Package 'taxa'

April 29, 2020

Type Package

Title Taxonomic Classes

Description Provides taxonomic classes for groupings of taxonomic names without data, and those with data. Methods provided are ``taxonomically aware'', in that they know about ordering of ranks, and methods that filter based on taxonomy also filter associated data. This package is described in the publication: ``Taxa: An R package implementing data standards and methods for taxonomic data'', Zachary S.L. Foster, Scott Chamberlain, Niklaus J. Grünwald (2018) <doi:10.12688/f1000research.14013.2>.

Version 0.3.4

Depends R (>= 3.0.2)

VignetteBuilder knitr

LazyLoad yes

LazyData yes

Encoding UTF-8

License MIT + file LICENSE

URL https://docs.ropensci.org/taxa, https://github.com/ropensci/taxa

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

Imports R6, jsonlite, dplyr, lazyeval, magrittr, tibble, knitr, rlang, stringr, crayon, tidyr, utils, taxize

Suggests roxygen2 (>= 6.0.1), testthat, rmarkdown (>= 0.9.6)

RoxygenNote 7.1.0

X-schema.org-applicationCategory Taxonomy

X-schema.org-keywords taxonomy, biology, hierarchy

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

NeedsCompilation no

Author Scott Chamberlain [aut] (<https://orcid.org/0000-0003-1444-9135>), Zachary Foster [aut, cre] (<https://orcid.org/0000-0002-5075-0948>)

Maintainer Zachary Foster <zacharyfoster1989@gmail.com>

Repository CRAN

Date/Publication 2020-04-29 19:10:02 UTC

R topics documented:

taxa-package *taxa*

Description

The `taxa` package is intended to:

- Provide a set of classes to store taxonomic data and any user-specific data associated with it
- Provide functions to convert commonly used formats to these classes
- Provide a common foundation for other packages to build on to enable an ecosystem of compatible packages dealing with taxonomic data.
- Provide generally useful functionality, such as filtering and mapping functions

Main classes

These are the classes users would typically interact with:

- taxon: A class used to define a single taxon. Many other classes in the 'taxa" package include one or more objects of this class.
- taxa: Stores one or more taxon objects. This is just a thin wrapper for a list of taxon objects.
- hierarchy: A class containing an ordered list of taxon objects that represent a hierarchical classification.
- hierarchies: A list of taxonomic classifications. This is just a thin wrapper for a list of hierarchy objects.
- taxonomy: A taxonomy composed of taxon objects organized in a tree structure. This differs from the hierarchies class in how the taxon objects are stored. Unlike a hierarchies object, each unique taxon is stored only once and the relationships between taxa are stored in an edgelist.
- taxmap: A class designed to store a taxonomy and associated user-defined data. This class builds on the taxonomy class. User defined data can be stored in the list `obj$data`, where `obj` is a taxmap object. Any number of user-defined lists, vectors, or tables mapped to taxa can be manipulated in a cohesive way such that relationships between taxa and data are preserved.

Minor classes

These classes are mostly components for the larger classes above and would not typically be used on their own.

- taxon_database: Used to store information about taxonomy databases.
- taxon_id: Used to store taxon IDs, either arbitrary or from a particular taxonomy database.
- taxon_name: Used to store taxon names, either arbitrary or from a particular taxonomy database.
- taxon_rank: Used to store taxon ranks (e.g. species, family), either arbitrary or from a particular taxonomy database.

Major manipulation functions

These are some of the more important functions used to filter data in classes that store multiple taxa, like hierarchies, taxmap, and taxonomy.

- filter_taxa: Filter taxa in a taxonomy or taxmap object with a series of conditions. Relationships between remaining taxa and user-defined data are preserved (There are many options controlling this).
- filter_obs: Filter user-defined data taxmap object with a series of conditions. Relationships between remaining taxa and user-defined data are preserved (There are many options controlling this);
- sample_n_taxa: Randomly sample taxa. Has same abilities as filter_taxa.
- sample_n_obs: Randomly sample observations. Has same abilities as filter_obs.
- mutate_obs: Add datasets or columns to datasets in the `data` list of taxmap objects.
- pick: Pick out specific taxa, while others are dropped in hierarchy and hierarchies objects.
- pop: Pop out taxa (drop them) in hierarchy and hierarchies objects.
- span: Select a range of taxa, either by two names, or relational operators in hierarchy and hierarchies objects.

Mapping functions

There are lots of functions for getting information for each taxon.

- subtaxa: Return data for the subtaxa of each taxon in an taxonomy or taxmap object.
- supertaxa: Return data for the supertaxa of each taxon in an taxonomy or taxmap object.
- roots: Return data for the roots of each taxon in an taxonomy or taxmap object.
- leaves: Return data for the leaves of each taxon in an taxonomy or taxmap object.
- obs: Return user-specific data for each taxon and all of its subtaxa in an taxonomy or taxmap object.

The kind of classes used

Note, this is mostly of interest to developers and advanced users.

The classes in the `taxa` package are mostly R6 classes (R6Class). A few of the simpler ones (taxa and hierarchies) are S3 instead. R6 classes are different than most R objects because they are mutable (e.g. A function can change its input without returning it). In this, they are more similar to class systems in object-oriented languages like python. As in other object-oriented class systems, functions are thought to "belong" to classes (i.e. the data), rather than functions existing independently of the data. For example, the function `print` in R exists apart from what it is printing, although it will change how it prints based on what the class of the data is that is passed to it. In fact, a user can make a custom print method for their own class by defining a function called `print.myclassname`. In contrast, the functions that operate on R6 functions are "packaged" with the data they operate on. For example, a print method of an object for an R6 class might be called like `my_data$print()` instead of `print(my_data)`.

