Package ‘taxize’

February 20, 2015

Title  Taxonomic Information from Around the Web

Description  Taxonomic information from around the web. This package interacts with a suite of web APIs for taxonomic tasks, such as verifying species names, getting taxonomic hierarchies, and verifying name spelling.

Version  0.5.2

Date  2014-12-18

License  MIT + file LICENSE

URL  https://github.com/ropensci/taxize

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

LazyLoad  yes

LazyData  yes

VignetteBuilder  knitr

Depends  R(>= 2.10.0)

Imports  XML, RCurl (>= 1.6), stringr, plyr, httr (>= 0.2), jsonlite, foreach, ape, Taxonstand, reshape2, data.table, vegan, bold

Suggests  testthat, roxygen2, knitr

Author  Scott Chamberlain [aut, cre], Eduard Szucs [aut], Zachary Foster [aut], Carl Boettiger [aut], Karthik Ram [aut], Ignasi Bartomeus [aut], John Baumgartner [aut]

Maintainer  Scott Chamberlain <myrmecocystus@gmail.com>

NeedsCompilation  no

Repository  CRAN

Date/Publication  2014-12-19 07:37:14
R topics documented:

taxize-package .................................................. 4
apg_families .................................................. 5
apg_genera .................................................. 5
apg_lookup .................................................. 6
apg_orders .................................................. 6
bold_search .................................................. 7
children .................................................. 8
class2tree .................................................. 10
classification .................................................. 11
col_children .................................................. 15
col_downstream ................................................ 16
col_search .................................................. 17
comm2sci .................................................. 19
downstream .................................................. 20
eol_dataobjects ................................................ 22
eol_invasive ................................................ 23
eol_pages .................................................. 25
eol_search .................................................. 26
gbif_name_usage ................................................ 27
gbiz_parse ................................................ 28
genbank2uid ................................................ 29
get_boldid ................................................ 30
get_colid ................................................ 32
get_eolid ................................................ 35
get_gbifid ................................................ 37
get_ids ................................................ 39
get_nbnid ................................................ 41
get_tpsid ................................................ 43
get_tsn ................................................ 45
get_ubioid ................................................ 48
get_uid ................................................ 50
gisd_isinvasive ............................................... 52
gni_details ................................................ 53
gni_parse ................................................ 54
gni_search ................................................ 55
gnr_datasources .............................................. 57
gnr_resolve ................................................ 58
iplant_resolve ................................................ 59
ipni_search ................................................ 60
itis-api ................................................ 62
itis_acceptname ............................................... 64
itis_downstream ............................................... 64
itis_getrecord ............................................... 66
itis_hierarchy ............................................... 66
itis_kingdomnames .......................................... 67
itis_lsid ................................................ 68
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>itis_name</td>
<td>69</td>
</tr>
<tr>
<td>itis_native</td>
<td>69</td>
</tr>
<tr>
<td>itis_refs</td>
<td>70</td>
</tr>
<tr>
<td>itis_searchcommon</td>
<td>71</td>
</tr>
<tr>
<td>itis_taxrank</td>
<td>71</td>
</tr>
<tr>
<td>itis_terms</td>
<td>72</td>
</tr>
<tr>
<td>iucn_getname</td>
<td>73</td>
</tr>
<tr>
<td>iucn_status</td>
<td>74</td>
</tr>
<tr>
<td>iucn_summary</td>
<td>74</td>
</tr>
<tr>
<td>names_list</td>
<td>76</td>
</tr>
<tr>
<td>nbn_classification</td>
<td>77</td>
</tr>
<tr>
<td>nbn_search</td>
<td>77</td>
</tr>
<tr>
<td>nbn_synonyms</td>
<td>78</td>
</tr>
<tr>
<td>ncbi_children</td>
<td>79</td>
</tr>
<tr>
<td>ncbi_getbyid</td>
<td>80</td>
</tr>
<tr>
<td>ncbi_getbyname</td>
<td>81</td>
</tr>
<tr>
<td>ncbi_get_taxon_summary</td>
<td>82</td>
</tr>
<tr>
<td>ncbi_search</td>
<td>83</td>
</tr>
<tr>
<td>phylomatic_format</td>
<td>85</td>
</tr>
<tr>
<td>phylomatic_tree</td>
<td>86</td>
</tr>
<tr>
<td>ping</td>
<td>87</td>
</tr>
<tr>
<td>plantGenusNames</td>
<td>89</td>
</tr>
<tr>
<td>plantminer</td>
<td>89</td>
</tr>
<tr>
<td>plantNames</td>
<td>90</td>
</tr>
<tr>
<td>rankagg</td>
<td>90</td>
</tr>
<tr>
<td>rank_ref</td>
<td>91</td>
</tr>
<tr>
<td>resolve</td>
<td>91</td>
</tr>
<tr>
<td>sci2comm</td>
<td>92</td>
</tr>
<tr>
<td>scrapenames</td>
<td>93</td>
</tr>
<tr>
<td>status_codes</td>
<td>95</td>
</tr>
<tr>
<td>synonyms</td>
<td>95</td>
</tr>
<tr>
<td>taxize-defunct</td>
<td>97</td>
</tr>
<tr>
<td>taxize-deprecated</td>
<td>97</td>
</tr>
<tr>
<td>taxize_capwords</td>
<td>98</td>
</tr>
<tr>
<td>taxize_cite</td>
<td>99</td>
</tr>
<tr>
<td>tax_agg</td>
<td>99</td>
</tr>
<tr>
<td>tax_name</td>
<td>101</td>
</tr>
<tr>
<td>tax_rank</td>
<td>102</td>
</tr>
<tr>
<td>theplantlist</td>
<td>103</td>
</tr>
<tr>
<td>tnrs</td>
<td>103</td>
</tr>
<tr>
<td>tnrs_sources</td>
<td>105</td>
</tr>
<tr>
<td>tpl_families</td>
<td>106</td>
</tr>
<tr>
<td>tpl_get</td>
<td>106</td>
</tr>
<tr>
<td>tpl_search</td>
<td>108</td>
</tr>
<tr>
<td>tp_accnames</td>
<td>109</td>
</tr>
<tr>
<td>tp_dist</td>
<td>109</td>
</tr>
<tr>
<td>tp.refs</td>
<td>110</td>
</tr>
<tr>
<td>tp_search</td>
<td>111</td>
</tr>
</tbody>
</table>
taxize-package

Taxonomic information from around the web.

**Description**

Taxonomic search and phylogeny retrieval.

**Details**

We are developing taxize as a package to allow 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 Encyclopedia of Life (EOL), the Universal Biological Indexer and Organizer (uBio), Tropicos, and Plantminer.

Currently supported APIs:

<table>
<thead>
<tr>
<th>API</th>
<th>prefix</th>
<th>SOAP?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encyclopedia of Life (EOL)</td>
<td>eol</td>
<td>FALSE</td>
</tr>
<tr>
<td>Integrated Taxonomic Information Service (ITIS)</td>
<td>itis</td>
<td>FALSE</td>
</tr>
<tr>
<td>Phylomatic</td>
<td>phylomatic</td>
<td>FALSE</td>
</tr>
<tr>
<td>uBio</td>
<td>ubio</td>
<td>FALSE</td>
</tr>
<tr>
<td>Global Names Resolver (from EOL/GBIF)</td>
<td>gnr</td>
<td>FALSE</td>
</tr>
<tr>
<td>Global Names Index (from EOL/GBIF)</td>
<td>gni</td>
<td>FALSE</td>
</tr>
<tr>
<td>IUCN Red List</td>
<td>iucn</td>
<td>FALSE</td>
</tr>
<tr>
<td>Tropicos (from Missouri Botanical Garden)</td>
<td>tp</td>
<td>FALSE</td>
</tr>
<tr>
<td>Plantminer</td>
<td>plantminer</td>
<td>FALSE</td>
</tr>
<tr>
<td>Theplantlist.org</td>
<td>tpl</td>
<td>FALSE</td>
</tr>
<tr>
<td>Catalogue of Life</td>
<td>col</td>
<td>FALSE</td>
</tr>
<tr>
<td>Global Invasive Species Database</td>
<td>gisd</td>
<td>FALSE</td>
</tr>
<tr>
<td>National Center for Biotechnology Information</td>
<td>ncbi</td>
<td>FALSE</td>
</tr>
<tr>
<td>Source</td>
<td>Description</td>
<td>SOAP?</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>---------------</td>
<td>-------</td>
</tr>
<tr>
<td>CANADENSYS Vascan name search API</td>
<td>vascan</td>
<td>FALSE</td>
</tr>
<tr>
<td>International Plant Names Index (IPNI)</td>
<td>ipni</td>
<td>FALSE</td>
</tr>
<tr>
<td>World Register of Marine Species (WoRMS)</td>
<td>worms</td>
<td>TRUE</td>
</tr>
<tr>
<td>Barcode of Life Data Systems (BOLD)</td>
<td>bold</td>
<td>FALSE</td>
</tr>
<tr>
<td>Pan-European Species directories Infrastructure (PESI)</td>
<td>pesi</td>
<td>TRUE</td>
</tr>
<tr>
<td>Mycobank</td>
<td>myco</td>
<td>TRUE</td>
</tr>
<tr>
<td>National Biodiversity Network (UK)</td>
<td>nbn</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

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

**Author(s)**

Scott Chamberlain <myrmecocystus@gmail.com>
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>

---

**apg_families**

Lookup-table for APGIII family names

**Description**

Lookup-table for APGIII family names

---

**apg_genera**

Table of APGIII generic names

**Description**

Table of APGIII generic names
**apg_lookup**  
*Lookup in the APGII taxonomy and replace family names.*

**Description**

Lookup in the APGII taxonomy and replace family names.

**Usage**

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

**Arguments**

- `taxa`  
  Taxonomic name to lookup a synonym for in APGII taxonomy.

- `rank`  
  Taxonomic rank to lookup a synonym for. One of family or order.

**Value**

A APGII family or order name, or the original name if no match.

**Examples**

```r
# New name found in the APGII taxonomy
apg_lookup(taxa = "Hyacinthaceae", rank = "family")
apg_lookup(taxa = "Poaceae", rank = "family")

# Name not found in the APGII taxonomy
apg_lookup(taxa = "Asteraceae", rank = "family")
```

---

**apg_orders**  
*Lookup-table for APGIII order names*

**Description**

Lookup-table for APGIII order names
**bold_search**  

**Search Barcode of Life for taxonomic IDs**

**Description**

Search Barcode of Life for taxonomic IDs

**Usage**

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

**Arguments**

- **name**  
  (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.

- **...**  
  Further args passed on to `GET`, main purpose being curl debugging

**Details**

You must provide one of name or id to this function. The other parameters are optional. Note that when passing in name, 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(name="Apis")
bold_search(name="Agapostemon")
bold_search(name="Poa")

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

# Many names
bold_search(name=c("Apis","Puma concolor"))
nms <- names_list('species')
bold_search(name=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(x, db = NULL, ...)

## Not run:
bold_search(name="Apis")
bold_search(name="Agapostemon")
bold_search(name="Poa")

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

# Many names
bold_search(name=c("Apis","Puma concolor"))
nms <- names_list('species')
bold_search(name=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)
## S3 method for class 'tsn'

children(x, db = NULL, ...)

## S3 method for class 'colid'

children(x, db = NULL, ...)

## S3 method for class 'ids'

children(x, db = NULL, ...)

## S3 method for class 'uid'

children(x, db = NULL, ...)

### Arguments

... Further args passed on to `col_children`, `gethierarchydownfromtsn`, or `ncbi_children`. See those functions for what parameters can be passed on.

- **x** character; taxons to query.
- **db** character; database to query. One or more of itis, col, or ncbi.

### 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.

### Examples

```r
## Not run:
# Plug in taxon names
children("Salmo", db = 'col')
children("Salmo", db = 'itis')
children("Salmo", db = 'ncbi')

# Plug in IDs
(id <- get_colid("Apis"))
children(id)

## Equivalently, plug in the call to get the id via e.g., get_colid into children
identical(children(id), children(get_colid("Apis")))

(id <- get_colid("Apis"))
children(id)
children(get_colid("Apis"))

# Many taxa
(sp <- names_list("genus", 3))
children(sp, db = 'col')
children(sp, db = 'itis')

# Two data sources
(ids <- get_ids("Apis", db = c('col','itis'))) children(ids)
```
class2tree

Convert 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, ...)

## 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.

... Further arguments passed on to hclust.

x Input object to print or plot - output from class2tree function.

Details

See taxa2dist. Thanks to Jari Oksanen for making the taxa2dist function and pointing it out, and Clarke & Warwick (1998, 2001), which taxa2dist was based on.
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 from a call to `taxa2dist`
- 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

```r
# Not run
spnames <- c('Klattia flava', 'Trollius sibiricus', 'Arachis paraguariensis',
             'Tanacetum boreale', 'Gentiana yakushimensis', 'Sesamum schinziianum',
             'Pilea verrucosa', 'Tibouchina striphnocalyx', 'Lycium dasystemum',
             'Schoenus centralis', 'Berkheya echinacea', 'Androcymbium villosum',
             'Helenium annuus', 'Madia elegans', 'Lupinus albicaulis', 'Poa annua',
             'Pinus lambertiana')
out <- classification(spnames, db='ncbi')
tr <- class2tree(out)
plot(tr)

# another example using random sets of names with names_list() fxn
spnames <- names_list('species', 50)
out <- classification(spnames, db='ncbi')
tr <- class2tree(out)
plot(tr)
plot(tr, no.margin=TRUE)
```

## End(Not run)
## S3 method for class 'tsn'
classification(id, callopts = list(), return_id = TRUE, ...)

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

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

## S3 method for class 'colid'
classification(id, start = NULL, checklist = NULL,
  return_id = TRUE, ...)

## S3 method for class 'tpsid'
classification(id, key = NULL, 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 'ids'
classification(id, ...)

cbind.classification(x)

rbind.classification(x)

cbind.classification_ids(...)

rbind.classification_ids(...)

