plants of the World

Usage

```
pow_synonyms(id, ...)
```

Arguments

`id`	(character) taxon id. required
`...`	Further args passed on to `pow_lookup()`

See Also

Other pow: `get_pow`(), `pow_lookup`(), `pow_search`()

Examples

```
## Not run:
pow_synonyms(id = 'urn:lsid:ipni.org:names:320035-2')
pow_synonyms(id = 'urn:lsid:ipni.org:names:358881-1')
pow_synonyms(id = 'urn:lsid:ipni.org:names:359855-1')

## End(Not run)
```

`rankagg` *Aggregate data by given taxonomic rank*

Description

Aggregate data by given taxonomic rank

Usage

```
rankagg(data = NULL, datacol = NULL, rank = NULL, fxn = "sum")
```

Arguments

`data`	A data.frame. Column headers must have capitalized ranks (e.g., Genus, Tribe, etc.) (data.frame)
`datacol`	The data column (character)
`rank`	Taxonomic rank to aggregate by (character)
`fxn`	Arithmetic function or vector or functions (character)

Examples

```
if (require(vegan)) {
data(dune.taxon, dune, package='vegan')
dat <- dune.taxon
dat$abundance <- colSums(dune)
rankagg(data=dat, datacol="abundance", rank="Genus")
rankagg(data=dat, "abundance", rank="Family")
rankagg(data=dat, "abundance", rank="Genus", fxn="mean")
rankagg(data=dat, "abundance", rank="Subclass")
rankagg(data=dat, "abundance", rank="Subclass", fxn="sd")
}
```

`rank_ref` *Lookup-table for IDs of taxonomic ranks*

Description

data.frame of 46 rows, with 2 columns:

- rankid - a numeric rank id, consecutive
- ranks - a comma separated vector of names that are considered equal to one another within the row

Details

We use this data.frame to do data sorting/filtering based on the ordering of ranks.

Please let us know if there is a rank that occurs from one of the data sources **taxize** that we don't have in `rank_ref` dataset.

Let us know if you disagree with the ordering of ranks.

Note that `rankid` 280 are essentially "genetic variants"; placed just above 'unspecified' to denote they're not without rank, but they're not really taxonomic ranks either. As far as I know there's no way to delineate among these "genetic variant" types.

`rank_ref_zoo` *Lookup-table for IDs of taxonomic ranks (WoRMS)*

Description

Same as `rank_ref` but specifically for WoRMS, where section/subsection ranks are put between family/order rather than between species/genus.

`resolve` *Resolve names from different data sources*

Description

Resolve names from iPlant's name resolver, and the Global Names Resolver (GNR)

Usage

```
resolve(sci, db = "gnr", query = NULL, ...)
```

Arguments

`sci`	Vector of one or more taxonomic names (common names not supported)
`db`	Source to check names against. One of iplant or gnr. Default: gnr. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting).
`query`	Deprecated, see `sci`
`...`	Curl options passed on to crul::verb-GET or crul::verb-POST. In addition, further named args passed on to each respective function. See examples

Value

A list with length equal to length of the db parameter (number of sources requested), with each element being a data.frame or list with results from that source.

Examples

```
## Not run:
resolve(sci=c("Helianthus annuus", "Homo sapiens"))
resolve(sci="Quercus keloggii", db='gnr')
resolve(sci=c("Helianthus annuus", "Homo sapiens"), db=c('iplant', 'gnr'))
resolve(sci="Quercus keloggii", db=c('iplant', 'gnr'))

# pass in options specific to each source
resolve("Helianthus annuus", db = 'gnr', preferred_data_sources = c(3, 4))
resolve("Helianthus annuus", db = 'iplant', retrieve = 'best')
identical(
 resolve("Helianthus annuus", db = 'iplant', retrieve = 'best')$iplant,
 iplant_resolve("Helianthus annuus", retrieve = 'best')
)

# pass in curl options
resolve(sci="Qercuss", db = "iplant", verbose = TRUE)

## End(Not run)
```

`sci2comm` *Get common names from scientific names.*

Description

Get common names from scientific names.

Usage

```
sci2comm(...)

## Default S3 method:
sci2comm(sci, db = "ncbi", simplify = TRUE, scinames = NULL, ...)

## S3 method for class 'uid'
sci2comm(id, ...)

## S3 method for class 'tsn'
sci2comm(id, simplify = TRUE, ...)

## S3 method for class 'wormsid'
sci2comm(id, simplify = TRUE, ...)

## S3 method for class 'iucn'
sci2comm(id, simplify = TRUE, ...)
```

Arguments

`...`	Further arguments passed on to functions `get_uid()`, `get_tsn()`.
`sci`	character; One or more scientific names or partial names.
`db`	character; Data source, one of `"ncbi"` (default), `"itis"` `"eol"`, `"worms"`, or `"iucn"`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using ncbi or iucn we recommend getting an API key; see taxize-authentication
`simplify`	(logical) If TRUE, simplify output to a vector of names. If FALSE, return variable formats from different sources, usually a data.frame. Only applies to eol and itis. Specify `FALSE` to obtain the language of each vernacular in the output for eol and itis.
`scinames`	Deprecated, see `sci`
`id`	character; identifiers, as returned by `get_tsn()`, `get_uid()`.

Value

List of character vectors, named by input taxon name, or taxon ID. `character(0)` on no match

Authentication

See taxize-authentication for help on authentication

HTTP version for NCBI requests

We hard code `http_version = 2L` to use HTTP/1.1 in HTTP requests to the Entrez API. See `curl::curl_symbols('CURL_HTTP_VERSION')`

Author(s)

Scott Chamberlain

See Also

`comm2sci()`

Examples

```
## Not run:
sci2comm(sci='Helianthus annuus')
sci2comm(sci='Helianthus annuus', db='eol')
sci2comm(sci=c('Helianthus annuus', 'Poa annua'))
sci2comm(sci='Puma concolor', db='ncbi')
sci2comm('Gadus morhua', db='worms')
sci2comm('Pomatomus saltatrix', db='worms')
sci2comm('Loxodonta africana', db='iucn')

# Passing id in, works for sources: itis and ncbi, not eol
sci2comm(get_uid('Helianthus annuus'))
sci2comm(get_wormsid('Gadus morhua'))
sci2comm(get_iucn('Loxodonta africana'))

# Don't simplify returned
sci2comm(get_iucn('Loxodonta africana'), simplify=FALSE)

# Use curl options
sci2comm('Helianthus annuus', db="ncbi", verbose = TRUE)

## End(Not run)
```

`scrapenames` *Find taxon names using Global Names Recognition and Discovery*

Description

Uses the Global Names Recognition and Discovery service, see http://gnrd.globalnames.org/

NOTE: This function sometimes gives data back and sometimes not. The API that this function is using is extremely buggy.