The two ways to call functions

Note, you will need to read the previous section to fully understand this one.

Since the R6 function syntax (e.g. `my_data$print()`) might be confusing to many R users, all functions in `taxa` also have S3 versions. For example, the `filter_taxa()` function can be called on a taxmap object called `my_obj` like `my_obj$filter_taxa(...)` (the R6 syntax) or `filter_taxa(my_obj, ...)` (the S3 syntax). For some functions, these two way of calling the function can have different effect. For functions that do not returned a modified version of the input (e.g. `subtaxa()`), the two ways have identical behavior. However, functions like `filter_taxa()`, that modify their inputs, actually change the object passed to them as the first argument as well as returning that object. For example,

```
my_obj <-filter_taxa(my_obj, ...)
```

and

```
my_obj$filter_taxa(...)
```

and

```
new_obj <-my_obj$filter_taxa(...)
```

all replace `my_obj` with the filtered result, but

```
new_obj <-filter_taxa(my_obj, ...)
```

will not modify `my_obj`.

Non-standard evaluation

This is a rather advanced topic.

Like packages such as `ggplot2` and dplyr, the `taxa` package uses non-standard evaluation to allow code to be more readable and shorter. In effect, there are variables that only "exist" inside a function call and depend on what is passed to that function as the first parameter (usually a class object). For example, in the `dpylr` function `filter()`, column names can be used as if they were independent variables. See `?dpylr::filter` for examples of this. The `taxa` package builds on this idea.

For many functions that work on taxonomy or taxmap objects (e.g. filter_taxa), some functions that return per-taxon information (e.g. `taxon_names()`) can be referred to by just the name of the function. When one of these functions are referred to by name, the function is run on the relevant object and its value replaces the function name. For example,

```
new_obj <-filter_taxa(my_obj,taxon_names == "Bacteria")
```

is identical to:

```
new_obj <-filter_taxa(my_obj,taxon_names(my_obj) == "Bacteria")
```

which is identical to:

```
new_obj <-filter_taxa(my_obj,my_obj$taxon_names() == "Bacteria")
```

which is identical to:

```
my_names <-taxon_names(my_obj)
new_obj <-filter_taxa(my_obj,my_names == "Bacteria")
```

For `taxmap` objects, you can also use names of user defined lists, vectors, and the names of columns in user-defined tables that are stored in the `obj$data` list. See `filter_taxa()` for examples. You can even add your own functions that are called by name by adding them to the `obj$funcs` list. For any object with functions that use non-standard evaluation, you can see what values can be used with `all_names()` like `all_names(obj)`.

Dependencies and inspiration

Various elements of the taxa package were inspired by the dplyr and taxize packages. This package started as parts of the metacoder and binomen packages. There are also many dependencies that make taxa possible.

Feedback and contributions

Find a problem? Have a suggestion? Have a question? Please submit an issue at our GitHub repository:

https://github.com/ropensci/taxa/issues

A GitHub account is free and easy to set up. We welcome feedback! If you don't want to use GitHub for some reason, feel free to email us. We do prefer posting to github since it allows others that might have the same issue to see our conversation. It also helps us keep track of what problems we need to address.

Want to contribute code or make a change to the code? Great, thank you! Please fork our GitHub repository and submit a pull request.

For more information

Checkout the vignette (browseVignettes("taxa")) for detailed introduction and examples.

Author(s)

Scott Chamberlain <myrmecocystus+r@gmail.com>

Zachary Foster <zacharyfoster1989@gmail.com>

---

all_names *Return names of data in* taxonomy() *or* taxmap()

Description

Return the names of data that can be used with functions in the taxa package that use non-standard evaluation (NSE), like filter_taxa().

```
obj$all_names(tables = TRUE, funcs = TRUE,
  others = TRUE, warn = FALSE)
all_names(obj, tables = TRUE, funcs = TRUE,
  others = TRUE, warn = FALSE)
```

Arguments

obj	(taxonomy() or taxmap()) The object containing taxon information to be queried.
tables	This option only applies to taxmap() objects. If TRUE, include the names of columns of tables in obj$data

funcs	This option only applies to taxmap() objects. If TRUE, include the names of user-definable functions in obj$funcs.
others	This option only applies to taxmap() objects. If TRUE, include the names of data in obj$data besides tables.
builtin_funcs	This option only applies to taxmap() objects. If TRUE, include functions like n_supertaxa() that provide information for each taxon.
warn	option only applies to taxmap() objects. If TRUE, warn if there are duplicate names. Duplicate names make it unclear what data is being referred to.

Value

character

See Also

Other NSE helpers: data_used, get_data(), names_used

Examples

```
# Get the names of all data accesible by non-standard evaluation
all_names(ex_taxmap)

# Dont include the names of automatically included functions.
all_names(ex_taxmap, builtin_funcs = FALSE)
```

---

arrange_obs *Sort user data in taxmap() objects*

Description

Sort rows of tables or the elements of lists/vectors in the obj$data list in taxmap() objects. Any variable name that appears in all_names() can be used as if it was a vector on its own. See dplyr::arrange() for the inspiration for this function and more information. Calling the function using the obj$arrange_obs(...) style edits "obj" in place, unlike most R functions. However, calling the function using the arrange_obs(obj,...) imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$arrange_obs(data, ...)
arrange_obs(obj, data, ...)
```

Arguments

obj	An object of type taxmap().
data	Dataset names, indexes, or a logical vector that indicates which datasets in obj$data to sort If multiple datasets are sorted at once, then they must be the same length.
...	One or more expressions (e.g. column names) to sort on.
target	DEPRECIATED. use "data" instead.

Value

An object of type taxmap()

See Also

Other taxmap manipulation functions: arrange_taxa(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_obs(), sample_frac_taxa(), sample_n_obs(), sample_n_taxa(), select_obs(), transmute_obs()

Examples