### Arguments

- ... Other arguments passed to `get_tsn`, `get_uid`, `get_eolid`, `get_colid`, `get_tpsid`, `get_gbifid`.
- `x` character; taxons to query.
- `db` character; database to query. either `ncbi`, `itis`, `eol`, `col`, `tropicos`, `gbif`, or `nbn`.
- `callopts` Curl options passed on to `GET`.
- `return_id` (logical) If TRUE (default), return the taxon id as well as the name and rank of taxa in the lineage returned.
id character; identifiers, returned by get_tsn, get_uid, get_eolid, get_colid, get_tpsid, get_gbifid.

key Your API key; loads from .Rprofile.

start The first record to return. If omitted, the results are returned from the first record (start=0). This is useful if the total number of results is larger than the maximum number of results returned by a single Web service query (currently the maximum number of results returned by a single query is 500 for terse queries and 50 for full queries).

checklist character; The year of the checklist to query, if you want a specific year’s checklist instead of the lastest as default (numeric).

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.

See Also

get_tsn, get_uid, get_eolid, get_colid, get_tpsid, get_gbifid

Examples

## Not run:
# Plug in taxon names directly
classification(c("Chironomus riparius", "aaa vva"), db = 'ncbi')
classification(c("Chironomus riparius", "aaa vva"), db = 'ncbi', verbose=FALSE)
classification(c("Chironomus riparius", "aaa vva"), db = 'itis')
classification(c("Chironomus riparius", "aaa vva"), db = 'itis', verbose=FALSE)
classification(c("Chironomus riparius", "aaa vva"), db = 'eol')
classification(c("Chironomus riparius", "aaa vva"), db = 'col')
classification("Alopia vulpinus", db = 'nbn')
classification("Chironomus riparius", "aaa vva", db = 'col', verbose=FALSE)
classification("Chironomus riparius", "asdfsdfsdfsdfsd", db = 'gbif')
classification("Poa annua", db = 'tropicos')

# Use methods for get_uid, get_tsn, get_eolid, get_colid, get_tpsid
classification(get_uid(c("Chironomus riparius", "Puma concolor")))
classification(get_tsn(c("Chironomus riparius", "aaa vva")))
classification(get_tsn(c("Chironomus riparius", "aaa vva")), verbose = FALSE)
classification(get_eolid(c("Chironomus riparius", "aaa vva")))
classification(get_colid(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(names="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(names=c("Puma concolor","Accipiter striatus"), db = c('ncbi','itis','col')))  
(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) )

# rbind and cbind on class classification (from a call to get_colid, get_tsn, etc.  
# - other than get_ids)
(cl_col <- classification(get_colid(c("Puma concolor","Accipiter striatus"))))  
rbind(cl_col)
cbind(cl_col)

(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))
rbind(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)

## End(Not run)
col_children

Search Catalogue of Life for direct children of a particular taxon.

Description

Search Catalogue of Life for direct children of a particular taxon.

Usage

col_children(name = NULL, id = NULL, format = NULL, start = NULL, checklist = NULL)

Arguments

name The string to search for. Only exact matches found the name given will be returned, unless one or wildcards are included in the search string. An * (asterisk) character denotes a wildcard; a character may also be used. The name must be at least 3 characters long, not counting wildcard characters.

id The record ID of the specific record to return (only for scientific names of species or infraspecific taxa)

format format of the results returned. Valid values are format=xml and format=php; if the format parameter is omitted, the results are returned in the default XML format. If format=php then results are returned as a PHP array in serialized string format, which can be converted back to an array in PHP using the unserialize command

start The first record to return. If omitted, the results are returned from the first record (start=0). This is useful if the total number of results is larger than the maximum number of results returned by a single Web service query (currently the maximum number of results returned by a single query is 500 for terse queries and 50 for full queries).

checklist The year of the checklist to query, if you want a specific year's checklist instead of the lastest as default (numeric).

Details

You must provide one of name or id. The other parameters (format and start) are optional.

Value

A list of data.frame’s.

Examples

## Not run:
# A basic example
col_children(name = "Apis")
# An example where there is no classification, results in data.frame with no rows
col_children(id=11935941)

# Use a specific year's checklist
col_children(name="Apis", checklist="2012")
col_children(name="Apis", checklist="2009")

# Pass in many names or many id's
out <- col_children(name=c("Buteo","Apis","Accipiter","asdf"), checklist="2012")
out$Apis # get just the output you want
library(plyr)
ldply(out) # or combine to one data.frame

# or pass many id's
out <- col_children(id=c(2346405,2344165,2346405), checklist="2012")
library(plyr)
ldply(out) # combine to one data.frame

## End(Not run)

---

**col_downstream**

*Use Catalogue of Life to get downstream taxa to a given taxonomic level.*

### Description

Use Catalogue of Life to get downstream taxa to a given taxonomic level.

### Usage

```r
col_downstream(name = NULL, id = NULL, downto, format = NULL,
               start = NULL, checklist = NULL, verbose = TRUE, intermediate = FALSE)
```

### Arguments

- **name**: The string to search for. Only exact matches found the name given will be returned, unless one or wildcards are included in the search string. An * (asterisk) character denotes a wildcard; a character may also be used. The name must be at least 3 characters long, not counting wildcard characters.
- **id**: The record ID of the specific record to return (only for scientific names of species or infraspecific taxa)
- **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.
- **format**: The returned format (default = NULL). If NULL xml is used. Currently only xml is supported.
col_search

The first record to return (default = NULL). If NULL, the results are returned from the first record (start=0). This is useful if the total number of results is larger than the maximum number of results returned by a single Web service query (currently the maximum number of results returned by a single query is 500 for terse queries and 50 for full queries).

checklist

The year of the checklist to query, if you want a specific year’s checklist instead of the lastest as default (numeric).

verbose

Print or suppress messages.

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

Details

Provide only names instead of id’s

Value

A list of data.frame’s.

Examples

```r
## Not run:
# Some basic examples
col_downstream(name="Apis", downto="Species")
col_downstream(name="Bryophyta", downto="Family")

col_downstream(name="Animalia", downto="Class")
col_downstream(name="Animalia", downto="Class", intermediate=TRUE)

# An example that takes a bit longer
col_downstream(name=c("Plantae","Animalia"), downto="Class")

# Using a checklist from a specific year
col_downstream(name="Bryophyta", downto="Family", checklist=2009)

# By id
col_downstream(id=2346405, downto="Genus", checklist=2012)

## End(Not run)
```

---

**col_search**  
*Search Catalogue of Life for taxonomic IDs*

**Description**  
Search Catalogue of Life for taxonomic IDs
Usage

col_search(name = NULL, id = NULL, start = NULL, checklist = NULL)

Arguments

name  The string to search for. Only exact matches found the name given will be returned, unless one or wildcards are included in the search string. An * (asterisk) character denotes a wildcard; a character may also be used. The name must be at least 3 characters long, not counting wildcard characters.

id  The record ID of the specific record to return (only for scientific names of species or infraspecific taxa)

start  The first record to return. If omitted, the results are returned from the first record (start=0). This is useful if the total number of results is larger than the maximum number of results returned by a single Web service query (currently the maximum number of results returned by a single query is 500 for terse queries and 50 for full queries).

checklist  The year of the checklist to query, if you want a specific year’s checklist instead of the lastest as default (numeric).

Details

You must provide one of name or id. The other parameters (format and start) are optional.

Value

A list of data.frame’s.

References

http://webservice.catalogueoflife.org/

Examples

```r
## Not run:
# A basic example
col_search(name="Apis")
col_search(name="Agapostemon")
col_search(name="Poa")

# Many names
col_search(name=c("Apis","Puma concolor"))

# An example where there is no data
col_search(id=11935941)

## End(Not run)
```
Get scientific names from common names.

Description

Get scientific names from common names.

Usage

```r
comm2sci(commnames, db = "eol", itisby = "search", simplify = TRUE, ...)
```

Arguments

- `commnames`: One or more common names or partial names.
- `db`: Data source, one of "eol" (default), "itis", "tropicos" or "ncbi".
- `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.
- `...`: Further arguments passed on to internal methods.

Value

A vector of names.

Author(s)

Scott Chamberlain (myrmecocystus@gmail.com)

See Also

- `searchbycommonname`
- `searchbycommonnamebeginswith`
- `searchbycommonnameendswith`
- `eol_search`
- `tp_search`
- `sci2comm`

Examples

```r
# Not run:
comm2sci(commnames='black bear')
comm2sci(commnames='black bear', db='itis')
comm2sci(commnames='annual blue grass', db='tropicos')
comm2sci(commnames=c('annual blue grass','tree of heaven'), db='tropicos')
comm2sci(commnames=c('black bear', 'roe deer'))

# Output easily converts to a data.frame with `plyr`'s ldply)
library(plyr)
ldply(comm2sci(commnames=c('annual blue grass','tree of heaven'), db='tropicos'))
```
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) or Catalogue of Life (col). There is no method exposed by itis or col for getting taxa at a specific taxonomic rank, so we do it ourselves inside the function.

Usage

downstream(...)

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

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

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

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

Arguments

... Further args passed on to itis_downstream or col_downstream
x character; taxons to query.
db character; database to query. One or both of itis, col.
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
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.

Examples

```r
## Not run:
# Plug in taxon names
downstream("Insecta", db = 'col', downto = 'Order')
downstream("Apis", db = 'col', downto = 'Species')
downstream("Apis", db = 'itis', downto = 'Species')
downstream(c("Apis","Epeoloides"), db = 'itis', downto = 'Species')
downstream(c("Apis","Epeoloides"), db = 'col', downto = 'Species')

# Plug in IDs
id <- get_colid("Apis")
downstream(id, downto = 'Species')

## Equivalently, plug in the call to get the id via e.g., get_colid into downstream
identical(downstream(id, downto = 'Species'),
          downstream(get_colid("Apis"), downto = 'Species'))

id <- get_colid("Apis")
downstream(id, downto = 'Species')
downstream(get_colid("Apis"), downto = 'Species')

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

# Both data sources
ids <- get_ids("Apis", db = c('col','itis'))
downstream(ids, downto = 'Species')

## same result
downstream(get_ids("Apis", db = c('col','itis')), downto = 'Species')

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

## col
downstream(get_colid("Animalia"), downto="Class")
downstream(get_colid("Animalia"), downto="Class", intermediate=TRUE)

## End(Not run)
```
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, returntype = "data.frame", usekey = TRUE, key = NULL, verbose = TRUE)

Arguments

id
The EOL data object identifier (character)

returntype
one of "list" of "data.frame" (character)

usekey
use your API key or not (TRUE or FALSE)

key
Your EOL API key; loads from .Rprofile.

verbose
logical; If TRUE the actual taxon queried is printed on the console.

Details

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

Value

List or dataframe (default).

Examples

```r
# Not run:
eol_dataobjects(id="d72801627bf4adfla38d9c5f10cc767f")
eol_dataobjects(id="21929584")
```

# End(Not run)
eol_invasive

Search for presence of taxonomic names in EOL invasive species databases.

Description

See Details for important information.

Usage

```r
eol_invasive(name = NULL, dataset = "all", searchby = grep, page = NULL,
  per_page = NULL, key = NULL, callopts = list(), verbose = TRUE,
  count = FALSE)
```

Arguments

- `name`: A taxonomic name, or a vector of names.
- `dataset`: One of all, gisd100, gisd, isc, daisie, i3n, or mineps. See the Details for what each dataset ID.
- `searchby`: One of `grep` (exact match) or `agrep` (fuzzy match)
- `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)
- `per_page`: Results to get per page
- `key`: Your EOL API key; loads from .Rprofile.
- `callopts`: Further args passed on to GET.
- `verbose` (logical) If TRUE the actual taxon queried is printed on the console.
- `count` (logical) If TRUE, give back a count of number of taxa listed as invasive, if FALSE (default), the normal output is given.

Details

IMPORTANT: When you get a returned NaN for a taxon, that means it’s not on the invasive list in question. If the taxon is found, a taxon identifier is returned.

Beware that some datasets are quite large, and may take 30 sec to a minute to pull down all data before we can search for your species. Note there is no parameter in this API method for searching by taxon name.

This function is vectorized, so you can pass a single name or a vector of names.

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

Options for the dataset parameter are:

- `all` - All datasets
- `gisd100` - 100 of the World’s Worst Invasive Alien Species (Global Invasive Species Database)

http://eol.org/collections/54500
• gisd - Global Invasive Species Database 2013 http://eol.org/collections/54983
• isc - Centre for Agriculture and Biosciences International Invasive Species Compendium (ISC) 
  http://eol.org/collections/55180
• daisie - Delivering Alien Invasive Species Inventories for Europe (DAISIE) Species List http://eol.org/collections/55179
• i3n - IABIN Invasives Information Network (I3N) Species http://eol.org/collections/55176
• mineps - Marine Invaders of the NE Pacific Species http://eol.org/collections/55331

Datasets are not updated that often. Here’s last updated dates for some of the datasets as of 2014-08-25

• gisd100 updated 6 mos ago
• gisd updated 1 yr ago
• isc updated 1 yr ago
• daisie updated 1 yr ago
• i3n updated 1 yr ago
• mineps updated 1 yr ago

Value

A list of data.frame’s/strings with results, with each element named by the input elements to the name parameter.