Usage

```
scrapenames(
  url = NULL,
  text = NULL,
  format = "csv",
  bytes_offset = FALSE,
  return_content = FALSE,
```

```
  unique_names = TRUE,
  ambiguous_names = FALSE,
  no_bayes = FALSE,
  odds_details = FALSE,
  language = "detect",
  words_around = 0,
  verification = TRUE,
  sources = NULL,
  all_matches = FALSE,
  ...,
  file = NULL,
  unique = NULL,
  engine = NULL,
  detect_language = NULL,
  data_source_ids = NULL
)
```

Arguments

`url`	(character) If text parameter is empty, and `url` is given, GNfinder will process the URL and will find names in the content of its body.
`text`	(character) Contains the text which will be checked for scientific names. If this parameter is not empty, the `url` parameter is ignored.
`format`	(character) Sets the output format. It can be set to: `"csv"` (the default), `"tsv"`, or `"json"`.
`bytes_offset`	(logical) This changes how the position of a detected name in text is calculated. Normally a name's start and end positions are given as the number of UTF-8 characters from the beginning of the text. If this is `TRUE`, the start and end offsets are recalculated in the number of bytes.
`return_content`	(logical) If this is `TRUE`, the text used for the name detection is returned back. This is especially useful if the input was not a plain UTF-8 text and had to be prepared for name-finding. Then the returned content can be used together with start and end fields of detected name-strings to locate the strings in the text.
`unique_names`	(logical) If this is `TRUE`, the output returns a list of unique names, instead of a list of all name occurrences. Unique list of names does not provide position information of a name in the text.
`ambiguous_names`	(logical) If this is `TRUE`, strings which are simultaneously scientific names and "normal" words are not filtered out from the results. For example, generic names like America, Cancer, Cafeteria will be returned in the results.
`no_bayes`	(logical) If this is `TRUE`, only heuristic algorithms are used for name detection.
`odds_details`	(logical) If `TRUE`, the result will contain odds of all features used for calculation of NaiveBayes odds. Odds describe probability of a name to be 'real'. The higher the odds, the higher the probability that a detected name is not a false positive. Odds are calculated by multiplication of the odds of separate features. Odds details explain how the final odds value is calculated.

`language`	(character) The language of the text. Language value is used for calculation of Bayesian odds. If this parameter is not given, `"eng"` is used by default. Currently only English and German languages are supported. Valid values are: `"eng"`, `"deu"`, and `"detect"`.
`words_around`	(integer) Allows to see the context surrounding a name-string. This sets the number of words located immediately before or after a detected name. These words are then returned in the output. Default is 0, maximum value is 5.
`verification`	(character) When this `TRUE`, there is an additional verification step for detected names. This step requires internet connection and uses https://verifier.globalnames.org/api/v1 for verification queries.
`sources`	Pipe separated list of data source ids to resolve found names against. See list of Data Sources http://resolver.globalnames.org/data_sources
`all_matches`	When this option is true all found results are returned, not only the bestResults. The bestResult field in this case is null, and results field should contain found results of the matches.
`...`	Further args passed to crul::verb-GET
`file`	Defunct. If you feel this is important functionality submit an issue at "https://github.com/ropensci/taxize"
`unique`	Defunct. See the `unique_names` option.
`engine`	Defunct. The API used no longer supports this option.
`detect_language`	Defunct. See the `language` option.
`data_source_ids`	Defunct. See the `sources` option.

Value

A `tibble::tibble()` or list representing parsed JSON output depending on the value of the `format` option.

Author(s)

Scott Chamberlain, Zachary Foster

Examples

```
## Not run:
# Get data from a website using its URL
scrapenames('https://en.wikipedia.org/wiki/Spider')
scrapenames('https://en.wikipedia.org/wiki/Animal')
scrapenames('https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0095068')
scrapenames('https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0080498')

scrapenames(url = 'https://en.wikipedia.org/wiki/Spider', source=c(1, 169))

# Get data from text string
scrapenames(text='A spider named Pardosa moesta Banks, 1892')
```

```
# return OCR content
scrapenames(text='A spider named Pardosa moesta Banks, 1892',
            return_content = TRUE, format = 'json')

## End(Not run)
```

`species_plantarum_binomials`

Species names from Species Plantarum

Description

These names have been compiled from *Species Plantarum* by Carl Linnaeus originally published in 1753. It is the first work to consistently apply binomial names and was the starting point for the naming of plants. The book lists every species of plant known at the time, classified into genera. The dataset provides a useful reference point to see how taxonomic names have changed since their inception. The names were transcribed by Robert W. Kiger.

Format

A data frame with 5940 rows and 3 variables:

- genus First part of the binomial species name for each species within the genus
- epithet specific epithet or second part of the binomial species name for each species
- page_number The following abbreviations sometimes are used in the page_number field.
  - "add." refers to addenda that appear on the unnumbered last page of the index in volume two.
  - "err." refers to the unnumbered page of errata that appears following the index in volume two.
  - "canc." following a page number indicates that the binomial appeared on the cancelled version of that page and does not appear on its replacement (as in the 1957-1959 facsimile edition.

Author(s)

Carl Linnaeus

Source

Hunt Institute for Botanical Documentation

References

Linnaeus, C. 1753. Species Plantarum. 2 vols. Salvius, Stockholm. [Facsimile edition, 1957-1959, Ray Society, London.]