```
# Sort in ascending order
arrange_obs(ex_taxmap, "info", n_legs)
arrange_obs(ex_taxmap, "foods", name)

# Sort in decending order
arrange_obs(ex_taxmap, "info", desc(n_legs))

# Sort multiple datasets at once
arrange_obs(ex_taxmap, c("info", "phylopic_ids", "foods"), n_legs)
```

---

arrange_taxa *Sort the edge list of* taxmap() *objects*

Description

Sort the edge list and taxon list in taxonomy() or taxmap() objects. See dplyr::arrange() for the inspiration for this function and more information. Calling the function using the obj$arrange_taxa(...) style edits "obj" in place, unlike most R functions. However, calling the function using the arrange_taxa(obj,...) imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$arrange_taxa(...)
arrange_taxa(obj, ...)
```

Arguments

obj	taxonomy() or taxmap()
...	One or more expressions (e.g. column names) to sort on. Any variable name that appears in all_names() can be used as if it was a vector on its own.

Value

An object of type taxonomy() or taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_obs(), sample_frac_taxa(), sample_n_obs(), sample_n_taxa(), select_obs(), transmute_obs()

Examples

```
# Sort taxa in ascending order
arrange_taxa(ex_taxmap, taxon_names)

# Sort taxa in decending order
arrange_taxa(ex_taxmap, desc(taxon_names))

# Sort using an expression. List genera first.
arrange_taxa(ex_taxmap, taxon_ranks != "genus")
```

---

branches *Get "branch" taxa*

Description

Return the "branch" taxa for a taxonomy() or taxmap() object. A branch is anything that is not a root, stem, or leaf. Its the interior of the tree after the first split starting from the roots. Can also be used to get the branches of a subset of taxa.

```
obj$branches(subset = NULL, value = "taxon_indexes")
branches(obj, subset = NULL, value = "taxon_indexes")
```

Arguments

obj	The taxonomy() or taxmap() object containing taxon information to be queried.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes used to subset the tree prior to determining branches. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own. Note that branches are determined after the filtering, so a given taxon might be a branch on the unfiltered tree, but not a branch on the filtered tree.

value — What data to return. This is usually the name of column in a table in obj$data. Any result of all_names() can be used, but it usually only makes sense to use data that corresponds to taxa 1:1, such as taxon_ranks(). By default, taxon indexes are returned.

Value

character

See Also

Other taxonomy indexing functions: internodes(), leaves(), roots(), stems(), subtaxa(), supertaxa()

Examples

```
# Return indexes of branch taxa
branches(ex_taxmap)

# Return indexes for a subset of taxa
branches(ex_taxmap, subset = 2:17)
branches(ex_taxmap, subset = n_obs > 1)

# Return something besides taxon indexes
branches(ex_taxmap, value = "taxon_names")
```

---

classifications *Get classifications of taxa*

Description

Get character vector classifications of taxa in an object of type taxonomy() or taxmap() composed of data associated with taxa. Each classification is constructed by concatenating the data of the given taxon and all of its supertaxa.

```
obj$classifications(value = "taxon_names", sep = ";")
classifications(obj, value = "taxon_names", sep = ";")
```

Arguments

obj — (taxonomy() or taxmap())

value — What data to return. Any result of all_names(obj) can be used, but it usually only makes sense to data that corresponds to taxa 1:1, such as taxon_ranks(). By default, taxon indexes are returned.

sep — (character of length 1) The character(s) to place between taxon IDs

Value

character

See Also

Other taxonomy data functions: id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Defualt settings returns taxon names separated by ;
classifications(ex_taxmap)

# Other values can be returned besides taxon names
classifications(ex_taxmap, value = "taxon_ids")

# The separator can also be changed
classifications(ex_taxmap, value = "taxon_ranks", sep = "||")
```

---

database_list *Database list*

Description

The list of known databases. Not currently used much, but will be when we add more check for taxon IDs and taxon ranks from particular databases.

Usage

```
database_list
```

Format

An object of class list of length 8.

Details

List of databases with pre-filled details, where each has the format:

- url: A base URL for the database source.
- description: Description of the database source.
- id regex: identifier regex.

See Also

taxon_database

Examples

```
database_list
database_list$ncbi
database_list$ncbi$name
database_list$ncbi$description
database_list$ncbi$url
```

`extract_tax_data` *Extracts taxonomy info from vectors with regex*

Description

Convert taxonomic information in a character vector into a `taxmap()` object. The location and identity of important information in the input is specified using a regular expression with capture groups and a corresponding key. An object of type `taxmap()` is returned containing the specified information. See the `key` option for accepted sources of taxonomic information.

Usage