References

See info for each data source at http://eol.org/collections/55367/taxa

Examples

```r
## Not run:
eol_invasive(name='Brassica oleracea', dataset='gisd')
eol_invasive(name=c('Lymantria dispar','Cygnus olor','Hydrilla verticillata','Pinus concolor'),
  dataset='gisd')
eol_invasive(name='Sargassum', dataset='gisd')
eol_invasive(name='Ciona intestinalis', dataset='mineps')
eol_invasive(name=c('Lymantria dispar','Cygnus olor','Hydrilla verticillata','Pinus concolor'),
  dataset='i3n')
eol_invasive(name=c('Branta canadensis','Gallus gallus','Myiopsitta monachus'),
  dataset='daisie')
eol_invasive(name=c('Branta canadensis','Gallus gallus','Myiopsitta monachus'), dataset='isc')

# Count
eol_invasive(name=c('Lymantria dispar','Cygnus olor','Hydrilla verticillata','Pinus concolor'),
  dataset='gisd', count = TRUE)

## End(Not run)
```
Search for pages in EOL database using a taxonconceptID.

Description
Search for pages in EOL database using a taxonconceptID.

Usage
```
eol_pages(taxonconceptID, iucn = FALSE, images = 0, videos = 0,
sounds = 0, maps = 0, text = 0, subject = "overview",
licenses = "all", details = FALSE, common_names = FALSE,
synonyms = FALSE, references = FALSE, vetted = 0, cache_ttl = NULL,
key = NULL, callopts = list())
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taxonconceptID</td>
<td>The taxonconceptID (numeric), which is also the page number.</td>
</tr>
<tr>
<td>iucn</td>
<td>Include the IUCN Red List status object (Default: False)</td>
</tr>
<tr>
<td>images</td>
<td>Limits the number of returned image objects (values 0 - 75)</td>
</tr>
<tr>
<td>videos</td>
<td>Limits the number of returned video objects (values 0 - 75)</td>
</tr>
<tr>
<td>sounds</td>
<td>Limits the number of returned sound objects (values 0 - 75)</td>
</tr>
<tr>
<td>maps</td>
<td>Limits the number of returned map objects (values 0 - 75)</td>
</tr>
<tr>
<td>text</td>
<td>Limits the number of returned text objects (values 0 - 75)</td>
</tr>
<tr>
<td>subject</td>
<td>'overview' (default) to return the overview text (if exists), a pipe</td>
</tr>
<tr>
<td>licenses</td>
<td>A pipe</td>
</tr>
<tr>
<td>details</td>
<td>Include all metadata for data objects. (Default: False)</td>
</tr>
<tr>
<td>common_names</td>
<td>Return all common names for the page’s taxon (Default: False)</td>
</tr>
<tr>
<td>synonyms</td>
<td>Return all synonyms for the page’s taxon (Default: False)</td>
</tr>
<tr>
<td>references</td>
<td>Return all references for the page’s taxon (Default: False)</td>
</tr>
<tr>
<td>vetted</td>
<td>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)</td>
</tr>
<tr>
<td>cache_ttl</td>
<td>The number of seconds you wish to have the response cached.</td>
</tr>
<tr>
<td>key</td>
<td>Your EOL API key; loads from .Rprofile, or you can specify the key manually the in the function call.</td>
</tr>
<tr>
<td>callopts</td>
<td>Further args passed on to RCurl::getForm.</td>
</tr>
</tbody>
</table>
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

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

eol_search

Search for terms in EOL database.

Description

Search for terms in EOL database.

Usage

eol_search(terms, page = 1, exact = NULL, filter_tid = NULL,
filter_heid = NULL, filter_by_string = NULL, cache_ttl = NULL,
key = NULL, callopts = list())

Arguments

terms  search terms (character)
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.
key  Your EOL API key; loads from .Rprofile.
callopts  Curl options passed on to getForm.
gbif_name_usage

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.

Examples

## Not run:
eol_search(terms='Homo')
eol_search(terms='Salix')
eol_search(terms='Ursus americanus luteolus')

## 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, callopts = list())
```

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.
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 A character vector of scientific names.

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

http://dev.gbif.org/wiki/display/POR/Webservice+API. http://tools.gbif.org/nameparser/api.do

See Also

gni_parse

Examples

## Not run:

```r
library(gbio) # Package required

# with accession numbers
genbank2uid(id = 'AJ748748')
genbank2uid(id = 'Y13155')
genbank2uid(id = 'X78312')
genbank2uid(id = 'KM495596')
```

---

**genbank2uid**

Get NCBI taxonomy UID from GenBankID

Description

Get NCBI taxonomy UID from GenBankID

Usage

```r
genbank2uid(id, ...)
```

Arguments

- **id**
  - A GenBank accession alphanumeric string, or a gi numeric string.
- **...**
  - Curl args passed on to `GET`

Details


Examples

## Not run:

```r
# with accession numbers
genbank2uid(id = 'AJ748748')
genbank2uid(id = 'Y13155')
genbank2uid(id = 'X78312')
genbank2uid(id = 'KM495596')
```
get_boldid

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

Usage
```r
get_boldid(searchterm, fuzzy = FALSE, dataTypes = "basic",
  includeTree = FALSE, ask = TRUE, verbose = TRUE, rows = NA, ...)
```
```r
as.boldid(x, check = TRUE)
```
```r
## S3 method for class 'boldid'
as.boldid(x, check = TRUE)
```
```r
## S3 method for class 'character'
as.boldid(x, check = TRUE)
```
```r
## S3 method for class 'list'
as.boldid(x, check = TRUE)
```
```r
## S3 method for class 'numeric'
as.boldid(x, check = TRUE)
```
```r
## S3 method for class 'data.frame'
as.data.frame(x, ...)
```
get_boldid

get_boldid_(searchterm, verbose = TRUE, fuzzy = FALSE,
dataTypes = "basic", includeTree = FALSE, rows = NA, ...)

Arguments

  searchTerm  character; A vector of common or scientific names.
  fuzzy       (logical) Whether to use fuzzy search or not (default: FALSE).
  DataTypes   (character) Specifies the datatypes that will be returned. See Details 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 the species, the user is asked for input. If FALSE NA
              is returned for multiple matches.
  verbose     logical; should progress be printed?
  rows        numeric; Any number from 1 to infinity. If the default NA, all rows are consid-
              ered. 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.

...  Curl options passed on to GET

  x  Input to as.bolid

  check logical; Check if ID matches any existing on the DB, only used in as.bolid

Value

A vector of BOLD ids. If a taxon is not found NA. If more than one BOLD ID is found the
function asks for user input (if ask = TRUE), otherwise returns NA. Comes with an attribute
match to investigate the reason for NA (either 'not found', 'found' or if ask = FALSE 'multi match')

See Also

get_uid, classification

Examples

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

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

# Get back a subset
get_boldid(searchterm="Osmi", fuzzy=TRUE, rows = 1)
get_boldid(searchterm="Osmi", fuzzy=TRUE, rows = 1:10)
get_boldid(searchterm=c("Osmi","Agas"), fuzzy=TRUE, rows = 1)
get_colid

Get the Catalogue of Life ID from taxonomic names.

**Description**

Get the Catalogue of Life ID from taxonomic names.

**Usage**

```r
get_colid(sciname, ask = TRUE, verbose = TRUE, rows = NA)
```
as.colid(x, check = TRUE)

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

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

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

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

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

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

get_colid_(sciname, verbose = TRUE, rows = NA)

Arguments

sciname character; scientific name.
ask logical; should get_colid 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.
verbose 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 colid class object with one to many identifiers. See get_colid_ to get back all, or a subset, of the raw data that you are presented during the ask process.
x Input to as.colid
check logical; Check if ID matches any existing on the DB, only used in as.colid
... Ignored

Value

A vector of unique identifiers. If a taxon is not found NA. If more than one ID is found the function asks for user input.

Author(s)

Scott Chamberlain, <myrmecocystus@gmail.com>
get_colid

See Also

get_tsn, get_colid, get_tpsid, get_eolid

Examples

```r
## Not run:
get_colid(sciname='Poa annua')
get_colid(sciname='Pinus contorta')
get_colid(sciname='Puma concolor')
get_colid(sciname="Abudefduf saxatillis")

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

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

# When not found
get_colid(sciname="uauudnadndj")
gelog(c("Chironomus riparius", "uauudnadndj"))

# Convert a uid without class information to a uid class
as.numeric(get_colid("Chironomus riparius")) # already a uid, returns the same
as.numeric(get_colid(c("Chironomus riparius", "Pinus contorta"))) # same
as.numeric(8663146) # numeric
as.numeric(c(8663146,19736162,18158318)) # numeric vector, length > 1
as.character("19736162") # character
as.character(c("8663146","19736162","18158318")) # character vector, length > 1
as.character(list("8663146","19736162","18158318")) # list, either numeric or character

## dont check, much faster
as.numeric("8663146", check=FALSE)
as.numeric(8663146, check=FALSE)
as.character(c("8663146","19736162","18158318"), check=FALSE)
as.character(list("8663146","19736162","18158318"), check=FALSE)

(out <- as.numeric(c(8663146,19736162,18158318)))
data.frame(out)
as.numeric(data.frame(out))

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

going(sciname="Andropadus nigriceps fusciceps", rows=1)

## 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(sciname, ask = TRUE, verbose = TRUE, key = NULL, rows = NA, ...)

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_(sciname, verbose = TRUE, key = NULL, rows = NA, ...)
```

Arguments

- `sciname` : character; scientific name.
- `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.
- `verbose` : logical; If TRUE the actual taxon queried is printed on the console.
- `key` : API key
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 \code{get_eolid} to get back all, or a subset, of the raw data that you are presented during the ask process.

... Further args passed on to eol_search()

x Input to \code{as.eolid}

check logical; Check if ID matches any existing on the DB, only used in \code{as.eolid}

Value

A vector of unique identifiers (EOL). If a taxon is not found NA. If more than one ID is found the function asks for user input.

Author(s)

Scott Chamberlain, <myrmecocystus@gmail.com>

See Also

\code{get_tsn, get_uid, get_tpsid}

Examples

```r
# Not run:
get_eolid(sciname='Pinus contorta')
get_eolid(sciname='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(sciname="uaudnadndj")
get_eolid(c("Chironomus riparius", "uaudnadndj"))

# Convert a eolid without class information to a eolid class
as.eolid(get_eolid("Chironomus riparius")) # already a eolid, returns the same
as.eolid(get_eolid(c("Chironomus riparius","Pinus contorta"))) # same
as.eolid(24954444) # numeric
as.eolid(c(24954444,51389511,57266265)) # numeric vector, length > 1
as.eolid("24954444") # character
as.eolid(c("24954444","51389511","57266265")) # character vector, length > 1
as.eolid(list("24954444","51389511","57266265")) # list, either numeric or character

# don't check, much faster
as.eolid("24954444", check=FALSE)
as.eolid(24954444, check=FALSE)
as.eolid(c("24954444","51389511","57266265"), check=FALSE)
```
get_gbifid

Get the GBIF backbone taxon ID from taxonomic names.

Description

Get the GBIF backbone taxon ID from taxonomic names.

Usage

get_gbifid(sciname, ask = TRUE, verbose = TRUE, rows = NA)

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_.(sciname, verbose = TRUE, rows = NA)
Arguments

- **sciname**: character; scientific name.
- **ask**: logical; should get_colid 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.
- **verbose**: 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.
- **x**: Input to as.gbifid
- **check**: logical; Check if ID matches any existing on the DB, only used in as.gbifid
- **...**: Ignored

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 unique identifiers. If a taxon is not found NA. If more than one ID is found the function asks for user input.

Author(s)

Scott Chamberlain, <myrmecocystus@gmail.com>

See Also

get_tsn, get_uid, get_tpsid, get_eolid, get_colid

Examples

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

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

# specify rows to limit choices available
get_gbifid(sciname='Pinus')
get_gbifid(sciname='Pinus', rows=1:10)
get_gbifid(sciname='Pinus', rows=1:3)
```
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(names, db = c("itis", "ncbi", "eol", "col", "tropicos", "gbif", "ubio", "nbn"), ...)

get_ids_(names, db = c("itis", "ncbi", "eol", "col", "tropicos", "gbif", "ubio", "nbn"), rows = NA, ...)
Arguments

names character; Taxonomic name to query.
db character; database to query. One or more of ncbi, itis, eol, col, tropicos, gbif, ubio, or nbn. By default db is set to search all data sources.
... Other arguments passed to get_tsn, get_uid, get_eolid, get_colid, get_tpsid, get_gbifid, get_ubioid, 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).

Note

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

See Also

get_tsn, get_uid, get_eolid, get_colid, get_tpsid, get_gbifid, get_ubioid, or get_nbnid.

Examples

## Not run:
# Plug in taxon names directly
## By default you get ids for all data sources
get_ids(names="Chironomus riparius")

# specify rows to limit choices available
get_ids(names="Poa annua", db=c("col","eol"), rows=1)
get_ids(names="Poa annua", db=c("col","eol"), rows=1:2)

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

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

get_ids(names=c("Chironomus riparius", "Pinus contorta"), db = 'ncbi')
get_ids(names=c("Chironomus riparius", "Pinus contorta"), db = c('ncbi','itis'))
get_ids(names=c("Chironomus riparius", "Pinus contorta"), db = c('ncbi','itis','col'))
get_ids(names="Pinus contorta", db = c('ncbi','itis','col','eol','tropicos'))
get_ids(names="ava avvva", db = c('ncbi','itis','col','eol','tropicos'))
get_ids(names="ava avvva", db = c('ncbi','itis','col','eol','tropicos'), verbose=FALSE)

# Pass on to other functions
out <- get_ids(names="Pinus contorta", db = c('ncbi','itis','col','eol','tropicos'))
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(name, ask = TRUE, verbose = TRUE, rec_only = FALSE, rank = NULL, rows = NA, ...)
```

```
get_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, ...)
```

Arguments

- `name` character; scientific name.
- `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.
get_nbnid

**Arguments**

- `verbose` (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.
- `...` 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 unique identifiers. If a taxon is not found NA. If more than one ID is found the function asks for user input.