`status_codes` *Get HTTP status codes*

Description

Get HTTP status codes

Usage

```
status_codes()
```

See Also

`ping()`

Examples

```
status_codes()
```

`synonyms` *Retrieve synonyms from various sources given input taxonomic names or identifiers*

Description

Retrieve synonyms from various sources given input taxonomic names or identifiers

Usage

```
synonyms(...)

## Default S3 method:
synonyms(sci_id, db = NULL, rows = NA, x = NULL, ...)

## S3 method for class 'tsn'
synonyms(id, ...)

## S3 method for class 'tpsid'
synonyms(id, ...)

## S3 method for class 'nbnid'
synonyms(id, ...)

## S3 method for class 'wormsid'
synonyms(id, ...)
```

```
## S3 method for class 'iucn'
synonyms(id, ...)

## S3 method for class 'pow'
synonyms(id, ...)

## S3 method for class 'ids'
synonyms(id, ...)

synonyms_df(x)
```

Arguments

`...`	Other passed arguments to internal functions `get_*()` and functions to gather synonyms.
`sci_id`	Vector of taxa names (character) or IDs (character or numeric)
`db`	character; database to query. either `itis`, `tropicos`, `nbn`, `worms`, or `pow`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using tropicos, we recommend getting an API key; see taxize-authentication
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are considered. Note that this parameter is ignored if you pass in a taxonomic id of any of the acceptable classes: tsn, tpsid, nbnid, ids.
`x`	For `synonyms()`: deprecated, see `sci_id`. For `synonyms_df()`, the output of `synonyms()`
`id`	character; identifiers, returned by `get_tsn()`, `get_tpsid()`, `get_nbnid()`, `get_wormsid()`, `get_pow()`

Details

If IDs are supplied directly (not from the `get_*()` functions) you must specify the type of ID.

For `db = "itis"` you can pass in a parameter `accepted` to toggle whether only accepted names are used `accepted = TRUE`, or if all are used `accepted = FALSE`. The default is `accepted = FALSE`

Note that IUCN requires an API key. See rredlist::rredlist-package for help on authentiating with IUCN Redlist

Value

A named list of results with three types of output in each slot:

- if the name was not found: `NA_character_`
- if the name was found but no synonyms found, an empty data.frame (0 rows)
- if the name was found, and synonyms found, a data.frames with the synonyms - the column names vary by data source

See Also

get_tsn() get_tpsid() get_nbnid() get_wormsid() get_iucn() get_pow()

Examples

```
## Not run:
# Plug in taxon IDs
synonyms(526852, db="itis")
synonyms(183327, db="itis") # ID with no synonyms
synonyms("25509881", db="tropicos")
synonyms("NBNSYS0000004629", db='nbn')
synonyms(105706, db='worms')
synonyms(12392, db='iucn')
synonyms('urn:lsid:ipni.org:names:358881-1', db='pow')

# Plug in taxon names directly
synonyms("Pinus contorta", db="itis")
synonyms("Puma concolor", db="itis")
synonyms(c("Poa annua",'Pinus contorta','Puma concolor'), db="itis")
synonyms("Poa annua", db="tropicos")
synonyms("Pinus contorta", db="tropicos")
synonyms(c("Poa annua",'Pinus contorta'), db="tropicos")
synonyms("Pinus sylvestris", db='nbn')
synonyms('Pomatomus', db='worms')
synonyms('Pomatomus saltatrix', db='worms')
synonyms('Lithocarpus mindanaensis', db='pow')
synonyms('Poa annua', db='pow')
synonyms(c('Poa annua', 'Pinus contorta', 'foo bar'), db='pow')

# not accepted names, with ITIS
## looks for whether the name given is an accepted name,
## and if not, uses the accepted name to look for synonyms
synonyms("Acer drummondii", db="itis")
synonyms("Spinus pinus", db="itis")

# Use get_* methods
synonyms(get_tsn("Poa annua"))
synonyms(get_tpsid("Poa annua"))
synonyms(get_nbnid("Carcharodon carcharias"))
synonyms(get_iucn('Loxodonta africana'))
synonyms(get_pow('Lithocarpus mindanaensis'))

# Pass many ids from class "ids"
out <- get_ids(names="Poa annua", db = c('itis','tropicos'))
synonyms(out)

# Use the rows parameter to select certain rows
synonyms("Poa annua", db='tropicos', rows=1)
synonyms("Poa annua", db='tropicos', rows=1:3)
synonyms("Pinus sylvestris", db='nbn', rows=1:3)

# Use curl options
```

```
synonyms("Poa annua", db='tropicos', rows=1, verbose = TRUE)
synonyms("Poa annua", db='itis', rows=1, verbose = TRUE)


# combine many outputs together
x <- synonyms(c("Osmia bicornis", "Osmia rufa", "Osmia"), db = "itis")
synonyms_df(x)

## note here how Pinus contorta is dropped due to no synonyms found
synonyms_df(x)

## note here that ids are taxon identifiers b/c you start with them
x <- synonyms(c(25509881, 13100094), db="tropicos")
synonyms_df(x)

## NBN
x <- synonyms(c('Aglais io', 'Usnea hirta', 'Arctostaphylos uva-ursi'),
  db="nbn")
synonyms_df(x)

## End(Not run)
```

`taxize-authentication` *taxize authentication*

Description

Help on authentication

What is an API?

An API is an Application Programming Interface. The term "API" can be used for lots of scenarios, but in this case we're talking about web APIs, or APIs (interfaces) to web resources. **taxize** interacts with remote databases on the web via their APIs. You don't need to worry about the details of how that all works; just know that some of them require authentication and some do not.

What are API keys?

For those APIs that require authentication, the way that's typically done is through API keys: alphanumeric strings of variable lengths that are supplied with a request to an API.

taxize won't get these keys for you; rather, you have to go get a key for each service, but we do provide information on how to get those keys. See `key_helpers()` for help on how to obtain keys for this package.

Using API keys

You can store API keys as R options in your `.Rprofile` file, or as environment variables in either your `.Renviron` file or `.bash_profile` file, o`.zshrc` file (if you use oh-my-zsh) or similar. See Startup for help on R options and environment variables.