```
extract_tax_data(
  tax_data,
  key,
  regex,
  class_key = "taxon_name",
  class_regex = "(.*)",
  class_sep = NULL,
  sep_is_regex = FALSE,
  class_rev = FALSE,
  database = "ncbi",
  include_match = FALSE,
  include_tax_data = TRUE
)
```

Arguments

`tax_data`	A vector from which to extract taxonomy information.
`key`	(`character`) The identity of the capturing groups defined using `regex`. The length of `key` must be equal to the number of capturing groups specified in `regex`. Any names added to the terms will be used as column names in the output. Only `"info"` can be used multiple times. Each term must be one of those described below: • `taxon_id`: A unique numeric id for a taxon for a particular `database` (e.g. ncbi accession number). Requires an internet connection. • `taxon_name`: The name of a taxon (e.g. "Mammalia" or "Homo sapiens"). Not necessarily unique, but interpretable by a particular `database`. Requires an internet connection.

- `fuzzy_name`: The name of a taxon, but check for misspellings first. Only use if you think there are misspellings. Using `"taxon_name"` is faster.
- `class`: A list of taxon information that constitutes the full taxonomic classification (e.g. "K_Mammalia;P_Carnivora;C_Felidae"). Individual taxa are separated by the `class_sep` argument and the information is parsed by the `class_regex` and `class_key` arguments.
- `seq_id`: Sequence ID for a particular database that is associated with a taxonomic classification. Currently only works with the "ncbi" database.
- `info`: Arbitrary taxon info you want included in the output. Can be used more than once.

`regex` (`character` of length 1) A regular expression with capturing groups indicating the locations of relevant information. The identity of the information must be specified using the `key` argument.

`class_key` (`character` of length 1) The identity of the capturing groups defined using `class_regex`. The length of `class_key` must be equal to the number of capturing groups specified in `class_regex`. Any names added to the terms will be used as column names in the output. Only `"info"` can be used multiple times. Each term must be one of those described below:

- `taxon_name`: The name of a taxon. Not necessarily unique.
- `taxon_rank`: The rank of the taxon. This will be used to add rank info into the output object that can be accessed by `out$taxon_ranks()`.
- `info`: Arbitrary taxon info you want included in the output. Can be used more than once.

`class_regex` (`character` of length 1) A regular expression with capturing groups indicating the locations of data for each taxon in the `class` term in the `key` argument. The identity of the information must be specified using the `class_key` argument. The `class_sep` option can be used to split the classification into data for each taxon before matching. If `class_sep` is `NULL`, each match of `class_regex` defines a taxon in the classification.

`class_sep` (`character` of length 1) Used with the `class` term in the `key` argument. The character(s) used to separate individual taxa within a classification. After the string defined by the `class` capture group in `regex` is split by `class_sep`, its capture groups are extracted by `class_regex` and defined by `class_key`. If `NULL`, every match of `class_regex` is used instead with first splitting by `class_sep`.

`sep_is_regex` (`TRUE`/`FALSE`) Whether or not `class_sep` should be used as a regular expression.

`class_rev` (`logical` of length 1) Used with the `class` term in the `key` argument. If `TRUE`, the order of taxon data in a classification is reversed to be specific to broad.

`database` (`character` of length 1) The name of the database that patterns given in `parser` will apply to. Valid databases include "ncbi", "itis", "eol", "col", "tropicos", "nbn", and "none". `"none"` will cause no database to be queried; use this if you want to not use the internet. NOTE: Only `"ncbi"` has been tested extensively so far.

`include_match` (`logical` of length 1) If `TRUE`, include the part of the input matched by `regex` in the output object.

include_tax_data
: (TRUE/FALSE) Whether or not to include tax_data as a dataset.

Value

Returns an object of type taxmap()

Failed Downloads

If you have invalid inputs or a download fails for another reason, then there will be a "unknown" taxon ID as a placeholder and failed inputs will be assigned to this ID. You can remove these using filter_taxa() like so: filter_taxa(result,taxon_ids != "unknown"). Add drop_obs = FALSE if you want the input data, but want to remove the taxon.

See Also

Other parsers: lookup_tax_data(), parse_edge_list(), parse_tax_data()

Examples

```
## Not run:

  # For demonstration purposes, the following example dataset has all the
  # types of data that can be used, but any one of them alone would work.
  raw_data <- c(
 ">id:AB548412-tid:9689-Panthera leo-tax:K_Mammalia;P_Carnivora;C_Felidae;G_Panthera;S_leo",
 ">id:FJ358423-tid:9694-Panthera tigris-tax:K_Mammalia;P_Carnivora;C_Felidae;G_Panthera;S_tigris",
 ">id:DQ334818-tid:9643-Ursus americanus-tax:K_Mammalia;P_Carnivora;C_Felidae;G_Ursus;S_americanus"
  )

  # Build a taxmap object from classifications
  extract_tax_data(raw_data,
                   key = c(my_seq = "info", my_tid = "info", org = "info", tax = "class"),
                     regex = "^>id:(.+)-tid:(.+)-(.+)-tax:(.+)$",
                     class_sep = ";", class_regex = "^(.+)_(.+)$",
                     class_key = c(my_rank = "info", tax_name = "taxon_name"))

  # Build a taxmap object from taxon ids
  # Note: this requires an internet connection
  extract_tax_data(raw_data,
                  key = c(my_seq = "info", my_tid = "taxon_id", org = "info", tax = "info"),
                     regex = "^>id:(.+)-tid:(.+)-(.+)-tax:(.+)$")

  # Build a taxmap object from ncbi sequence accession numbers
  # Note: this requires an internet connection
  extract_tax_data(raw_data,
                   key = c(my_seq = "seq_id", my_tid = "info", org = "info", tax = "info"),
                     regex = "^>id:(.+)-tid:(.+)-(.+)-tax:(.+)$")

  # Build a taxmap object from taxon names
  # Note: this requires an internet connection
```

```
  extract_tax_data(raw_data,
              key = c(my_seq = "info", my_tid = "info", org = "taxon_name", tax = "info"),
                   regex = "^>id:(.+)-tid:(.+)-(.+)-tax:(.+)$")