**Author(s)**

Scott Chamberlain, <myrmecocystus@gmail.com>

**See Also**

`get_tsn, get_uid, get_tpsid, get_eolid`

**Examples**

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

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

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

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

Get the NameID codes from Tropicos for taxonomic names.

Description

Get the NameID codes from Tropicos for taxonomic names.

Usage

get_tpsid(sciname, ask = TRUE, verbose = TRUE, key = NULL, rows = NA, ...)

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)
get_tpsid

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

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

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

### Arguments

- `sciname` (character): One or more scientific name’s as a vector or list.
- `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.
- `verbose` (logical): If TRUE the actual taxon queried is printed on the console.
- `key` (Your API key; loads from .Rprofile).
- `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.
- `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 unique identifiers. If a taxon is not found NA. If more than one ID is found the function asks for user input.

### Author(s)

Scott Chamberlain, <myrmecocystus@gmail.com>

### See Also

`get_tsn, get_tpsid`

### Examples

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

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

# specify rows to limit choices available
get_tpsid('Poa annua')
get_tpsid('Poa annua', rows=1)
```
get_tsn

get_tpsid('Poa annua', rows=25)
get_tpsid('Poa annua', rows=1:2)

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

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

# factor class names are converted to character internally
spnames <- as.factor(c("Poa annua", "Pinus contorta"))
class(spnames)
get_tpsid(spnames)

# pass in a list, works fine
get_tpsid(list("Poa annua", "Pinus contorta"))

# 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("asdfasdf","Pinus contorta"), rows=1:5)

## 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

```r
get_tsn(searchterm, searchtype = "scientific", accepted = TRUE, 
  ask = TRUE, verbose = TRUE, rows = NA, ...)

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_(searchterm, verbose = TRUE, searchtype = "scientific", 
  accepted = TRUE, rows = NA)
```

Arguments

- **searchterm** character; A vector of common or scientific names.
- **searchtype** character; One of `scientific` or `common`, or any unique abbreviation
- **accepted** logical; If TRUE (default), removes names that are not accepted valid names by ITIS. Set to FALSE 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 teh species, the user is asked for input. If FALSE NA is returned for multiple matches.
- **verbose** 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.
- **...** 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 serial numbers (TSN). If a taxon is not found NA. If more than one TSN
is found the function asks for user input (if ask = TRUE), otherwise returns NA. Comes with an
attribute `match` to investigate the reason for NA (either 'not found', 'found' or if ask = FALSE
'multi match')

See Also

classification

Examples

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

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

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

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

# 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_ubioid

Get the uBio id for a search term.

Description

Retrieve the uBio id of a taxon. This function uses `ubiio_search` internally to search for names.

Usage

```r
get_ubioid(searchterm, searchtype = "scientific", ask = TRUE, 
            verbose = TRUE, rows = NA)

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

Arguments

- `searchterm` character; A vector of common or scientific names.
- `searchtype` character; One of 'scientific' or 'common', or any unique abbreviation.
get_ubioid

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.

verbose 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 ubioid class object with one to many identifiers. See get_ubioid to get back all, or a subset, of the raw data that you are presented during the ask process.

x Input to as.ubioid

check logical; Check if ID matches any existing on the DB, only used in as.ubioid

Value

A vector of uBio ids. If a taxon is not found NA is given. If more than one uBio id is found the function asks for user input (if ask = TRUE), otherwise returns NA. Comes with an attribute match to investigate the reason for NA (either 'not found', 'found' or if ask = FALSE 'multi match')

See Also

guid, ubio_search

Examples

```r
# Not run:
guide(searchterm = "Astragalus aduncus")
guide(c("Salvelinus fontinalis","Pomacentrus brachialis"))
splist <- c("Salvelinus fontinalis", 'Pomacentrus brachialis', "Leptocottus armatus", "Clinocottus recalvus", "Trachurus trachurus", "Harengula clupeola")
guide(splist, verbose=FALSE)

# specify rows to limit choices available
guide('Astragalus aduncus')
guide('Astragalus aduncus', rows=1)
guide('Astragalus aduncus', rows=8)
guide('Astragalus aduncus', rows=1:2)

# When not found
guide(searchterm="howdy")
guide(c("Salvelinus fontinalis", "howdy"))

# Using common names
guide(searchterm="great white shark", searchtype="common")
guide(searchterm=c("bull shark", "whale shark"), searchtype="common")

# Convert a ubioid without class information to a ubioid class
as.ubioid(guide("Astragalus aduncus")) # already a ubioid, returns the same
as.ubioid(guide(c("Chironomus riparius","Pinus contorta"))) # same
as.ubioid(2843681) # numeric
as.ubioid(c(2843681,3339,9696)) # numeric vector, length > 1
```
get_uid (sciname, ask = TRUE, verbose = TRUE, rows = NA)

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)

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(sciname, ask = TRUE, verbose = TRUE, rows = NA)

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)
get_uid

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

get_uid_(sciname, verbose = TRUE, rows = NA)

Arguments

- **sciname**: character; scientific name.
- **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.
- **verbose**: 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 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.
- **x**: Input to `as.uid`
- **check**: logical; Check if ID matches any existing on the DB, only used in `as.uid`
- **...**: Ignored

Value

A vector of unique identifiers (UID). If a taxon is not found NA. If more than one UID is found the function asks for user input (if ask = TRUE), otherwise returns NA. Comes with an attribute `match` to investigate the reason for NA (either 'not found', 'found' or if ask = FALSE 'multi match'). 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.

Author(s)

Eduard Szoece, <eduardszoecegmail.com>

See Also

get_tsn, classification

Examples

```r
## 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"))

# 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
```
Check invasive species status for a set of species from GISD database

Description

This function checks which species (both plants and animals) are considered "invaders" somewhere in the world.

For that end, it checks GISD (http://www.issg.org/database/welcome/) and returns a value, either "Not in GISD" or the brief description presented in GISD.

Note that the webpage contains more information. Also note that the function won’t tell you if it’s exotic in your area, a lot of exotic species are not considered invaders (yet).

As expected, the function is as good as the database is, which I find quite reliable and well maintained. The database is also able to recognize a lot (but not all) of the species synonyms.
Note that `eol_invasive` with source of `gisd` or `gisd100` may end up with different results as this function goes directly to the GISP website, whereas `eol_invasive` only updates their GISP data occasionally. See notes in `eol_invasive`.

Usage

```r
gisd_isinvasive(x, simplify = FALSE, verbose = TRUE)
```

Arguments

- `x`: character; a vector of scientific species names in the form of `c("Genus species")`.
- `simplify`: logical; returns a data.frame with the species name and the values "Invasive", "Not in GISP". I recommand to check first the not simplified version (default), which contains raw information about the level of invasiveness.
- `verbose`: logical; If TRUE (default), informative messages printed.

Value

A data.frame with species names and invasiveness.

Author(s)

Ignasi Bartomeus <nacho.bartomeus@gmail.com>

See Also

eol_invasive

Examples

```r
## Not run:
sp <- c("Carpobrotus edulis", "Rosmarinus officinalis")
## first species is invasive, second one is not.
gisd_isinvasive(sp)
gisd_isinvasive(sp, simplify = 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/](http://gni.globalnames.org/) for information.

Usage

```r
gni_details(id = NULL, all_records = NULL,
            url = "http://gni.globalnames.org/name_strings/")
```
gni_parse

Arguments

id 
Name id.

all_records 
If all_records is 1, GNI returns all records from all repositories for the name string (takes 0, or 1, default is 1).

url 
Base url for the API; leave as is.

Value

Data.frame of results.

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

See Also

gnr_datasources, gni_search.

Examples

## Not run:
gni_details(id = 17802847)
library("plyr")
ldply(list(1265133, 17802847), gni_details)

## End(Not run)

gni_parse Parse scientific names using EOL’s name parser.

Description

Parse scientific names using EOL’s name parser.

Usage

gni_parse(names)

Arguments

names A vector of length 1 or more of taxonomic names

Value

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

**References**

http://gni.globalnames.org/

**See Also**

gbif_parse

**Examples**

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

---

**gni_search**

Search for taxonomic names using the Global Names Index.

**Description**

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

**Usage**

```r
gni_search(search_term = NULL, per_page = NULL, page = NULL, justtotal = FALSE, parse_names = FALSE)
```

**Arguments**

- `search_term`: Name pattern you want to search for. WARNING: Does not work for vernacular/common names. Search term may include following options (Note: can, uni, gen, sp, ssp, au, yr work only for parsed names):
  - * wild card - Search by part of a word (E.g.: planta*)
  - exact exact match - Search for exact match of a literal string (E.g.: exact:Parus major)
  - ns name string- Search for literal string from its beginning (other modifiers will be ignored) (E.g.: ns:parus maj*)
  - can canonical form- Search name without authors (other modifiers will be ignored) (E.g.: can:parus major)
  - uni uninomial- Search for higher taxa (E.g.: uni:parus)
  - gen genus - Search by genus epithet of species name (E.g.: gen:parus)
  - sp species - Search by species epithet (E.g.: sp:major)
  - ssp subspecies - Search by infraspecies epithet (E.g.: ssp:major)
  - au author - Search by author word (E.g.: au:Shipunov)
gni_search

- yr year - Search by year (E.g.: yr:2005)

per_page  Number of items per one page (numbers larger than 1000 will be decreased to 1000) (default is 30).

page  Page number you want to see (default is 1).

justtotal  Return only the total results found.

parse_names  If TRUE, use gni_parse to parse names. Default is FALSE.

Details

Note that you can use fuzzy searching, e.g., by attaching an asterisk to the end of a search term. See the first two examples below.

Value

Data.frame of results.

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

References


See Also

gnr_datasources, gni_search.

Examples

## Not run:
gni_search(search_term = "ani*")
gni_search(search_term = "ama*", per_page = 3, page = 2)
gni_search(search_term = "animalia", per_page = 8, page = 1)
gni_search(search_term = "animalia", per_page = 8, page = 1, justtotal=TRUE)

gni_search(search_term = "Cyanistes caeruleus", parse_names=TRUE)

## End(Not run)
Description

Retrieve data sources used in Global Names Index, see http://gni.globalnames.org/ for information.

Usage

```r
gnr_datasources(todf = TRUE)
```

Arguments

- `todf` logical; Should a data.frame be returned?

Value

json or a data.frame

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

See Also

- `gnr_resolve`

Examples

```r
## 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), ]

# Output as a list  
gnr_datasources(FALSE)

## End(Not run)
```
**gnr_resolve**

Resolve names using Global Names Resolver.

**Description**

Uses the Global Names Index, see [http://gni.globalnames.org/](http://gni.globalnames.org/).

**Usage**

```r
gnr_resolve(names, data_source_ids = NULL, resolve_once = FALSE,
            with_context = FALSE, stripauthority = FALSE, highestscore = TRUE,
            best_match_only = FALSE, preferred_data_sources = NULL, http = "get",
            callopts = list())
```

**Arguments**

- `names` 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.
- `stripauthority` logical; If FALSE (default), gives back names with taxonomic authorities. If TRUE, strips author names.
- `highestscore` logical; Return those names with the highest score for each searched name?
- `best_match_only` (logical) If TRUE, best match only returned.
- `preferred_data_sources` (character) A vector of one or more data source IDs.
- `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
- `callopts` Curl debugging options to pass in http::GET or POST

**Value**

A data.frame.

**Author(s)**

Scott Chamberlain myrmecocystus@gmail.com
See Also

gnr_datasources

Examples

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

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

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

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

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

# Strip taxonomic authorities - default is stripauthority=FALSE
head(gnr_resolve(names = "Helianthus annuus")$results)
head(gnr_resolve(names = "Helianthus annuus", stripauthority=TRUE)$results)

## End(Not run)
```

iplant_resolve  

**iPlant name resolution**

Description

iPlant name resolution

Usage

```
iplant_resolve(query, retrieve = "all", callopts = list())
```

Arguments

- **query**: 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)
- **callopts**: Curl options passed on to `httr::GET`
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", callopts = list())

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 basionym 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 'yyy-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 'yyy-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
isgci record 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
callopts Curl options passed on to httr::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 data frame