Save your API keys with the following names:

- Tropicos: R option or env var as ’TROPICOS_KEY’
- IUCN: R option or env var as ’IUCN_REDLIST_KEY’
- ENTREZ: R option or env var as ’ENTREZ_KEY’

If you save in .Renviron it looks like: `ENTREZ_KEY=somekey`

If you save in a .bash_profile, .zshrc, or similar file it looks like: `export ENTREZ_KEY=somekey`

If you save in a .Rprofile it looks like: `options(ENTREZ_KEY = "somekey")`

Remember to restart your R session (and to start a new shell window/tab if you’re using the shell) to take advantage of the new R options or environment variables.

We strongly recommend using environment variables (https://en.wikipedia.org/wiki/Environment_variable) over R options because environment variables are widely used across programming languages, operating systems, and computing environments; whereas R options are specific to R.

Note that NCBI Entrez doesn’t require that you use an API key, but you do get a higher rate limit with a key (more requests per time period), from 3 to 10 requests per second, so do get one.

See Also

`key_helpers()`

`taxize-defunct` *Defunct functions in taxize*

Description

The following functions are now defunct (no longer available):

- All COL functions are defunct: `as.colid`, col_children, col_classification, col_downstream, col_search, get_colid dren.colid, classification.colid, downstream.colid, id2name.colid, lowest_common.colid, synonyms.colid, upstream.colid‘
- `col_classification()`: See`classification()`
- `tp_classification()`: See`classification()`
- `eol_hierarchy()`: See`classification()`
- `eol_invasive()`: See `eol` in the **originr** package.
- `use_eol()`: EOL no longer requires an API key
- `tpl_search()`: Use the **Taxonstand** functions `TPL` or `TPLck` directly.
- `get_seqs()`: This function changed name to`ncbi_getbyname()`()].
- `get_genes()`: This function changed name to`ncbi_getbyid()`()].
- `get_genes_avail()`: This function changed name to`ncbi_search()`()].
- `ncbi_getbyname()`: See `ncbi_byname` in the **traits** package.
- `ncbi_getbyid()`: See `ncbi_byid` in the **traits** package.
- `ncbi_search()`: See `ncbi_searcher` in the **traits** package.

- `gisd_isinvasive()`: See `gisd` in the **originr** package.
- `ubio_classification()`: The uBio web services was down for quite a while, is now (as of 2016-05-09) back up, but we don't trust that it will stay up and available.
- `ubio_classification_search()`: The uBio web services was down for quite a while, is now (as of 2016-05-09) back up, but we don't trust that it will stay up and available.
- `ubio_id()`: The uBio web services was down for quite a while, is now (as of 2016-05-09) back up, but we don't trust that it will stay up and available.
- `ubio_ping()`: The uBio web services was down for quite a while, is now (as of 2016-05-09) back up, but we don't trust that it will stay up and available.
- `ubio_search()`: The uBio web services was down for quite a while, is now (as of 2016-05-09) back up, but we don't trust that it will stay up and available.
- `ubio_synonyms()`: The uBio web services was down for quite a while, is now (as of 2016-05-09) back up, but we don't trust that it will stay up and available.
- `get_ubioid()`: The uBio web services are apparently down indefinitely.
- `phylomatic_tree()`: This function is defunct. See `phylomatic` in the package **brranching**
- `phylomatic_format()`: This function is defunct. See `phylomatic_names` in the package **brranching**
- `eubon()`: This function is defunct. Use `eubon_search()`
- `tnrs()`: This function is defunct. Was too unreliable
- `tnrs_sources()`: This function is defunct. Was too unreliable

`taxize-params` *taxize parameters*

Description

Information on standardized parameters across the package

Standardized parameters

- `sci`: scientific name
- `com`: common name
- `id`: name identifier
- `sci_com`: scientific name or common name
- `sci_id`: scientific name or name identifier

We were going to standardize parameter names for cases in which a parameter accepts either of three options: scientific name, common name, or name identifier. However, there was no clear parameter name we could use for this case, so we've left parameter names as they are for the two cases (`get_ids()` and `vascan_search()`)

`taxize_capwords` *Capitalize the first letter of a character string.*

Description

Capitalize the first letter of a character string.

Usage

```
taxize_capwords(s, strict = FALSE, onlyfirst = FALSE)
```

Arguments

`s`	A character string
`strict`	Should the algorithm be strict about capitalizing. Defaults to FALSE.
`onlyfirst`	Capitalize only first word, lowercase all others. Useful for taxonomic names.

Examples

```
taxize_capwords(c("using AIC for model selection"))
taxize_capwords(c("using AIC for model selection"), strict=TRUE)
```

`taxize_cite` *Get citations and licenses for data sources used in taxize*

Description

Get citations and licenses for data sources used in taxize

Usage

```
taxize_cite(fxn = "itis", what = "citation")
```

Arguments

`fxn`	Function to search on. A special case is the package name 'taxize' that will give the citations for the package.
`what`	One of citation (default), license, or both.

Examples

```
taxize_cite(fxn='eol_search')
taxize_cite(fxn='itis_hierarchy')
taxize_cite(fxn='tp_classification')
taxize_cite(fxn='gbif_ping')
taxize_cite(fxn='plantminer')
taxize_cite(fxn='get_natservid_')
taxize_cite(fxn='as.natservid')
taxize_cite(fxn='get_wormsid')
taxize_cite(fxn='as.wormsid')

# Functions that use many data sources
taxize_cite(fxn='synonyms')
taxize_cite(fxn='classification')

# Get the taxize citation
taxize_cite(fxn='taxize')

# Get license information
taxize_cite(fxn='taxize', "license")
```

`taxize_options` *taxize options*

Description

taxize options

Usage

```
taxize_options(taxon_state_messages = NULL, ncbi_sleep = NULL, quiet = FALSE)
```