## End(Not run)
```

`ex_hierarchies` *An example hierarchies object*

Description

An example hierarchies object built from the ground up.

Format

A `hierarchies()` object.

Source

Created from the example code in the `hierarchies()` documentation.

See Also

Other taxa-datasets: `ex_hierarchy1`, `ex_hierarchy2`, `ex_hierarchy3`, `ex_taxmap`, `ex_taxonomy`

`ex_hierarchy1` *An example Hierarchy object*

Description

An example Hierarchy object built from the ground up.

Format

A `hierarchy()` object with

- name: Poaceae / rank: family / id: 4479
- name: Poa / rank: genus / id: 4544
- name: Poa annua / rank: species / id: 93036

Based on NCBI taxonomic classification

Source

Created from the example code in the `hierarchy()` documentation.

See Also

Other taxa-datasets: `ex_hierarchies`, `ex_hierarchy2`, `ex_hierarchy3`, `ex_taxmap`, `ex_taxonomy`

ex_hierarchy2 *An example Hierarchy object*

Description

An example Hierarchy object built from the ground up.

Format

A `hierarchy()` object with

- name: Felidae / rank: family / id: 9681
- name: Puma / rank: genus / id: 146712
- name: Puma concolor / rank: species / id: 9696

Based on NCBI taxonomic classification

Source

Created from the example code in the `hierarchy()` documentation.

See Also

Other taxa-datasets: `ex_hierarchies`, `ex_hierarchy1`, `ex_hierarchy3`, `ex_taxmap`, `ex_taxonomy`

ex_hierarchy3 *An example Hierarchy object*

Description

An example Hierarchy object built from the ground up.

Format

A `hierarchy()` object with

- name: Chordata / rank: phylum / id: 158852
- name: Vertebrata / rank: subphylum / id: 331030
- name: Teleostei / rank: class / id: 161105
- name: Salmonidae / rank: family / id: 161931
- name: Salmo / rank: genus / id: 161994
- name: Salmo salar / rank: species / id: 161996

Based on ITIS taxonomic classification

Source

Created from the example code in the `hierarchy()` documentation.

See Also

Other taxa-datasets: `ex_hierarchies`, `ex_hierarchy1`, `ex_hierarchy2`, `ex_taxmap`, `ex_taxonomy`

---

ex_taxmap *An example taxmap object*

Description

An example taxmap object built from the ground up. Typically, data stored in taxmap would be parsed from an input file, but this data set is just for demonstration purposes.

Format

A `taxmap()` object.

Source

Created from the example code in the `taxmap()` documentation.

See Also

Other taxa-datasets: `ex_hierarchies`, `ex_hierarchy1`, `ex_hierarchy2`, `ex_hierarchy3`, `ex_taxonomy`

---

ex_taxonomy *An example Taxonomy object*

Description

An example Taxonomy object built from the ground up.

Format

A `taxonomy()` object.

Source

Created from the example code in the `taxonomy()` documentation.

See Also

Other taxa-datasets: `ex_hierarchies`, `ex_hierarchy1`, `ex_hierarchy2`, `ex_hierarchy3`, `ex_taxmap`

`filtering-helpers` *Taxonomic filtering helpers*

Description

Taxonomic filtering helpers

Usage

```
ranks(...)

nms(...)

ids(...)
```

Arguments

`...`	quoted rank names, taxonomic names, taxonomic ids, or any of those with supported operators (See **Supported Relational Operators** below)

How do these functions work?

Each function assigns some metadata so we can more easily process your query downstream. In addition, we check for whether you've used any relational operators and pull those out to make downstream processing easier

The goal of these functions is to make it easy to combine queries based on each of rank names, taxonomic names, and taxonomic ids.

These are designed to be used inside of `pop()`, `pick()`, `span()`. Inside of those functions, we figure out what rank names you want to filter on, then check against a reference dataset (ranks_ref) to allow ordered queries like *I want all taxa between Class and Genus*. If you provide rank names, we just use those, then do the filtering you requested. If you provide taxonomic names or ids we figure out what rank names you are referring to, then we can proceed as in the previous sentence.

Supported Relational Operators

- > all items above rank of x
- >= all items above rank of x, inclusive
- < all items below rank of x
- <= all items below rank of x, inclusive

ranks

Ranks can be any character string in the set of acceptable rank names.

nms

nms is named to avoid using names which would collide with the fxn base::names() in Base R. Can pass in any character taxonomic names.

ids

Ids are any alphanumeric taxonomic identifier. Some database providers use all digits, but some use a combination of digits and characters.

Note

NSE is not supported at the moment, but may be in the future

Examples

```
ranks("genus")
ranks("order", "genus")
ranks("> genus")

nms("Poaceae")
nms("Poaceae", "Poa")
nms("< Poaceae")

ids(4544)
ids(4544, 4479)
ids("< 4479")
```

---

filter_obs *Filter observations with a list of conditions*

Description

Filter data in a taxmap() object (in obj$data) with a set of conditions. See dplyr::filter() for the inspiration for this function and more information. Calling the function using the obj$filter_obs(...) style edits "obj" in place, unlike most R functions. However, calling the function using the filter_obs(obj,...) imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$filter_obs(data, ..., drop_taxa = FALSE, drop_obs = TRUE,
               subtaxa = FALSE, supertaxa = TRUE, reassign_obs = FALSE)
filter_obs(obj, data, ..., drop_taxa = FALSE, drop_obs = TRUE,
           subtaxa = FALSE, supertaxa = TRUE, reassign_obs = FALSE)
```