References

http://www.ipni.org/link_to_ipni.html
Examples

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

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

## End(Not run)
```

**itis-api**  
*Low level functions for working with the ITIS API.*

Description

Low level functions for working with the ITIS API.

Details

There are many low level functions underlying functions that are meant to be used more often by the user. They are exported from the package, but no manual pages are shown in the package function index. You can still use them though. Here’s a list and links to their manual pages.

- `getacceptednamesfromtsn`
- `getanymatchcount`
- `getcommentdetailfromtsn`
- `getcommonnamesfromtsn`
- `getcoremetadatafromtsn`
- `getcoveragefromtsn`
- `getcredibilityratingfromtsn`
- `getcredibilityratings`
- `getcurrencyfromtsn`
- `getdatedatafromtsn`
- `getdescription`
- `getexpertsfromtsn`
- `getfullhierarchyfromtsn`
- `getfullrecordfromlsid`
- `getfullrecordfromtsn`
- `getgeographicdivisionsfromtsn`
- `getgeographicvalues`
• getglobalspeciescompletenessfromtsn
• gethierarchydownfromtsn
• gethierarchyupfromtsn
• getitistermsfromcommonname
• getitisterms
• getitistermsfromscientificname
• getjurisdictionaloriginfromtsn
• getjurisdictionaloriginvalues
• getjurisdictionvalues
• getkingdomnamefromtsn
• getkingdomnames
• getlastchangedate
• getlsidfromtsn
• getothersourcesfromtsn
• getparenttsnfromtsn
• getpublicationsfromtsn
• getranknames
• getrecordfromlsid
• getreviewyearfromtsn
• getscientificnamefromtsn
• getsynonymnamesfromtsn
• gettaxonauthorshipfromtsn
• gettaxonomicranknamefromtsn
• gettaxonomicusagefromtsn
• gettsnbyvernacularlanguage
• gettsnfromlsid
• getunacceptabilityreasonfromtsn
• getvernacularlanguages
• searchbycommonname
• itis_searchcommon
• searchbycommonnamebeginswith
• searchbycommonnameendswith
• searchbyscientificname
• searchforanymatch
• searchforanymatchpaged
itis_acceptname

Retrieve accepted TSN (with accepted name).

Description

Retrieve accepted TSN (with accepted name).

Usage

itis_acceptname(searchtsn = NA, ...)

Arguments

searchtsn Quoted TSN for a taxonomic group (character).
... Further arguments passed on to getacceptednamesfromtsn

Details

You can print informative messages by setting supmess=FALSE.

Value

Names or TSNs of all downstream taxa.

Examples

## Not run:
itis_acceptname('208527') # TSN accepted - good name
itis_acceptname('504239') # TSN not accepted - input TSN is old

## 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(tsns, downto, intermediate = FALSE, ...)


Arguments

tsns  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

...  Further args passed on to gettaxonomicranknamefromtsn and gethierarchydownfromtsn

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 <myrmecocystus@gmail.com>

Examples

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

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

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

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

# get tribes down from the family Apidae
itis_downstream(tsns = 154394, downto="Tribe")
itis_downstream(tsns = 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 = NULL, by = "tsn", ...)`

**Arguments**

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

**Details**

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

**Examples**

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

# by lsid
itis_getrecord("urn:lsid:itiss.gov:itis_tsn:180543", "lsid")

# suppress message
itis_getrecord(202385, verbose=FALSE)

## End(Not run)
```

itis_hierarchy

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

**Description**

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

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

### Arguments

- **tsn**: One or more TSN’s (taxonomic serial number)
- **what**: One of full (full hierarchy), up (immediate upstream), or down (immediate downstream)
- **...**: Further arguments passed on to `getjurisdictionaloriginfromtsn`

### 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

```r
## 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)
```

### Description

Get kingdom names.

### Usage

```r
itis_kingdnamesmnames(tsn = NULL, ...)
```
**itis_lsid**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsn</td>
<td>One or more TSN’s (taxonomic serial number)</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments passed on to getkingdomnamefromtsn</td>
</tr>
</tbody>
</table>

**Examples**

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

# suppress message
itis_kingdomnames(c(202385, 183833, 180543), verbose=FALSE)

## End(Not run)
```

**itis_lsid**

*Get kingdom names.*

**Description**

Get kingdom names.

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsid</td>
<td>One or more lsid's</td>
</tr>
<tr>
<td>what</td>
<td>What to retrieve. One of tsn, record, or fullrecord</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments passed on to gettsnfromlsid, getrecordfromlsid, or getfullrecordfromlsid</td>
</tr>
</tbody>
</table>

**Examples**

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

# 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**

\[ \text{itis\_name(query = NULL, get = NULL)} \]

**Arguments**

- **query**: TSN number (taxonomic serial number).
- **get**: The rank of the taxonomic name to get.

**Value**

Taxonomic name for the searched taxon.

**Examples**

```r
## 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**

\[ \text{itis\_native(tsn = NULL, what = \"bytsn\", ...)} \]

**Arguments**

- **tsn**: One or more TSN’s (taxonomic serial number).
- **what**: One of bytsn, values, or originvalues
- **...**: Further arguments passed on to getjurisdictionaloriginfromtsn
Examples

```r
## 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))

# suppress message
itis_native(c(180543, 41074, 36616), verbose=FALSE)

## End(Not run)
```

### Description

Get references related to a ITIS TSN.

### Usage

`itis_refs(tsn, ...)`

### Arguments

- `tsn`: One or more TSNs (taxonomic serial number) for a taxonomic group (numeric)
- `...`: Further arguments passed on to getpublicationsfromtsn

### Examples

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

# suppress message
itis_refs(202385, verbose=FALSE)

## End(Not run)
```
**itis_searchcommon**

*Searches common name and acts as thin wrapper around searchbycommonnamebeginswith and searchbycommonnameendswith*

**Description**

Searches common name and acts as thin wrapper around searchbycommonnamebeginswith and searchbycommonnameendswith

**Usage**

`itis_searchcommon(x, from = "begin", ...)`

**Arguments**

- `x` Search terms
- `from` Default is to search from beginning. Use end to search from end.
- `...` Curl options passed on to GET

**Value**

data.frame

**See Also**

searchbycommonnamebeginswith searchbycommonnameendswith

**Examples**

```r
## Not run:
itis_searchcommon("inch", config=timeout(3))
itis_searchcommon("inch", from = "end", config=timeout(3))

## 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 `getranknames`). 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 `gettaxonomicranknamefromtsn`

Details

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

Value

Taxonomic rank names or data.frame of all ranks.

Examples

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

# A single TSN
itis_taxrank(query=202385)
# without message
itis_taxrank(query=202385, verbose=FALSE)

# 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.*
**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 `getitisterms`, `getitistermsfromcommonname`, `getitistermsfromscientificname`

**Examples**

```r
## 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)
```

---

**Description**

Get any matching IUCN species names

**Usage**

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

**Arguments**

- **name**: character; taxon name
- **verbose**: logical; should messages be printed?
- **...**: Further arguments passed on to `link{iucn_summary}`

**See Also**

`iucn_summary`, `iucn_status`

**Examples**

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

## 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 by iucn_summary
... Currently not used

Value

A character vector with the status.

See Also

iucn_summary

Examples

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

iucn_summary

Get a summary from the IUCN Red List.

Description

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

Usage

iucn_summary(sciname, silent = TRUE, parallel = FALSE, ...)

## S3 method for class 'iucn'
iucn_status(x, ...)


**Arguments**

- **sciname** character; Scientific name. Should be cleaned and in the format `<Genus> <Species>`.
- **silent** logical; Make errors silent or not (when species not found).
- **parallel** logical; Search in parallel to speed up search. You have to register a parallel backend if TRUE. See e.g., doMC, doSNOW, etc.
- **x** an iucn object as returned by `iucn_summary`.

**Value**

A list (for every species one entry) of lists with the following items:

- **status** Red List Category.
- **history** History of status, if available.
- **distr** Geographic distribution, if available.
- **trend** Trend of population size, if available.

**Note**

Not all entries (history, distr, trend) 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, <eduard@szoes@gmail.com>

**See Also**

`iucn_status`

**Examples**

```r
## Not run:
ia <- iucn_summary(c("Panthera uncia", "Lynx lynx"))
ia <- iucn_summary(c("Panthera uncia", "Lynx lynx", "aaa"))
# extract status
iucn_status(ia)
# extract other available information
ia["Lynx lynx"]$history
ia["Panthera uncia"]$distr
ia[[2]]$trend

## 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

```r
names_list(rank = "genus", size = 10)
```

Arguments

- **rank**: Taxonomic rank, one of species, genus (default), family, order.
- **size**: Number of names to get. Maximum depends on the rank.

Value

Vector of taxonomic names.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

Examples

```r
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**

```r
nbn_classification(id, ...)
```

**Arguments**

- `id` (character) An NBN identifier.
- `...` Further args passed on to `GET`.

**Author(s)**

Scott Chamberlain, <myrmecocystus@gmail.com>

**Examples**

```r
## Not run:
nbn_classification(id = "NHMSYS0000502940")

# get id first, then pass to this fxn
id <- get_nbnid("blue tit", rec_only = TRUE, rank = "Species")
nbn_classification(id)

library('httr')
nbn_classification(id = "NHMSYS0000502940", config=verbose())

## End(Not run)
```

---

**nbn_search**

*Search UK National Biodiversity Network database*

**Description**

Search UK National Biodiversity Network database

**Usage**

```r
nbn_search(q, prefered = FALSE, order = "asc", sort = NULL, start = 0,
                   rows = 25, taxonOutputGroupKey = NULL, all = FALSE, ...)
```
nbn_synonyms

Arguments

q (character) The query terms(s)
preferred (logical) Restrict search to preferred or any
order (character) The order in which we should sort the results. Default: asc
sort (character) Sort the results or not.
start (integer/numeric) The page that the user wants to start displaying the results at. Default: 0
rows (integer/numeric) The number of rows to show in each page of search results. Default: 25
taxonOutputGroupKey (character) Vector of taxon output groups.
all (logical) Get all results, overrides rows parameter if TRUE. Default: FALSE

Examples

## Not run:
nbn_search(q = "blackbird")
nbn_search(q = "blackbird", start = TRUE)
nbn_search(q = "blackbird", all = TRUE)
nbn_search(q = "blackbird", taxonOutputGroupKey = "NHMSYS0000000039")

# debug curl stuff
library('httr')
nbn_search(q = "blackbird", config = verbose())

## End(Not run)

nbn_synonyms id 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 GET
**ncbi_children**

Search NCBI for children of a taxon

**Value**

A data.frame

**Examples**

```r
## Not run:
nbn_synonyms(id = 'NHMSYS0000502940')
nbn_synonyms(id = 'NHMSYS0001501147')
nbn_synonyms(id = 'NHMSYS000045603')
## End(Not run)
```

---

```r
ncbi_children
```

**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.

**Usage**

```r
ncbi_children(name = NULL, id = NULL, start = 0, max_return = 1000,
ancestor = NULL, out_type = c("summary", "uid"), ambiguous = FALSE)
```

**Arguments**

- **name** (character) The string to search for. Only exact matches found the name given will be returned. Not compatible with id.
- **id** (character) 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".
Details

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.

Value

The output type depends on the value of the out_type parameter.

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
## End(Not run)

ncbi_getbyid Retrieve gene sequences from NCBI by accession number.

Description

Retrieve gene sequences from NCBI by accession number.

Usage

ncbi_getbyid(ids, format = "fasta", verbose = TRUE)

Arguments

ids (character) GenBank ids to search for.
format (character) Return type, e.g., "fasta"
verbose (logical) If TRUE (default), informative messages printed.
ncbi_getbyname

Details

Removes predicted sequences so you don’t have to remove them. Predicted sequences are those with accession numbers that have "XM_" or "XR_" prefixes. This function retrieves one sequences for each species, picking the longest available for the given gene.

Value

Data.frame of results.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

See Also

ncbi_search, ncbi_getbyname

Examples

## Not run:
# A single gene
ncbi_getbyid(ids="360040093", format="fasta")

# Many genes (with different accession numbers)
ncbi_getbyid(ids=c("360040093","347448433"), format="fasta")

## End(Not run)

ncbi_getbyname Retrieve gene sequences from NCBI by taxon name and gene names.

Description

Retrieve gene sequences from NCBI by taxon name and gene names.

Usage

ncbi_getbyname(taxa, gene = "COI", seqrange = "1:3000",
getrelated = FALSE, verbose = TRUE)

Arguments

taxa  (character) Scientific name to search for.
gene  (character) Gene or genes (in a vector) to search for. See examples.
seqrange  (character) Sequence range, as e.g., "1:1000". This is the range of sequence lengths to search for. So "1:1000" means search for sequences from 1 to 1000 characters in length.
getrelated (logical) If TRUE, gets the longest sequences of a species in the same genus as the one searched for. If FALSE, returns nothing if no match found.

verbose (logical) If TRUE (default), informative messages printed.

Details

Removes predicted sequences so you don’t have to remove them. Predicted sequences are those with accession numbers that have "XM_" or "XR_" prefixes. This function retrieves one sequence for each species, picking the longest available for the given gene.

Value

Data.frame of results.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

See Also

ncbi_search, ncbi_getbyid

Examples

## Not run:
# A single species
ncbi_getbyname(taxa="Acipenser brevirostrum")

# Many species
species <- c("Colletes similis","Halictus ligatus","Perdita trisignata")
ncbi_getbyname(taxa=species, gene = c("coi", "col"), seqrange = "1:2000")

## End(Not run)
**ncbi_search**

**Arguments**

- **id** (character) NCBI taxonomy uids to retrieve information for.

**Value**

A data.frame with the following rows:

- **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')

**Author(s)**

Zachary Foster <zacharyfoster1989@sgmail.com>

**Examples**

```r
## Not run:
ncbi_get_taxon_summary(c(1430660, 4751))

## End(Not run)
```

---

**ncbi_search**  
*Search for gene sequences available for taxa from NCBI.*

**Description**

Search for gene sequences available for taxa from NCBI.

**Usage**

```r
ncbi_search(taxa = NULL, id = NULL, seqlong = "1:3000",  
getrelated = FALSE, limit = 500, entrez_query = NULL,  
hypothetical = FALSE, verbose = TRUE)
```

**Arguments**

- **taxa** (character) Scientific name to search for.
- **id** (character) Taxonomic id to search for. Not compatible with argument taxa.
- **seqlong** (character) Sequence range, as e.g., "1:1000". This is the range of sequence lengths to search for. So "1:1000" means search for sequences from 1 to 1000 characters in length.
- **getrelated** (logical) If TRUE, gets the longest sequences of a species in the same genus as the one searched for. If FALSE, returns nothing if no match found.
- **limit** (numeric) Number of sequences to search for and return. Max of 10,000. If you search for 6000 records, and only 5000 are found, you will of course only get 5000 back.
entrez_query (character; length 1) An Entrez-format query to filter results with. This is useful to search for sequences with specific characteristics. The format is the same as the one used to search genbank. ([http://www.ncbi.nlm.nih.gov/books/NBK3837/#EntrezHelp.Entrez_Searching_Options](http://www.ncbi.nlm.nih.gov/books/NBK3837/#EntrezHelp.Entrez_Searching_Options))

hypothetical (logical; length 1) If FALSE, an attempt will be made to not return hypothetical or predicted sequences judging from accession number prefixes (XM and XR). This can result in less than the limit being returned even if there are more sequences available, since this filtering is done after searching NCBI.

verbose (logical) If TRUE (default), informative messages printed.