Arguments

`taxon_state_messages`	(logical) suppress messages? default: `NULL` (same as setting `FALSE`). Set to `TRUE` to suppress messages, and `FALSE` to not suppress messages
`ncbi_sleep`	(numeric/integer) number of seconds to sleep between NCBI ENTREZ http requests. applies to the functions: `classification()`, `comm2sci()`, `genbank2uid()`, `get_uid()` and `ncbi_children()`. defaults: 0.334 (without API key) or 0.101 (with API key). minimum value can not be less than 0.101
`quiet`	(logical) quiet informational output from this function. default: `TRUE`

Examples

```
## Not run:
taxize_options()
taxize_options(FALSE)
```

```
taxize_options(TRUE)
taxize_options(ncbi_sleep = 0.4)
taxize_options(taxon_state_messages = TRUE, ncbi_sleep = 0.4)

## End(Not run)
```

`taxon-state` *Last taxon state object from a `get_*` function call*

Description

Last taxon state object from a `get_*` function call

Usage

```
taxon_last()

taxon_clear()
```

Details

- `taxon_last()`: get the last `taxon_state` object in use
- `taxon_clear()`: clear any data from last `taxon_state` object

The `taxon_state` object is an R6 object that holds data and methods used for keeping track of results gathered within a `get_*` function. You shouldn't create `taxon_state` R6 objects yourself.

Behaviors to be aware of:

- If a `taxon_state` object is not passed you don't need to worry about a previously run `get_*` function interfering with another `get_*` function call - you have to explicitly pass a `taxon_state` object to use `taxon_state`
- The passed in `taxon_state` object must have a `$class` matching that of the `get_*` function being called. For example, you can only pass a `taxon_state` with `$class` of `gbifid` to `get_gbifid()`, and so on.
- If you run `taxon_clear()` while a `get*` function is running, you may lose track of any state known to this package before it was cleared

See the internal method progressor for information on how we control messages in `get*` functions

Value

`taxon_last()` returns an object of class `taxon_state`, the last one used, else `NULL` if none found. `taxon_clear()` clears the saved state

Examples

```
## Not run:
spp <- names_list("species", 3)
res <- get_gbifid(spp)
z <- taxon_last()
z
z$taxa_remaining()
z$taxa_completed()
z$count # active binding; no parens needed

# cleanup
taxon_clear()

## End(Not run)
```

`tax_agg` *Aggregate species data to given taxonomic rank*

Description

Aggregate species data to given taxonomic rank

Usage

```
tax_agg(x, rank, db = "ncbi", messages = FALSE, ...)

## S3 method for class 'tax_agg'
print(x, ...)
```

Arguments

`x`	Community data matrix. Taxa in columns, samples in rows.
`rank`	character; Taxonomic rank to aggregate by.
`db`	character; taxonomic API to use, 'ncbi, 'itis' or both, see `tax_name()`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting). If using ncbi we recommend getting an API key; see taxize-authentication
`messages`	(logical) If FALSE (Default) suppress messages
`...`	Other arguments passed to `get_tsn()` or `get_uid()`

Details

`tax_agg` aggregates (sum) taxa to a specific taxonomic level. If a taxon is not found in the database (ITIS or NCBI) or the supplied taxon is on higher taxonomic level this taxon is not aggregated.

Value

A list of class `tax_agg` with the following items:

- `x` Community data matrix with aggregated data.
- `by` A lookup-table showing which taxa were aggregated.
- `n_pre` Number of taxa before aggregation.
- `rank` Rank at which taxa have been aggregated.

See Also

tax_name

Examples

```
## Not run:
if (requireNamespace("vegan", quietly = TRUE)) {
  # use dune dataset
  data(dune, package='vegan')
  species <- c("Achillea millefolium", "Agrostis stolonifera",
    "Aira praecox", "Alopecurus geniculatus", "Anthoxanthum odoratum",
    "Bellis perennis", "Bromus hordeaceus", "Chenopodium album",
    "Cirsium arvense", "Comarum palustre", "Eleocharis palustris",
    "Elymus repens", "Empetrum nigrum", "Hypochaeris radicata",
    "Juncus articulatus", "Juncus bufonius", "Lolium perenne",
    "Plantago lanceolata", "Poa pratensis", "Poa trivialis",
    "Ranunculus flammula", "Rumex acetosa", "Sagina procumbens",
    "Salix repens", "Scorzoneroides autumnalis", "Trifolium pratense",
    "Trifolium repens", "Vicia lathyroides", "Brachythecium rutabulum",
    "Calliergonella cuspidata")
  colnames(dune) <- species

  # aggregate sample to families
  (agg <- tax_agg(dune, rank = 'family', db = 'ncbi'))

  # extract aggregated community data matrix for further usage
  agg$x
  # check which taxa have been aggregated
  agg$by
}

# A use case where there are different taxonomic levels in the same dataset
spnames <- c('Puma','Ursus americanus','Ursidae')
df <- data.frame(c(1,2,3), c(11,12,13), c(1,4,50))
names(df) <- spnames
out <- tax_agg(x=df, rank = 'family', db='itis')
out$x

# You can input a matrix too
mat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3,
 dimnames=list(NULL, c('Puma concolor','Ursus americanus','Ailuropoda melanoleuca')))
tax_agg(mat, rank = 'family', db='itis')
```

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

`tax_name` *Get taxonomic names for a given rank*

Description

Get taxonomic names for a given rank

Usage

```
tax_name(
  sci,
  get,
  db = "itis",
  pref = "ncbi",
  messages = TRUE,
  query = NULL,
  ...
)
```

Arguments

`sci`	(character) Vector of taxonomic names to query. required.
`get`	(character) The ranks of the taxonomic name to get, see rank_ref. required.
`db`	(character) The database to search from: 'itis', 'ncbi' or 'both'. If 'both' both NCBI and ITIS will be queried. Result will be the union of both. If using ncbi, we recommend getting an API key; see taxize-authentication
`pref`	(character) If db = 'both', sets the preference for the union. Either 'ncbi' (default) or 'itis'. Currently not implemented.
`messages`	(logical) If `TRUE` the actual taxon queried is printed on the console.
`query`	Deprecated, see `sci`
`...`	Other arguments passed to `get_tsn()` or `get_uid()`.