Arguments

obj	An object of type taxmap()
data	Dataset names, indexes, or a logical vector that indicates which datasets in obj$data to filter. If multiple datasets are filterd at once, then they must be the same length.
...	One or more filtering conditions. Any variable name that appears in all_names() can be used as if it was a vector on its own. Each filtering condition can be one of two things: • integer: One or more dataset indexes. • logical: A TRUE/FALSE vector of length equal to the number of items in the dataset.
drop_taxa	(logical of length 1) If FALSE, preserve taxa even if all of their observations are filtered out. If TRUE, remove taxa for which all observations were filtered out. Note that only taxa that are unobserved due to this filtering will be removed; there might be other taxa without observations to begin with that will not be removed.
drop_obs	(logical) This only has an effect when drop_taxa is TRUE. When TRUE, observations for other data sets (i.e. not data) assigned to taxa that are removed when filtering data are also removed. Otherwise, only data for taxa that are not present in all other data sets will be removed. This option can be either simply TRUE/FALSE, meaning that all data sets will be treated the same, or a logical vector can be supplied with names corresponding one or more data sets in obj$data. For example, c(abundance = TRUE, stats = FALSE) would remove observations in obj$data$abundance, but not in obj$data$stats.
subtaxa	(logical or numeric of length 1) This only has an effect when drop_taxa is TRUE. If TRUE, include subtaxa of taxa passing the filter. Positive numbers indicate the number of ranks below the target taxa to return. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
supertaxa	(logical or numeric of length 1) This only has an effect when drop_taxa is TRUE. If TRUE, include supertaxa of taxa passing the filter. Positive numbers indicate the number of ranks above the target taxa to return. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
reassign_obs	(logical) This only has an effect when drop_taxa is TRUE. If TRUE, observations assigned to removed taxa will be reassigned to the closest supertaxon that passed the filter. If there are no supertaxa of such an observation that passed the filter, they will be filtered out if drop_obs is TRUE. This option can be either simply TRUE/FALSE, meaning that all data sets will be treated the same, or a logical vector can be supplied with names corresponding one or more data sets in obj$data. For example, c(abundance = TRUE, stats = FALSE) would reassign observations in obj$data$abundance, but not in obj$data$stats.
target	DEPRECIATED. use "data" instead.

Value

An object of type taxmap()

See Also

Other taxmap manipulation functions: `arrange_obs()`, `arrange_taxa()`, `filter_taxa()`, `mutate_obs()`, `sample_frac_obs()`, `sample_frac_taxa()`, `sample_n_obs()`, `sample_n_taxa()`, `select_obs()`, `transmute_obs()`

Examples

```
# Filter by row index
filter_obs(ex_taxmap, "info", 1:2)

# Filter by TRUE/FALSE
filter_obs(ex_taxmap, "info", dangerous == FALSE)
filter_obs(ex_taxmap, "info", dangerous == FALSE, n_legs > 0)
filter_obs(ex_taxmap, "info", n_legs == 2)

# Remove taxa whose obserservations were filtered out
filter_obs(ex_taxmap, "info", n_legs == 2, drop_taxa = TRUE)

# Preserve other data sets while removing taxa
filter_obs(ex_taxmap, "info", n_legs == 2, drop_taxa = TRUE,
           drop_obs = c(abund = FALSE))

# When filtering taxa, do not return supertaxa of taxa that are preserved
filter_obs(ex_taxmap, "info", n_legs == 2, drop_taxa = TRUE,
           supertaxa = FALSE)

# Filter multiple datasets at once
filter_obs(ex_taxmap, c("info", "phylopic_ids", "foods"), n_legs == 2)
```