Value
data.frame of results if a single input is given. A list of data.frames if multiple inputs are given.

Author(s)
Scott Chamberlain <myrmecocystus@gmail.com>, Zachary Foster <zacharyfoster1989@gmail.com>

See Also
ncbi_getbyid, ncbi_getbyname

Examples

```r
## Not run:
# A single species
out <- ncbi_search(taxa="Umbra limi", seqrange = "1:2000")
# Get the same species information using a taxonomy id
out <- ncbi_search(id = "75935", seqrange = "1:2000")
# If the taxon name is unique, using the taxon name and id are equivalent
all(ncbi_search(id = "75935") == ncbi_search(taxa="Umbra limi"))
# If the taxon name is not unique, use taxon id
# "266948" is the uid for the butterfly genus, but there is also a genus of orchids with the
# same name
nrow(ncbi_search(id = "266948")) == nrow(ncbi_search(taxa="Satyrium"))
# get list of genes available, removing non-unique
unique(out$gene_desc)
# does the string 'RAG1' exist in any of the gene names
out[grep("RAG1", out$gene_desc, ignore.case=TRUE),]

## A single species without records in NCBI
out <- ncbi_search(taxa="Sequoia wellingtonia", seqrange="1:2000", getrelated=TRUE)

## Many species, can run in parallel or not using plyr
species <- c("Salvelinus alpinus","Ictalurus nebulosus","Carassius auratus")
out2 <- ncbi_search(taxa=species, seqrange = "1:2000")
lapply(out2, head) # see heads of all
library("plyr")
out2df <- ldply(out2) # make data.frame of all
unique(out2df$gene_desc) # get list of genes available, removing non-unique
out2df[grep("60S ribosomal protein", out2df$gene_desc, ignore.case=TRUE),] # search across all
```
phylomatic_format

Get family names to make Phylomatic input object, and output input string to Phylomatic for use in the function phylomatic_tree.

Description

Get family names to make Phylomatic input object, and output input string to Phylomatic for use in the function phylomatic_tree.

Usage

phylomatic_format(taxa = NA, format = “isubmit”, db = “ncbi”)

Arguments

taxa quoted tsn number (taxonomic serial number)
format output format, isubmit (you can paste in to the Phylomatic website), or ‘rsubmit’ to use in fxn phylomatic_tree
db One of “ncbi”, “itis”, or “apg”

Value

e.g., “pinaceae/pinus/pinus_contorta”, in Phylomatic submission format.

Examples

## Not run:
mynames <- c(“Poa annua”, “Salix goodingii”, “Helianthus annuus”)
phylomatic_format(mynames, format=’rsubmit’)
phylomatic_format(mynames, format=’isubmit’, db=”apg”)

## End(Not run)
**phylomatic_tree**  
*Query Phylomatic for a phylogenetic tree.*

### Description

Query Phylomatic for a phylogenetic tree.

### Usage

```r
phylomatic_tree(taxa, taxnames = TRUE, get = "GET", informat = "newick",
method = "phylomatic", storedtree = "R20120829", treeuri = NULL,
taxaformat = "slashpath", outformat = "newick", clean = "true",
db = "apg", verbose = TRUE)
```

### Arguments

- **taxa**: Phylomatic format input of taxa names.
- **taxnames**: If true, we get the family names for you to attach to your species names to send to Phylomatic API. If FALSE, you have to provide the strings in the right format.
- **get**: 'GET' or 'POST' format for submission to the website.
- **informat**: One of newick, nexml, or cdaordf. If using a stored tree, informat should always be newick.
- **method**: One of phylomatic or convert
- **storedtree**: One of R20120829 (Phylomatic tree R20120829 for plants), smith2011 (Smith 2011, plants), or binindaemonds2007 (Bininda-Emonds 2007, mammals).
- **treeuri**: URL for a phylogenetic tree in newick format.
- **taxaformat**: Only option is slashpath for now. Leave as is.
- **outformat**: One of newick, nexml, or fyt.
- **clean**: Return a clean tree or not.
- **db**: One of "ncbi", "itis", or "apg". If there are gymnosperms in your taxa list, don’t use apg, instead use ncbi or itis.
- **verbose**: Print messages (default: TRUE).

### Details

Use the web interface here [http://phylodiversity.net/phylomatic/](http://phylodiversity.net/phylomatic/)

### Value

Newick formatted tree or nxml text.
Examples

## Not run:

```r
# Input taxonomic names
taxa <- c("Poa annua", "Phlox diffusa", "Helianthus annuus")
tree <- phylomatic_tree(taxa=taxa, get = 'POST')
plot(tree, no.margin=TRUE)
```

```r
# Genus names
taxa <- c("Poa", "Phlox", "Helianthus")
tree <- phylomatic_tree(taxa=taxa, storedtree='R20120829', get='POST')
plot(tree, no.margin=TRUE)
```

```r
# Lots of names
taxa <- c("Poa annua", "Collomia grandiflora", "Lilium lankongense", "Phlox diffusa", 
"Iteadaphne caudata", "Gagea sarmentosa", "Helianthus annuus")
tree <- phylomatic_tree(taxa=taxa, get = 'POST')
plot(tree, no.margin=TRUE)
```

```r
# Output NeXML format
out <- phylomatic_tree(taxa=taxa, out = 'POST', outformat = "nexml")
cat(out)
```

```r
# Lots of names, note that when you have enough names (number depends on length of individual 
# names, so there's no per se rule), you will get an error when using \code{get='GET'},
# when that happens use \code{get='POST'}
spp <- names_list("species", 200)
(out <- phylomatic_tree(taxa = spp, get = "GET"))
(out <- phylomatic_tree(taxa = spp, get = "POST"))
plot(out)
```

```r
# Pass in a tree from a URL on the web
url <- "http://datadryad.org/bitstream/handle/10255/dryad.8791/final_tree?sequence=1"
spp <- c('Abies amabilis','Abies balsamea','Abies bracteata','Abies concolor','Abies fraseri', 
'Abies grandis','Abies lasiocarpa','Abies magnifica','Abies procera','Acacia berlandieri')
tree <- phylomatic_tree(taxa = spp, treeuri=url)
```

```r
# If there gymnosperms in your taxa list, use db of itis or ncbi
taxa <- c("Abies amabilis","Abies balsamea","Abies grandis","Abies lasiocarpa", 
"Abies magnifica","Abies procera","Acacia berlandieri","Poa annua")
tree1 <- phylomatic_tree(taxa=taxa, db="ncbi")
plot(tree1)
```

## 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

```r
col_ping(what = "status", ...)

eol_ping(what = "status", ...)

itis_ping(what = "status", ...)

ncbi_ping(what = "status", ...)

tropicos_ping(what = "status", ...)

nbn_ping(what = "status", ...)

gbif_ping(what = "status", ...)

ubio_ping(what = "status", ...)

bold_ping(what = "status", ...)

ipni_ping(what = "status", ...)

vascan_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 `GET`

Details

For ITIS, see `getdescription`, which provides number of scientific and common names in a character string.

Value

A logical, TRUE or FALSE

Examples

```r
## Not run:
col_ping()
col_ping("content")
col_ping(200)
col_ping("200")
col_ping(204)
```
### plantGenusNames

Vector of 793 genus names of plants

**Description**

Vector of 793 genus names of plants

---

### plantminer

Search for taxonomy data from Plantminer.com

**Description**

Search for taxonomy data from Plantminer.com

**Usage**

```r
plantminer(plants, key = NULL, verbose = TRUE)
```
Arguments

plants  Vector of plant species names.
key    Your api key for the plantminer.com site. Go to http://www.plantminer.com/ to get your api key. Two options for inputting your key. 1) You can input it manually within the function as the second argument, or 2) you can put the key in your .Rprofile file, which will then be loaded when you start R. See http://bit.ly/135eG0b for help on how to put api keys in your .Rprofile file.
verbose Verbose or not, logical

Value
data.frame of results.

Examples

## Not run:
plants <- c("Myrcia lingua", "Myrcia bella", "Ocotea pulchella", "Miconia", "Coffea arabica var. amarella", "Bleh")
plantminer(plants)
## End(Not run)

plantNames  Vector of 1,182 species names of plants

Description

Vector of 1,182 species names of plants

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

```r
data(dune.taxon, package='vegan')
dat <- dune.taxon
dat$abundance <- round(rlnorm(n=nrow(dat),meanlog=5,sdlog=2),0)
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="Class")
rankagg(data=dat, "abundance", rank="Class", fxn="sd")
```

---

**rank_ref**

*Lookup-table for IDs of taxonomic ranks*

**Description**

Lookup-table for IDs of taxonomic ranks

---

**resolve**

*Resolve names from many different sources*

**Description**

Resolve names from iPlant’s name resolver, the Taxonomic Name Resolution Service (TNRS), the Global Names Resolver (GNR)

**Usage**

```r
resolve(query, db = "iplant", callopts = list(), ...)
```

**Arguments**

- **query**
  - Vector of one or more names.
- **db**
  - Source to check names against. One of iplant, tnr5, or gnr
- **callopts**
  - Curl options passed on to `httr::GET`
- **...**
  - Further named args passed on to each respective function. See examples. Note that parameters for specific data sources are specific to those data sources. E.g. you can pass the parameter `source` when using `db="tnrs"`, but is meaningless when passed when `db="gnr"`. There is one exception - the `callopts` parameter is shared among all data sources, so if you pass that parameter it will influence each data source.

**Value**

A list with length equal to length of the `db` parameter (number of sources requested.)
Examples

```r
## Not run:
resolve(query=c("Helianthus annuus", "Homo sapiens"))
resolve(query=c("Helianthus annuus", "Homo sapiens"), db='tnrs', source="iPlant_TNRS")
resolve(query="Quercus kelloggii", db='gnr')
resolve(query="Helianthus annuus", db='gnr', preferred_data_sources = c(3,4))
resolve(query=c("Helianthus annuus", "Homo sapiens"), db=c('iplant','gnr'))
library("httr")
resolve(query="Quercus", callopts=verbose())
resolve(query=c("Helianthus annuus", "Homo sapiens"), callopts=timeout(3))
```

## End(Not run)

---

**sci2comm**  
*Get common names from scientific names.*

Description

Get common names from scientific names.

Usage

`sci2comm(...)`

## Default S3 method:
`sci2comm(scinames, db = "eol", simplify = TRUE, ...)`

## S3 method for class 'uid'
`sci2comm(id, ...)`

## S3 method for class 'tsn'
`sci2comm(id, simplify = TRUE, ...)`

Arguments

... Further arguments passed on to functions `get_uid`, `get_tsn`.

scinames character; One or more scientific names or partial names.

db character; Data source, one of "eol" (default), "itis" or "ncbi".

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 its.

id character; identifiers, as returned by `get_tsn`, `get_uid`. 
scrapenames

Details

Note that EOL requires an API key. You can pass in your EOL api key in the function call like `sci2comm('Helianthus annuus', key="<your eol api key>")`. You can also store your EOL API key in your .Rprofile file as `options(eolApiKey = "<your eol api key>")`, or just for the current session by running `options(eolApiKey = "<your eol api key>")` in the console.

Value

List of character vectors.

Author(s)

Scott Chamberlain (myrmecocystus@gmail.com)

See Also

`searchbycommonname`, `searchbycommonnamebeginswith`, `searchbycommonnameendswith`, `eol_search`, `tp_search`, `comm2sci`

Examples

```r
## Not run:
si2comm(scinames='Helianthus annuus', db='eol')
si2comm(scinames='Helianthus annuus', db='itis')
si2comm(scinames=c('Helianthus annuus', 'Poa annua'))
si2comm(scinames='Puma concolor', db='ncbi')

# Passing id in, works for sources: itis and ncbi, not eol
si2comm(get_tsn('Helianthus annuus'))
si2comm(get_uid('Helianthus annuus'))

# Don't simplify returned
si2comm(get_tsn('Helianthus annuus'), simplify=FALSE)

## End(Not run)
```

scrapenames

Resolve names using Global Names Recognition and Discovery.

Description

Uses the Global Names Recognition and Discovery service, see [http://gnrd.globalnames.org/](http://gnrd.globalnames.org/).

Usage