Value

A data.frame with one column for every queried rank, in addition to a column for db and queried term.

Authentication

See taxize-authentication for help on authentication

Note

While `tax_rank()` returns the actual rank of a taxon, `tax_name()` searches and returns any specified rank higher in taxonomy.

See Also

`classification()`

Examples

```
## Not run:
# A case where itis and ncbi use the same names
tax_name(sci = "Helianthus annuus", get = "family", db = "itis")
tax_name(sci = "Helianthus annuus", get = "family", db = "ncbi")
tax_name(sci = "Helianthus annuus", get = c("genus","family","order"),
  db = "ncbi")

# Case where itis and ncbi use different names
tax_name(sci = "Helianthus annuus", get = "kingdom", db = "itis")
tax_name(sci = "Helianthus annuus", get = "kingdom", db = "ncbi")

# multiple rank arguments
tax_name(sci = c("Helianthus annuus","Baetis rhodani"), get = c("genus",
"kingdom"), db = "ncbi")
tax_name(sci = c("Helianthus annuus","Baetis rhodani"), get = c("genus",
"kingdom"), db = "itis")

# query both sources
tax_name(sci=c("Helianthus annuus", 'Baetis rhodani'), get=c("genus",
"kingdom"), db="both")

## End(Not run)
```

`tax_rank` *Get rank for a given taxonomic name.*

Description

Get rank for a given taxonomic name.

Usage

```
tax_rank(sci_id, db = NULL, rows = NA, x = NULL, ...)
```

Arguments

`sci_id`	(character) Vector of one or more taxon names (character) or IDs (character or numeric) to query. Or objects returned from `get_*()` functions like `get_tsn()`
`db`	(character) database to query. either `ncbi`, `itis`, `eol`, `tropicos`, `gbif`,`nbn`, `worms`, `natserv`, `bold`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you may get a result, but it will likely be wrong (not what you were expecting). If using ncbi we recommend getting an API key; see taxize-authentication
`rows`	numeric; Any number from 1 to infinity. If the default NA, all rows are considered. passed down to `get_*()` functions.
`x`	Deprecated, see `sci_id`
`...`	Additional arguments to `classification()`

Value

A named list of character vectors with ranks (all lower-cased)

Note

While `tax_name()` returns the name of a specified rank, `tax_rank()` returns the actual rank of the taxon.

See Also

`classification()`,`tax_name()`

Examples

```
## Not run:
tax_rank("Helianthus annuus", db = "itis")
tax_rank("Helianthus annuus", db = "natserv")
tax_rank(get_tsn("Helianthus annuus"))
tax_rank(c("Helianthus", "Pinus", "Poa"), db = "itis")

tax_rank(get_boldid("Helianthus annuus"))
tax_rank("421377", db = "bold")
tax_rank(421377, db = "bold")

tax_rank(c("Plantae", "Helianthus annuus",
  "Puma", "Homo sapiens"), db = 'itis')
tax_rank(c("Helianthus annuus", "Quercus", "Fabaceae"), db = 'tropicos')

tax_rank(names_list("species"), db = 'gbif')
tax_rank(names_list("family"), db = 'gbif')

tax_rank(c("Gadus morhua", "Lichenopora neapolitana"),
  db = "worms")

## End(Not run)
```

`theplantlist` *Lookup-table for family, genus, and species names for ThePlantList*

Description

These names are from http://www.theplantlist.org, and are from version 1.1 of their data. This data is used in the function `names_list()`. This is a randomly selected subset of the ~350K accepted species names in Theplantlist.

Format

A data frame with 10,000 rows and 3 variables:

- `family` family name
- `genus` genus name
- `species` specific epithet name

Source

http://www.theplantlist.org

`tol_resolve` *Resolve names using Open Tree of Life (OTL) resolver*

Description

Resolve names using Open Tree of Life (OTL) resolver

Usage

```
tol_resolve(
  names = NULL,
  context_name = NULL,
  do_approximate_matching = TRUE,
  ids = NULL,
  include_suppressed = FALSE,
  ...
)
```

Arguments

`names`	(character) taxon names to be queried
`context_name`	name of the taxonomic context to be searched (length-one character vector). Must match (case sensitive) one of the values returned by `rotl::tnrs_contexts()`.
`do_approximate_matching`	(logical) A logical indicating whether or not to perform approximate string (a.k.a. "fuzzy") matching. Using `FALSE` will greatly improve speed. Default: `TRUE`
`ids`	An array of OTL ids to use for identifying names. These will be assigned to each name in the names array. If ids is provided, then ids and names must be identical in length.
`include_suppressed`	(logical) Ordinarily, some quasi-taxa, such as incertae sedis buckets and other non-OTUs, are suppressed from TNRS results. If this parameter is true, these quasi-taxa are allowed as possible TNRS results. Default: `FALSE`
`...`	Curl options passed on to `httr::POST` within `rotl::tnrs_match_names()`

Value

A data frame summarizing the results of the query. The original query output is appended as an attribute to the returned object (and can be obtained using `attr(object, "original_response")`).

Author(s)

Francois Michonneau <`francois.michonneau@gmail.com`> Scott Chamberlain

References

https://github.com/OpenTreeOfLife/germinator/wiki/TNRS-API-v3#match_names

See Also

`gnr_resolve()`, `tnrs()`

Examples

```
## Not run:
tol_resolve(names=c("echinodermata", "xenacoelomorpha",
  "chordata", "hemichordata"))
tol_resolve(c("Hyla", "Salmo", "Diadema", "Nautilus"))
tol_resolve(c("Hyla", "Salmo", "Diadema", "Nautilus"),
  context_name = "Animals")

turducken_spp <- c("Meleagris gallopavo", "Anas platyrhynchos",
  "Gallus gallus")
tol_resolve(turducken_spp, context_name="Animals")

## End(Not run)
```

`tpl_families` *Get The Plant List families.*

Description

Get The Plant List families.

Usage

```
tpl_families(...)
```

Arguments

`...`	(list) Curl options passed on to crul::verb-GET

Details

Requires an internet connection in order to connect to <www.theplantlist.org>.