---

filter_taxa *Filter taxa with a list of conditions*

Description

Filter taxa in a `taxonomy()` or `taxmap()` object with a series of conditions. Any variable name that appears in `all_names()` can be used as if it was a vector on its own. See `dplyr::filter()` for the inspiration for this function and more information. Calling the function using the `obj$filter_taxa(...)` style edits "obj" in place, unlike most R functions. However, calling the function using the `filter_taxa(obj,...)` imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
filter_taxa(obj, ..., subtaxa = FALSE, supertaxa = FALSE,
  drop_obs = TRUE, reassign_obs = TRUE, reassign_taxa = TRUE,
  invert = FALSE, keep_order = TRUE)
obj$filter_taxa(..., subtaxa = FALSE, supertaxa = FALSE,
  drop_obs = TRUE, reassign_obs = TRUE, reassign_taxa = TRUE,
  invert = FALSE, keep_order = TRUE)
```

Arguments

obj	An object of class taxonomy() or taxmap()
...	One or more filtering conditions. Any variable name that appears in all_names() can be used as if it was a vector on its own. Each filtering condition must resolve to one of three things: • character: One or more taxon IDs contained in obj$edge_list$to • integer: One or more row indexes of obj$edge_list • logical: A TRUE/FALSE vector of length equal to the number of rows in obj$edge_list • NULL: ignored
subtaxa	(logical or numeric of length 1) If TRUE, include subtaxa of taxa passing the filter. Positive numbers indicate the number of ranks below the target taxa to return. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
supertaxa	(logical or numeric of length 1) If TRUE, include supertaxa of taxa passing the filter. Positive numbers indicate the number of ranks above the target taxa to return. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
drop_obs	(logical) This option only applies to taxmap() objects. If FALSE, include observations (i.e. user-defined data in obj$data) even if the taxon they are assigned to is filtered out. Observations assigned to removed taxa will be assigned to NA. This option can be either simply TRUE/FALSE, meaning that all data sets will be treated the same, or a logical vector can be supplied with names corresponding one or more data sets in obj$data. For example, c(abundance = FALSE, stats = TRUE) would include observations whose taxon was filtered out in obj$data$abundance, but not in obj$data$stats. See the reassign_obs option below for further complications.
reassign_obs	(logical of length 1) This option only applies to taxmap() objects. If TRUE, observations (i.e. user-defined data in obj$data) assigned to removed taxa will be reassigned to the closest supertaxon that passed the filter. If there are no supertaxa of such an observation that passed the filter, they will be filtered out if drop_obs is TRUE. This option can be either simply TRUE/FALSE, meaning that all data sets will be treated the same, or a logical vector can be supplied with names corresponding one or more data sets in obj$data. For example, c(abundance = TRUE, stats = FALSE) would reassign observations in obj$data$abundance, but not in obj$data$stats.
reassign_taxa	(logical of length 1) If TRUE, subtaxa of removed taxa will be reassigned to the closest supertaxon that passed the filter. This is useful for removing intermediate levels of a taxonomy.
invert	(logical of length 1) If TRUE, do NOT include the selection. This is different than just replacing a == with a != because this option negates the selection after taking into account the subtaxa and supertaxa options. This is useful for removing a taxon and all its subtaxa for example.
keep_order	(logical of length 1) If TRUE, keep relative order of taxa not filtered out. For example, the result of filter_taxa(ex_taxmap, 1:3) and filter_taxa(ex_taxmap, 3:1) would be the same. Does not affect dataset order, only taxon order. This is useful for maintaining order correspondence with a dataset that has one value per taxon.

Value

An object of type `taxonomy()` or `taxmap()`

See Also

Other taxmap manipulation functions: `arrange_obs()`, `arrange_taxa()`, `filter_obs()`, `mutate_obs()`, `sample_frac_obs()`, `sample_frac_taxa()`, `sample_n_obs()`, `sample_n_taxa()`, `select_obs()`, `transmute_obs()`

Examples

```
# Filter by index
filter_taxa(ex_taxmap, 1:3)

# Filter by taxon ID
filter_taxa(ex_taxmap, c("b", "c", "d"))

# Fiter by TRUE/FALSE
filter_taxa(ex_taxmap, taxon_names == "Plantae", subtaxa = TRUE)
filter_taxa(ex_taxmap, n_obs > 3)
filter_taxa(ex_taxmap, ! taxon_ranks %in% c("species", "genus"))
filter_taxa(ex_taxmap, taxon_ranks == "genus", n_obs > 1)

# Filter by an observation characteristic
dangerous_taxa <- sapply(ex_taxmap$obs("info"),
                         function(i) any(ex_taxmap$data$info$dangerous[i]))
filter_taxa(ex_taxmap, dangerous_taxa)

# Include supertaxa
filter_taxa(ex_taxmap, 12, supertaxa = TRUE)
filter_taxa(ex_taxmap, 12, supertaxa = 2)

# Include subtaxa
filter_taxa(ex_taxmap, 1, subtaxa = TRUE)
filter_taxa(ex_taxmap, 1, subtaxa = 2)

# Dont remove rows in user-defined data corresponding to removed taxa
filter_taxa(ex_taxmap, 2, drop_obs = FALSE)
filter_taxa(ex_taxmap, 2, drop_obs = c(info = FALSE))

# Remove a taxon and it subtaxa
filter_taxa(ex_taxmap, taxon_names == "Mammalia",
            subtaxa = TRUE, invert = TRUE)
```

`get_data` *Get data in a taxmap object by name*

Description

Given a vector of names, return a list of data (usually lists/vectors) contained in a `taxonomy()` or `taxmap()` object. Each item will be named by taxon ids when possible.

```
obj$get_data(name = NULL, ...)
get_data(obj, name = NULL, ...)
```

Arguments

`obj`	A `taxonomy()` or `taxmap()` object
`name`	(`character`) Names of data to return. If not supplied, return all data listed in `all_names()`.
`...`	Passed to `all_names()`. Used to filter what kind of data is returned (e.g. columns in tables or function output?) if `name` is not supplied or what kinds are allowed if `name` is supplied.

Value

`list` of vectors or lists. Each vector or list will be named by associated taxon ids if possible.

See Also

Other NSE helpers: `all_names()`, `data_used`, `names_used`

Examples

```
# Get specific values
get_data(ex_taxmap, c("reaction", "n_legs", "taxon_ranks"))

# Get all values
get_data(ex_taxmap)
```

`get_dataset` *Get a data set from a taxmap object*

Description

Get a data set from a taxmap object and complain if it does not exist.

Arguments

`obj`	A taxmap object
`data`	Dataset name, index, or a logical vector that indicates which dataset in `obj$data` to add columns to.

Examples

```
## Not run:
# Get data set by name
get_dataset(ex_taxmap, "info")

# Get data set by indeex_taxmap
get_dataset(ex_taxmap, 1)

# Get data set by T/F vector
get_dataset(ex_taxmap, startsWith(names(ex_taxmap$data), "i"))


## End(Not run)
```

`get_data_frame` *Get data in a taxonomy or taxmap object by name*

Description

Given a vector of names, return a table of the indicated data contained in a `taxonomy()` or `taxmap()` object.