```r
scrapenames(url = NULL, file = NULL, text = NULL, engine = NULL,
unique = NULL, verbatim = NULL, detect_language = NULL,
all_data_sources = NULL, data_source_ids = NULL, callopts = list())
```
Arguments

url
An encoded URL for a web page, PDF, Microsoft Office document, or image file, see examples

file
When using multipart/form-data as the content-type, a file may be sent. This should be a path to your file on your machine.

text
Type: string. Text content; best used with a POST request, see examples

engine
(optional) (integer) Default: 0. Either 1 for TaxonFinder, 2 for NetiNeti, or 0 for both. If absent, both engines are used.

unique
(optional) (logical) If TRUE (default), response has unique names without offsets.

verbatim
(optional) Type: boolean. If TRUE (default to FALSE), response excludes verbatim strings.

detect_language
(optional) Type: boolean. When TRUE (default), NetiNeti is not used if the language of incoming text is determined not to be English. When 'false', NetiNeti will be used if requested.

all_data_sources
(optional) Type: boolean. Resolve found names against all available Data Sources.

data_source_ids
(optional) Type: string. Pipe separated list of data source ids to resolve found names against. See list of Data Sources http://resolver.globalnames.org/data_sources.

callopts
Further args passed to GET

Details

Note: this function sometimes gives data back and sometimes not. The API that this function is extremely buggy.

One of url, file, or text must be specified - and only one of them.

Value

A list of length two, first is metadata, second is the data as a data.frame.

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

Examples

## Not run:
# Get data from a website using its URL
scrapenames(url = 'http://en.wikipedia.org/wiki/Araneae')
scrapenames(url = 'http://en.wikipedia.org/wiki/Animalia')
scrapenames(url = 'http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0095068')
scrapenames(url = 'http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0080498')
scrapenames(url = 'http://ucjeps.berkeley.edu/cgi-bin/get_JM_treatment.pl?CARYOPHYLLACEAE')
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(x, db = NULL, ...)

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

## S3 method for class 'tpsid'
synonyms(id, ...)

## S3 method for class 'ubioid'
synonyms(id, ...)

## S3 method for class 'nbnid'
synonyms(id, ...)

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

Arguments

... Other passed arguments.

x character; taxons to query.

db character; database to query. either itis, tropicos, ubio, or nbn.

id character; identifiers, returned by get_tsn, get_tpsid, get_ubioid, or get_nbnid

Value

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

Note

If IDs are supplied directly (not from the get_* functions) you must specify the type of ID.

See Also

get_tsn, get_tpsid, get_ubioid, get_nbnid

Examples

## Not run:
# Plug in taxon names directly
synonyms("Poa annua", 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")
Defunct functions in taxize

Description

The following functions are now defunct (no longer available):

Details

- **col_classification**: See `classification`
- **eol_hierarchy**: See `classification`
- **tp_classification**: See `classification`
- **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`.

Deprecated functions in taxize

Description

The following functions are now deprecated:
Details

- `ncbi_getbyname`: This function is deprecated and will be removed in a future version of this package.
- `ncbi_getbyid`: This function is deprecated and will be removed in a future version of this package.
- `ncbi_search`: This function is deprecated and will be removed in a future version of this package.
- `phylomatic_tree`: This function is deprecated and will be removed in a future version of this package.
- `eol_invasive`: This function is deprecated and will be removed in a future version of this package.
- `gisd_isinvasive`: This function is deprecated and will be removed in a future version of this package.

All of the above functions will be available in an R package in development called traits [https://github.com/opensci/traits](https://github.com/opensci/traits) that will be available on CRAN before these functions are made defunct here.

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

```r
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

```r
taxize_cite(fxn='tnrs')
taxize_cite(fxn='eol')

# Get the taxize citation
taxize_cite(fxn='taxize')

# Get license information
taxize_cite(fxn='taxize', "license")
taxize_cite(fxn='tnrs', "license")
```

tax_agg

Aggregate species data to given taxonomic rank

Description

Aggregate species data to given taxonomic rank

Usage

```r
tax_agg(x, rank, db = "ncbi", verbose = 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`.
- **verbose**: (logical) If FALSE (Default) suppresses 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:
# use dune dataset
data(dune, package='vegan')
species <- c("Bellis perennis", "Empetrum nigrum", "Juncus bufonius",
"Juncus articulatus",
"Aira praecox", "Eleocharis parvula", "Rumex acetosa", "Vicia lathyroides",
"Brachythecium rutabulum", "Ranunculus flammula", "Cirsium arvense",
"Hypochaeris radicata", "Leontodon autumnalis", "Potentilla palustris",
"Poa pratensis", "Calliergonella cuspidata", "Trifolium pratense",
"Trifolium repens", "Anthoxanthum odoratum", "Salix repens", "Achillea millefolium",
"Poa trivialis", "Chenopodium album", "Elymus repens", "Sagina procumbens",
"Plantago lanceolata", "Agrostis stolonifera", "Lolium perenne", "Alopecurus geniculatus",
"Bromus hordeaceus")
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

# A use case where there are different taxonomic levels in the same dataset

```r
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(df, rank = 'family', db='itis')
out$x
```

# You can input a matrix too

```r
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**

Retrieve name of queried taxonomic rank of a taxon.

**Usage**

```r
tax_name(query, get, db = "itis", pref = "ncbi", verbose = TRUE, ...)
```

**Arguments**

- `query` character; Vector of taxonomic names to query.
- `get` character; The ranks of the taxonomic name to get, see `rank_ref`.
- `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.
- `pref` character; If `db = 'both'`, sets the preference for the union. Either 'ncbi' (default) or 'itis'. Currently not implemented.
- `verbose` logical; If TRUE the actual taxon queried is printed on the console.
- `...` 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.

**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

```R
## Not run:
# A case where itis and ncbi use the same names
tax_name(query = "Helianthus annuus", get = "family", db = "itis")
tax_name(query = "Helianthus annuus", get = "family", db = "ncbi")
tax_name(query = "Helianthus annuus", get = c("genus","family","order"), db = "ncbi")

# Case where itis and ncbi use different names
tax_name(query = "Helianthus annuus", get = "kingdom", db = "itis")
tax_name(query = "Helianthus annuus", get = "kingdom", db = "ncbi")

# multiple get arguments
tax_name(query = c("Helianthus annuus","Baetis rhodani"), get = c("genus","kingdom"), db = "ncbi")
tax_name(query = c("Helianthus annuus","Baetis rhodani"), get = c("genus","kingdom"), db = "itis")

# query both sources
tax_name(query=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 taxonomic rank for a given taxon name.

Usage

tax_rank(query = NULL, db = "itis", pref = "ncbi", verbose = TRUE, ...)

Arguments

- `query` character; Vector of taxonomic names to query.
- `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.
- `pref` If `db = 'both'`, sets the preference for the union. Either 'ncbi' or 'itis'.
- `verbose` logical; If TRUE the actual taxon queried is printed on the console.
- `...` Other arguments passed to `get_tsn` or `get_uid`. 
### Value

A data.frame with one column for every queried taxon.

### 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

```r
## Not run:
tax_rank(query = "Helianthus annuus", db = "itis")
tax_rank(query = "Helianthus annuus", db = "ncbi")
tax_rank(query = "Helianthus", db = "itis")

# query both
tax_rank(query=c("Helianthus annuus", 'Puma'), db="both")

# An alternative way would be to use \link{classification} and sapply over
# the list
x <- 'Baetis'
classi <- classification(get_uid(x))
sapply(classi, function(x) x[row(x), 'rank'])

## End(Not run)
```
Usage

tnrs(query = NA, source = NULL, code = NULL, getpost = "POST",
sleep = 0, splitby = 30, verbose = TRUE, callopts = list())

Arguments

- **query**: Quoted taxonomic names to search in a vector (character).
- **source**: Specify the source you want to match names against. Defaults to just retrieve data from all sources. Options: NCBI, iPlant_TNRS, or MSW3. Only available when using getpost="POST".
- **code**: Nomenclatural code. One of: ICZN (zoological), ICN (algae, fungi, and plants), ICNB (bacteria), ICBN (botanical), ICNCP (cultivated plants), ICTV (viruses). Only available when using getpost="POST".
- **getpost**: Use GET or POST method to send the query. If you have more than say 50 species or so in your query, you should probably use POST. IMPORTANT!!!!!! -> POST is the only option for this parameter if you want to use source or code parameters.
- **sleep**: Number of seconds by which to pause between calls. Defaults to 0 seconds. Use when doing many calls in a for loop or lapply type call.
- **splitby**: Number by which to split species list for querying the TNRS.
- **verbose**: Verbosity or not (default TRUE)
- **callopts**: Curl debugging options to pass in httr::GET or POST

Details

If there is no match in the Taxosaurus database, nothing is returned, so you will not get anything back for non matches.

Value

data.frame of results from TNRS plus the name submitted.

Examples

```r
## Not run:
# Default, uses GET curl method, you can't specify any other parameters when using GET
mynames <- c("Panthera tigris", "Neotamias minimus", "Magnifera indica")
tnrs(query = mynames)

# Specifying the source to match against
mynames <- c("Helianthus annuus", "Poa annua")
tnrs(query = mynames, source = "iPlant_TNRS")

# Specifying the nomenclatural code to match against
mynames <- c("Helianthus annuus", "Poa annua")
tnrs(query = mynames, code = "ICBN")
```
tnrs_sources

# You can specify multiple sources, by comma-separating them
mynames <- c("Panthera tigris", "Eutamias minimus", "Magnifera indica", "Humbert humbert")
trs(query = mynames, source = "NCBI,MSW3")

# Using POST method, especially useful when you have a lot of species
trs(mynames, source = "NCBI")

# And even more names
mynames <- names_list(rank="species", size=75)
trs(query=mynames, source = "NCBI")
## Or use splitby
trs(mynames, source = "NCBI", splitby=50)

## End(Not run)

---

**tnrs_sources**

*Get sources for the Phylotastic Taxonomic Name Resolution Service.*

**Description**

Get sources for the Phylotastic Taxonomic Name Resolution Service.

**Usage**

```r
tnrs_sources(source = NULL)
```

**Arguments**

| source       | The source to get information on, one of ”iPlant_TNRS”, ”NCBI”, or ”MSW3”. |

**Value**

Sources for the TNRS API.

**Examples**

```r
## Not run:
# All
tnrs_sources()

# A specific source
tnrs_sources(source="NCBI")

## End(Not run)
```
 tpl_families  

*Get The Plant List families.*

**Description**

Get The Plant List families.

**Usage**

```r
tpl_families()
```

**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_search` `tpl_get`

**Examples**

```r
## 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**

The Plant List [http://www.theplantlist.org/](http://www.theplantlist.org/). Note there is a package on CRAN (taxonstand - [http://cran.r-project.org/web/packages/Taxonstand/](http://cran.r-project.org/web/packages/Taxonstand/)) that uses theplantlist.org to search plant names - we have a wrapper around that function called `tpl_search`. 
Usage

tpl_get(dir_, family = NULL)

Arguments

dir_ Directory to write csv files to.
family If you want just one, or >1 family, but not all, list them in a vector.

Details

Throws a warning if you already have a directory of the one provided, but still works. Writes to your home directory, change dir_ as needed.

Value

Returns nothing to console, except a message and progress bar. Writes csv files to dir_.

Author(s)

John Baumgartner (johnbb@student.unimelb.edu.au)

See Also

tpl_search tpl_families

Examples

## Not run:
# Get a few families
tpl_get(dir_ = "~/foo2", family = c("Platanaceae","Winteraceae"))

# You can now get Gymnosperms as well
tpl_get(dir_ = "~/foo2", family = c("Pinaceae","Taxaceae"))

# You can get mosses too!
tpl_get2(dir_ = "~/foo4", family = "Echinochiaceae")

# Get all families
tpl_get(dir_ = "~/foo")

## 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(taxon, paral = FALSE, ...)

Arguments

- **taxon**: A taxonomic name, or a vector of names.
- **paral**: Paralellize or not (logical). Which back-end package you use depends on your operating system and just general taste. Possibilities include: snow, multicore, parallel, doMC, etc.
- **...**: Further arguments passed on to the TPL or TPLck function of Taxonstand. See_tpl_and tplck for arguments.

Details

This function was a very light wrapper around TPLck anyway - we don’t see further reason to keep this function. Use the Taxonstand functions TPL and TPLck, directly. However, we are keeping two functions that give you access to download Theplantlist.org data tpl_get and to get Theplantlist.org families tpl_families.

See Also

tpl_get, tpl_families

Examples

---

```r
## Not run:
# Regular non-parallel
splist <- c("Helianthus annuus","Abies procera","Poa annua",
  "Platanus occidentalis","Carex abrupta","Arctostaphylos canescens",
  "Ocimum basilicum","Vicia faba","Quercus kelloggii","Lactuca serriola")
tpl_search(taxon = splist)

# Use more arguments within TPLck
tpl_search(taxon = "Microbryum curvicollum", corr = TRUE)
tpl_search(taxon = "Microbryum curvicollum", corr = TRUE, max.distance=5)

## End(Not run)
```
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, callopts = list())

Arguments

- id: the taxon identifier code
- key: Your Tropicos API key; loads from .Rprofile.
- callopts: Further args passed on to httr::GET

Value

List or dataframe.

Examples

```r
## 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 a taxon name with a given id.

Description

Return all distribution records for a taxon name with a given id.

Usage

tp_dist(id, key = NULL, callopts = list())
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the taxon identifier code</td>
</tr>
<tr>
<td>key</td>
<td>Your Tropicos API key; loads from .Rprofile. Or you can pass in your key in this arg.</td>
</tr>
<tr>
<td>callopts</td>
<td>Further args passed on to http::GET</td>
</tr>
</tbody>
</table>

Value

List of two data.frame’s, one named "location", and one "reference".

References

http://services.tropicos.org/help?method=GetNameDistributionsXml

Examples

```r
## 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 a taxon name with a given id.

Description

Return all reference records for a taxon name with a given id.

Usage

tp_refs(id, key = NULL, callopts = list())

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the taxon identifier code</td>
</tr>
<tr>
<td>key</td>
<td>Your Tropicos API key; loads from .Rprofile.</td>
</tr>
<tr>
<td>callopts</td>
<td>Curl options.</td>
</tr>
</tbody>
</table>

Value

List or data frame.
tp_search

Examples

```r
## 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

```r
tp_search(name = NULL, commonname = NULL, nameid = NULL, orderby = NULL,
          sortorder = NULL, pagesize = NULL, startrow = NULL, type = NULL,
          key = NULL, callopts = list())
```

Arguments

- `name`: Your search string. For instance "poa annua"
- `commonname`: Your search string. For instance "annual blue grass"
- `nameid`: Your search string. For instance "25509881"
- `orderby`: Your search string. For instance "1"
- `sortorder`: Your search string. For instance "ascending"
- `pagesize`: Your search string. For instance "100"
- `startrow`: Your search string. For instance "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; loads from `.Rprofile`
- `callopts`: Further args passed on to `httr::GET`

Value

List or dataframe.

References

[http://services.tropicos.org/help?method=SearchNameXml](http://services.tropicos.org/help?method=SearchNameXml)

Examples

```r
## Not run:
tp_search(name = 'Poa 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**

```r
tp_summary(id, key = NULL, callopts = list())
```

**Arguments**

- `id`: the taxon identifier code
- `key`: Your Tropicos API key; loads from .Rprofile.
- `callopts`: Curl options.