Value

Returns a `data.frame` including the names of all families indexed by The Plant List, and the major groups into which they fall (i.e. Angiosperms, Gymnosperms, Bryophytes and Pteridophytes).

Author(s)

John Baumgartner (johnbb@student.unimelb.edu.au)

See Also

`tpl_get()`

Examples

```
## Not run:
# Get a data.frame of plant families, with the group name
# (Angiosperms, etc.)
head(tpl_families())

## End(Not run)
```

`tpl_get` *Get The Plant List csv files.*

Description

Get The Plant List csv files.

Usage

```
tpl_get(x, family = NULL, ...)
```

Arguments

`x`	Directory to write csv files to.
`family`	If you want just one, or >1 family, but not all, list them in a vector.
`...`	(list) Curl options passed on to crul::verb-GET

Details

Throws a warning if you already have a directory of the one provided, but still works. Writes to your home directory, change x as needed.

Value

Returns nothing to console, except a message and progress bar. Writes csv files to x.

Author(s)

John Baumgartner `<johnbb@student.unimelb.edu.au>`

References

The Plant List http://www.theplantlist.org

See Also

`tpl_families()`

Examples

```
## Not run:
# Get a few families
dir <- file.path(tempdir(), "abc")
tpl_get(dir, family = c("Platanaceae","Winteraceae"))
readLines(file.path(dir, "Platanaceae.csv"), n = 5)

# You can now get Gymnosperms as well
dir1 <- file.path(tempdir(), "def")
```

```
tpl_get(dir1, family = c("Pinaceae","Taxaceae"))

# You can get mosses too!
dir2 <- file.path(tempdir(), "ghi")
tpl_get(dir2, family = "Echinodiaceae")

# Get all families
## Beware, will take a while
## dir3 <- file.path(tempdir(), "jkl")
## tpl_get("dir3)

## End(Not run)
```

`tpl_search` *A light wrapper around the taxonstand fxn to call Theplantlist.org database.*

Description

THIS FUNCTION IS DEFUNCT.

Usage

```
tpl_search()
```

`tp_accnames` *Return all accepted names for a taxon name with a given id.*

Description

Return all accepted names for a taxon name with a given id.

Usage

```
tp_accnames(id, key = NULL, ...)
```

Arguments

`id`	the taxon identifier code
`key`	Your Tropicos API key; See taxize-authentication for help on authentication
`...`	Curl options passed on to crul::verb-GET

Value

List or dataframe.

Examples

```
## Not run:
tp_accnames(id = 25503923)
tp_accnames(id = 25538750)

# No accepted names found
tp_accnames(id = 25509881)

## End(Not run)
```

`tp_dist` *Return all distribution records for for a taxon name with a given id.*

Description

Return all distribution records for for a taxon name with a given id.

Usage

```
tp_dist(id, key = NULL, ...)
```

Arguments

`id`	the taxon identifier code
`key`	Your Tropicos API key; See taxize-authentication for help on authentication
`...`	Curl options passed on to crul::HttpClient

Value

List of two data.frame's, one named "location", and one "reference".

References

http://services.tropicos.org/help?method=GetNameDistributionsXml

Examples

```
## Not run:
# Query using a taxon name Id
out <- tp_dist(id = 25509881)
## just location data
head(out[['location']])
## just reference data
head(out[['reference']])

## End(Not run)
```

`tp_refs` *Return all reference records for for a taxon name with a given id.*

Description

Return all reference records for for a taxon name with a given id.

Usage

```
tp_refs(id, key = NULL, ...)
```

Arguments

`id`	the taxon identifier code
`key`	Your Tropicos API key; See taxize-authentication for help on authentication
`...`	Curl options passed on to crul::HttpClient

Value

List or dataframe.

Examples

```
## Not run:
tp_refs(id = 25509881)

## End(Not run)
```

`tp_search` *Search Tropicos by scientific name, common name, or Tropicos ID.*

Description

Search Tropicos by scientific name, common name, or Tropicos ID.

Usage

```
tp_search(
  sci = NULL,
  com = NULL,
  nameid = NULL,
  orderby = NULL,
  sortorder = NULL,
  pagesize = NULL,
  startrow = NULL,
```

```
    type = NULL,
    key = NULL,
    name = NULL,
    commonname = NULL,
    ...
  )
```

Arguments

`sci`	A scientific name, e.g., "poa annua". See Details.
`com`	A common name, e.g., "annual blue grass"
`nameid`	Your search string. e.g., "25509881"
`orderby`	Your search string. e.g., "1"
`sortorder`	Your search string. e.g., "ascending"
`pagesize`	Your search string. e.g., "100"
`startrow`	Your search string. e.g., "1"
`type`	Type of search, "wildcard" (default) will add a wildcard to the end of your search string. "exact" will use your search string exactly.
`key`	Your Tropicos API key; See taxize-authentication for help on authentication
`name`	Deprecated, see `sci`
`commonname`	Deprecated, see `com`
`...`	Further args passed on to crul::HttpClient

Details

More details on the `name` parameter: Tropicos will fail if you include a period (`.`) in your name string, e.g., `var.`, so we replace periods before the request is made to the Tropicos web service. In addition, Tropicos for some reason doesn't want to see sub-specific rank names like `var`/`subsp`, so remove those from your query.

Value

List or dataframe.

References

http://services.tropicos.org/help?method=SearchNameXml

Examples

```
## Not run:
tp_search(sci = 'Poa annua')
tp_search(sci = 'Poa annua subsp. annua')
tp_search(sci = 'Poa annua var. annua')
tp_search(sci = 'Poa annua var annua')
tp_search(sci = 'Poa annua annua')

## End(Not run)
```

`tp_summary` *Return summary data a taxon name with a given id.*

Description

Return summary data a taxon name with a given id.

Usage

```
tp_summary(id, key = NULL, ...)
```

Arguments

`id`	the taxon identifier code
`key`	Your Tropicos API key; See taxize-authentication for help on authentication
`...`	Curl options passed on to crul::verb-GET

Value

A data.frame.