**Value**

A data.frame.

**Examples**

```r
## 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**

```r
tp_synonyms(id, key = NULL, callopts = list())
```

**Arguments**

- `id`: the taxon identifier code
- `key`: Your Tropicos API key; loads from .Rprofile.
- `callopts`: Further args passed on to httr::GET
This function will return all ClassificationBank data pertaining to a particular ClassificationBankID

Usage

```r
ubio_classification(hierarchiesID = NULL, childrenFlag = 0,
  ancestryFlag = 0, justificationsFlag = 0, synonymsFlag = 0,
  keyCode = NULL, callopts = list())
```

Arguments

- **hierarchiesID** (string) - ClassificationBank identifier for the record you wish to receive
- **childrenFlag** (0 or 1) to include the taxon’s children taxa
- **ancestryFlag** (0 or 1) to include the taxon’s taxonomic ancestry
- **justificationsFlag** (0 or 1) to include the bibliographic references
- **synonymsFlag** (0 or 1) to include the taxon’s synonymous taxa
- **callopts** Parameters passed on to http::GET call.

Value

A list of four data.frame’s, one for the name itself, one for synonyms, one for vernacular names, and one for citations.
ubio_classification_search

**Examples**

```r
## Not run:
ubio_classification(hierarchiesID = 2483153)
ubio_classification(hierarchiesID = 2483153, childrenFlag=1)
ubio_classification(hierarchiesID = 2483153, ancestryFlag=1)

## End(Not run)
```

---

**Description**

This function will return ClassificationBankIDs (hierarchiesIDs) that refer to the given NamebankID

**Usage**

```r
ubio_classification_search(namebankID = NULL, classificationTitleID = NULL,
keyCode = NULL, callopts = list())
```

**Arguments**

- `namebankID` (character) NameBank identifier you wish to search for in ClassificationBank
- `classificationTitleID` Include if you only wish to search within a particular classification
- `keyCode` Your uBio API key; loads from .Rprofile. If you don’t have one, obtain one at http://www.ubio.org/index.php?pagename=form.
- `callopts` Parameters passed on to httr::GET call.

**Value**

A data.frame with columns classificationBankID, classificationTitleID, and classificationTitle

**Examples**

```r
## Not run:
ubio_classification_search(namebankID = 3070378)

## End(Not run)
```
**ubio_id**  
*Search uBio by namebank ID.*

**Description**

Search uBio by namebank ID.

**Usage**

```r
ubio_id(namebankID = NULL, keyCode = NULL, callopts = list())
```

**Arguments**

- `namebankID` (character) uBio namebank ID
- `callopts` Parameters passed on to httr::GET call.

**Value**

A list of four data.frame’s, one for the name itself, one for synonyms, one for vernacular names, and one for citations.

**Examples**

```r
## Not run:
ubio_id(namebankID = 2483153)
ubio_id(namebankID = 105509)
ubio_id(namebankID = 2843601)
ubio_id(namebankID = 2478181)

# Pass in curl options
library("httr")
ubio_id(namebankID = 2478181, callopts=verbose())
ubio_id(namebankID = 2478181, callopts=timeout(3))

## End(Not run)
```
ubio_search  

This function will return NameBankIDs that match given search terms

Description

This function will return NameBankIDs that match given search terms

Usage

ubio_search(searchName = NULL, searchAuth = NULL, searchYear = NULL, 
order = NULL, sci = 1, vern = 1, keyCode = NULL, callopts = list())

Arguments

searchName (character) - term to search within name string
searchAuth (character) - term to search within name authorship
searchYear (character) - term to search within name year
order (character) - (name or namebankID) field by which the results will be sorted (default is namebankID)
sci (integer) - 0 (no) or 1 (yes; default) to include scientific name results (default is all)
vern (integer) - 0 (no) or 1 (yes; default) to include common name (vernacular) results
keyCode (character) Your uBio API key; loads from .Rprofile. If you don’t have one, obtain one at http://www.ubio.org/index.php?pagename=form.
callopts (list) Parameters passed on to httr::GET call.

Value

A data.frame.

Examples

## Not run:
ubio_search(searchName = 'elephant')
ubio_search(searchName = 'elephant', sci = 1, vern = 0)
ubio_search(searchName = 'Astragalus aduncus', sci = 1, vern = 0)
ubio_search(searchName = 'puma concolor', sci=1, vern=0)

## End(Not run)
ubio_synonyms

Search uBio for taxonomic synonyms by hierarchiesID.

Description

Search uBio for taxonomic synonyms by hierarchiesID.

Usage

ubio_synonyms(hierarchiesID = NULL, keyCode = NULL, callopts = list())

Arguments

hierarchiesID you must include the hierarchiesID (ClassificationBankID) to receive the classification synonyms
keyCode Your uBio API key; loads from .Rprofile. If you don’t have one, obtain one at http://www.ubio.org/index.php?pagename=form.
callopts Parameters passed on to httr::GET call.

Value

A data.frame.

Examples

## Not run:
ubio_synonyms(hierarchiesID = 4091702)
ubio_synonyms(hierarchiesID = 2483153)
ubio_synonyms(hierarchiesID = 2465599)
ubio_synonyms(hierarchiesID = 1249021)
ubio_synonyms(hierarchiesID = 4069372)

## End(Not run)

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) or Catalogue of Life (col). There is no method exposed by itis or col for getting taxa at a specific taxonomic rank, so we do it ourselves inside the function.
Usage

upstream(...)

## Default S3 method:
upstream(x, db = NULL, upto = NULL, ...)

## S3 method for class 'tsn'
upstream(x, db = NULL, upto = NULL, ...)

## S3 method for class 'colid'
upstream(x, db = NULL, upto = NULL, ...)

## S3 method for class 'ids'
upstream(x, db = NULL, upto = NULL, ...)

Arguments

... Further args passed on to itis_downstream or col_downstream
x character; taxons to query.
db character; database to query. One or both of itis, col.
upto What taxonomic rank to go down to. One of: 'Superkingdom', 'Kingdom', 'Subkingdom', 'Infra
kingdom', 'Phylum', 'Division', 'Subphylum', 'Subdivision', 'Infraclass', 'Class', 'Subclass', 'Infraclass', 'Superorder',
'Order', 'Suborder', 'Infraorder', 'Superfamily', 'Family', 'Subfamily', 'Tribe', 'Subtribe', 'Genus', 'Subgenus',
'Section', 'Subsection', 'Species', 'Subspecies', 'Variety', 'Form', 'Subvariety', 'Race',
'Stirp', 'Morph', 'Aberration', 'Subform', 'Unspecified'

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:
## col
upstream("Pinus contorta", db = 'col', upto = 'Genus') # get all genera at one level up
upstream("Abies", db = 'col', upto = 'Genus') # goes to same level, Abies is a genus
upstream("Pinus contorta", db = 'col', upto = 'Family')
upstream("Poa annua", db = 'col', upto = 'Family')
upstream("Poa annua", db = 'col', upto = 'Order')

## itis
upstream(x='Pinus contorta', db = 'itis', upto = 'Genus')

## both
upstream(get_ids("Pinus contorta", db = c('col','itis')), upto = 'Genus')

## End(Not run)
vascan_search

Search the CANADENSYS Vascan API.

Description

For more information, see http://data.canadensys.net/vascan/search.

Usage

vascan_search(q, format = "json", raw = FALSE, callopts = list())

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.
callopts        (list) Further args passed on to hht::GET.

Value

json, xml or a list.

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

References

API docs http://data.canadensys.net/vascan/api. You can also download bulk data http://data.canadensys.net/ipt/resource.do?r=vascan&request_locale=en

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("XML")
xmlParse(d)

# lots of names, in this case 50
splist <- names_list(rank='species', size=50)
vascan_search(q = splist)

## End(Not run)
Index

*Topic **data**
apg_families, 5
apg_genera, 5
apg_orders, 6
plantGenusNames, 89
plantNames, 90
rank_ref, 91
theplantlist, 103
*Topic **globalnamesindex**
gni_details, 53
gni_search, 55
*Topic **names**
gni_details, 53
gni_search, 55
gnr_datasources, 57
gnr_resolve, 58
vascan_search, 119
*Topic **package**
taxizeMpackage, 4
*Topic **resolve**
gnr_datasources, 57
gnr_resolve, 58
*Topic **taxonomy**
gni_details, 53
gni_search, 55
gnr_datasources, 57
gnr_resolve, 58
vascan_search, 119
apg_families, 5
apg_genera, 5
apg_lookup, 6
apg_orders, 6
as.boldid, 31
as.boldid(get_boldid), 30
as.colid, 33
as.colid(get_colid), 32
as.data.frame.boldid(get_boldid), 30
as.data.frame.colid(get_colid), 32
as.data.frame.eolid(get_eolid), 35
as.data.frame.gbifid(get_gbifid), 37
as.data.frame.nbnid(get_nbnid), 37
as.data.frame.tpsid(get_tpsid), 43
as.data.frame.tsn(get_tsn), 45
as.data.frame.uboid(get_uboid), 48
as.data.frame.uid(get_uid), 50
as.eolid, 36
as.eolid(get_eolid), 35
as.gbifid, 38
as.gbifid(get_gbifid), 37
as.nbnid, 42
as.nbnid(get_nbnid), 41
as.tpsid, 44
as.tpsid(get_tpsid), 43
as.tsn, 46
as.tsn(get_tsn), 45
as.uboid, 49
as.uboid(get_uboid), 48
as.uid, 51
as.uid(get_uid), 50
bold_ping(ping), 87
bold_search, 7
cbind.classification(classification), 11
cbind.classification_ids
(classification), 11
children, 8, 80
class2tree, 10
classification, 11, 31, 47, 51, 97, 102, 103
col_children, 9, 15
col_classification, 97
col_downstream, 16
col_ping(ping), 87
col_search, 17
comm2sci, 19, 93
downstream, 8, 20
eol_dataobjects, 22
get_gbifid
get_eolid_(
  get_gbifid_,
  get_tpsid_,
  get_nbnid_(
    get_genes,
    get_gbifid_,
    get_tpsid_,
    get_nbnid_,
    get_ids_,
    get_ids,
    get_uid_,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
    get_gene_avail,
    get_tsn_,
    get_tsn_,
    get_tpsid_,
    get_guild,,
    get_eolid_,
    get_eolid_,
    get_eolid_,
    get_gbifid_,
    get_gbifid_,
    get_gbifid_,
    get_genes,
gnr_datasources, 54, 56, 57, 58, 59
gnr_resolve, 57, 58
iplant_resolve, 59
ipni_ping (ping), 87
ipni_search, 60
itis-api, 62
itis_acceptname, 64
itis_downstream, 64, 67
itis_getrecord, 66
itis_hierarchy, 66
itis_kingdomnames, 67
itis_lsid, 68
itis_name, 69
itis_native, 69
itis_ping (ping), 87
itisrefs, 70
itis_searchcommon, 63, 71
itis_taxrank, 71
itis_terms, 72
iucn_getname, 73
iucn_status, 73, 74, 75
iucn_status.iucn (iucn_summary), 74
iucn_summary, 73, 74, 75
names_list, 76
nbn_classification, 77
nbn_ping (ping), 87
nbn_search, 77
nbn_synonyms, 78
ncbi_children, 9, 79
ncbi_get_taxon_summary, 80, 82
ncbi_getbyid, 80, 82, 84, 97, 98
ncbi_getbyname, 81, 82, 84, 97, 98
ncbi_ping (ping), 87
ncbi_search, 81, 82, 83, 97, 98
phylomatic_format, 85
phylomatic_tree, 86, 98
ping, 87, 95
plantGenusNames, 89
plantminer, 89
plantNames, 90
plot.class2tree (class2tree), 10
print.class2tree (class2tree), 10
print.tax_agg (tax_agg), 99
rank_ref, 91, 101
rankagg, 90
rbind.classification (classification), 11
rbind.classification_ids (classification), 11
resolve, 91
sci2comm, 19, 92
scrapenames, 93
searchbycommonname, 19, 63, 93
searchbycommonnamebeginswith, 19, 63, 93
searchbycommonnameendswith, 19, 63, 93
searchbyscientificname, 63
searchforanymatch, 63
searchforanymatchpaged, 63
status_codes, 88, 95
synonyms, 95
tax_agg, 99
tax_name, 100, 101, 101, 103
tax_rank, 101, 102, 103
taxa2dist, 10, 11
taxize (taxize-package), 4
taxize-defunct, 97
taxize-deprecated, 97
taxize-package, 4
taxize_capwords, 98
taxize_cite, 99
theplantlist, 103
tnrs, 103	nrnsources, 105
tp_accnames, 109
tp_classification, 97
tp_dist, 109
tprefs, 110
tp_search, 19, 44, 93, 111
tp_summary, 112
tp_synonyms, 112
TPL, 97, 108
tpl_families, 106, 107, 108
tpl_get, 106, 106, 108
tpl_search, 97, 106, 107, 108
TPLck, 97, 108
tropicos_ping (ping), 87
ubio_classification, 113
ubio_classification_search, 114
ubio_id, 115
ubio_ping (ping), 87
ubio_search, 48, 49, 116
ubio_synonyms, 117
upstream, 117
vascan_ping (ping), 87
vascan_search, 119