Examples

```
## Not run:
tp_summary(id = 25509881)
tp_summary(id = 2700851)
tp_summary(id = 24900183)

## End(Not run)
```

`tp_synonyms` *Return all synonyms for a taxon name with a given id.*

Description

Return all synonyms for a taxon name with a given id.

Usage

```
tp_synonyms(id, key = NULL, ...)
```

Arguments

`id`	the taxon identifier code
`key`	Your Tropicos API key; See taxize-authentication for help on authentication
`...`	Curl options passed on to crul::HttpClient

Value

List or dataframe.

Examples

```
## Not run:
tp_synonyms(id = 25509881)

## End(Not run)
```

`ubio_ping` *uBio ping*

Description

uBio ping

Usage

```
ubio_ping()
```

`upstream` *Retrieve the upstream taxa for a given taxon name or ID.*

Description

This function uses a while loop to continually collect taxa up to the taxonomic rank that you specify in the `upto` parameter. You can get data from ITIS (itis) only currently. There is no method exposed by itis for getting taxa at a specific taxonomic rank, so we do it ourselves inside the function.

Usage

```
upstream(...)

## Default S3 method:
upstream(sci_id, db = NULL, upto = NULL, rows = NA, x = NULL, ...)

## S3 method for class 'tsn'
upstream(sci_id, db = NULL, upto = NULL, ...)

## S3 method for class 'ids'
upstream(sci_id, db = NULL, upto = NULL, ...)
```

Arguments

`...`	Further args passed on to `itis_downstream()`
`sci_id`	Vector of taxa names (character) or IDs (character or numeric) to query.
`db`	character; database to query. One or both of `itis`. Note that each taxonomic data source has their own identifiers, so that if you provide the wrong `db` value for the identifier you could get a result, but it will likely be wrong (not what you were expecting).
`upto`	What taxonomic rank to go down to. One of: 'superkingdom', 'kingdom', 'subkingdom','infrakingdom','phylum','division','subphylum', 'subdivision','infradivision', 'superclass','class','subclass','infraclass', 'superorder','order','suborder','infraorder','superfamily','famil 'subfamily','tribe','subtribe','genus','subgenus', 'section','subsection', 'species','subspecies','variety','fo 'stirp', 'morph','aberration','subform', or 'unspecified'
`rows`	(numeric) Any number from 1 to infinity. If the default NA, all rows are considered. Note that this parameter is ignored if you pass in a taxonomic id of any of the acceptable classes: tsn.
`x`	Deprecated, see `sci_id`

Value

A named list of data.frames with the upstream names of every supplied taxa. You get an NA if there was no match in the database.

Examples

```
## Not run:
upstream('Pinus contorta', db = 'itis', upto = 'genus')

## End(Not run)
```

`vascan_search` *Search the CANADENSYS Vascan API.*

Description

Search the CANADENSYS Vascan API.

Usage

```
vascan_search(q, format = "json", raw = FALSE, ...)
```

Arguments

`q`	(character) Can be a scientific name, a vernacular name or a VASCAN taxon identifier (e.g. 861)
`format`	(character) One of json (default) or xml.
`raw`	(logical) If TRUE, raw json or xml returned, if FALSE, parsed data returned.
`...`	(list) Further args passed on to crul::verb-GET

Details

Note that we lowercase all outputs in data.frame's, but when a list is given back, we don't touch the list names.

Value

json, xml or a list.

Author(s)

Scott Chamberlain

References

API docs https://data.canadensys.net/vascan/api

Examples

```
## Not run:
vascan_search(q = "Helianthus annuus")
vascan_search(q = "Helianthus annuus", raw=TRUE)
vascan_search(q = c("Helianthus annuus", "Crataegus dodgei"), raw=TRUE)

# format type
## json
c <- vascan_search(q = "Helianthus annuus", format="json", raw=TRUE)
library("jsonlite")
fromJSON(c, FALSE)

## xml
d <- vascan_search(q = "Helianthus annuus", format="xml", raw=TRUE)
library("xml2")
xml2::read_xml(d)

# lots of names, in this case 50
splist <- names_list(rank='species', size=50)
vascan_search(q = splist)

# Curl options
invisible(vascan_search(q = "Helianthus annuus", verbose = TRUE))

## End(Not run)
```

`worms_downstream` *Retrieve all taxa names downstream in hierarchy for WORMS*

Description

Retrieve all taxa names downstream in hierarchy for WORMS

Usage

```
worms_downstream(id, downto, intermediate = FALSE, start = 1, ...)
```

Arguments

`id`	(integer) One or more AphiaID's
`downto`	(character) The taxonomic level you want to go down to. See examples below. The taxonomic level IS case sensitive, and you do have to spell it correctly. See rank_ref_zoo for spelling.
`intermediate`	(logical) If `TRUE`, return a list of length two with target taxon rank names, with additional list of data.frame's of intermediate taxonomic groups. Default: `FALSE`
`start`	(integer) Record number to start at
`...`	crul options passed on to `worrms::wm_children()`, including the parameters `marine_only` and `offset`, see `?worrms::wm_children` for details

Value

data.frame of taxonomic information downstream to family from e.g., Order, Class, etc., or if `intermediated=TRUE`, list of length two, with target taxon rank names, and intermediate names.

Examples

```
## Not run:
## the genus Gadus
worms_downstream(id = 125732, downto="species")
worms_downstream(id = 125732, downto="species", intermediate=TRUE)

worms_downstream(id = 51, downto="class")
worms_downstream(id = 51, downto="subclass", intermediate=TRUE)

worms_downstream(id = 105, downto="subclass")

# marine_only parameter
worms_downstream(545470, downto = "species")
worms_downstream(545470, downto = "species", marine_only = FALSE)

## End(Not run)
```

`worrms_ranks` *WORMS ranks*

Description

Created using `worrms::wm_ranks_id(-1)` on 2020-02-11.

Format

A data frame with 216 rows and 2 variables:

- `id`: rank id
- `rank`: rank name

Details

Present in taxize in the case where WORMS does not return rank names - with this dataset we can fill in rank information as long as rank ids are returned

Index