```
obj$get_data_frame(name = NULL, ...)
get_data_frame(obj, name = NULL, ...)
```

Arguments

`obj`	A `taxonomy()` or `taxmap()` object
`name`	(`character`) Names of data to return. If not supplied, return all data listed in `all_names()`.
`...`	Passed to `all_names()`. Used to filter what kind of data is returned (e.g. columns in tables or function output?) if `name` is not supplied or what kinds are allowed if `name` is supplied.

Details

Note: This function will not work with variables in datasets in `taxmap()` objects unless their rows correspond 1:1 with all taxa.

Value

`data.frame`

Examples

```
# Get specific values
get_data_frame(ex_taxonomy, c("taxon_names", "taxon_indexes", "is_stem"))

# Get all values
get_data_frame(ex_taxonomy)
```

---

hierarchies *Make a set of many* `hierarchy()` *class objects*

Description

Make a set of many `hierarchy()` class objects. This is just a thin wrapper over a standard list.

Usage

```
hierarchies(..., .list = NULL)
```

Arguments

`...`	Any number of object of class `hierarchy()`
`.list`	Any number of object of class `hierarchy()` in a list

Value

An `R6Class` object of class `hierarchy()`

See Also

Other classes: `hierarchy()`, `taxa()`, `taxmap()`, `taxon_database()`, `taxon_id()`, `taxon_name()`, `taxon_rank()`, `taxonomy()`, `taxon()`

Examples

```
x <- taxon(
  name = taxon_name("Poaceae"),
  rank = taxon_rank("family"),
  id = taxon_id(4479)
)
y <- taxon(
  name = taxon_name("Poa"),
  rank = taxon_rank("genus"),
  id = taxon_id(4544)
)
z <- taxon(
  name = taxon_name("Poa annua"),
  rank = taxon_rank("species"),
  id = taxon_id(93036)
```

```
)
hier1 <- hierarchy(z, y, x)

a <- taxon(
  name = taxon_name("Felidae"),
  rank = taxon_rank("family"),
  id = taxon_id(9681)
)
b <- taxon(
  name = taxon_name("Puma"),
  rank = taxon_rank("genus"),
  id = taxon_id(146712)
)
c <- taxon(
  name = taxon_name("Puma concolor"),
  rank = taxon_rank("species"),
  id = taxon_id(9696)
)
hier2 <- hierarchy(c, b, a)

d <- taxon(
  name = taxon_name("Chordata"),
  rank = taxon_rank("phylum"),
  id = taxon_id(158852)
)
e <- taxon(
  name = taxon_name("Vertebrata"),
  rank = taxon_rank("subphylum"),
  id = taxon_id(331030)
)
f <- taxon(
  name = taxon_name("Teleostei"),
  rank = taxon_rank("class"),
  id = taxon_id(161105)
)
g <- taxon(
  name = taxon_name("Salmonidae"),
  rank = taxon_rank("family"),
  id = taxon_id(161931)
)
h <- taxon(
  name = taxon_name("Salmo"),
  rank = taxon_rank("genus"),
  id = taxon_id(161994)
)
i <- taxon(
  name = taxon_name("Salmo salar"),
  rank = taxon_rank("species"),
  id = taxon_id(161996)
)
hier3 <- hierarchy(d, e, f, g, h, i)

hiers <- hierarchies(hier1, hier2, hier3)
```

```
# pass into the .list parameter
hierarchies(.list = list(hier1, hier2, hier3))
```

`hierarchy` *The Hierarchy class*

Description

A class containing an ordered list of `taxon()` objects that represent a hierarchical classification.

Usage

```
hierarchy(..., .list = NULL)
```

Arguments

`...`	Any number of object of class `Taxon` or taxonomic names as character strings
`.list`	An alternate to the `...` input. Any number of object of class `taxon()` or character vectors in a list. Cannot be used with `...`.

Details

On initialization, taxa are sorted if they have ranks with a known order.

Methods

`pop(rank_names)` Remove `Taxon` elements by rank name, taxon name or taxon ID. The change happens in place, so you don't need to assign output to a new object. returns self - rank_names (character) a vector of rank names

`pick(rank_names)` Select `Taxon` elements by rank name, taxon name or taxon ID. The change happens in place, so you don't need to assign output to a new object. returns self - rank_names (character) a vector of rank names

Value

An `R6Class` object of class `Hierarchy`

See Also

Other classes: `hierarchies()`, `taxa()`, `taxmap()`, `taxon_database()`, `taxon_id()`, `taxon_name()`, `taxon_rank()`, `taxonomy()`, `taxon()`

Examples

```
(x <- taxon(
  name = taxon_name("Poaceae"),
  rank = taxon_rank("family"),
  id = taxon_id(4479)
))

(y <- taxon(
  name = taxon_name("Poa"),
  rank = taxon_rank("genus"),
  id = taxon_id(4544)
))

(z <- taxon(
  name = taxon_name("Poa annua"),
  rank = taxon_rank("species"),
  id = taxon_id(93036)
))

(res <- hierarchy(z, y, x))

res$taxa
res$ranklist

# pop off a rank
pop(res, ranks("family"))

# pick a rank
(res <- hierarchy(z, y, x))
pick(res, ranks("family"))


# null taxa
x <- taxon(NULL)
(res <- hierarchy(x, x, x))
## similar to hierarchy(), but `taxa` slot is not empty
```

---

highlight_taxon_ids *Highlight taxon ID column*

Description

Changes the font of a taxon ID column in a table print out.

Usage

```
highlight_taxon_ids(table_text, header_index, row_indexes)
```

Arguments

table_text	The print out of the table in a character vector, one element per line.
header_index	The row index that contains the table column names
row_indexes	The indexes of the rows to be formatted.

---

id_classifications *Get ID classifications of taxa*

Description

Get classification strings of taxa in an object of type taxonomy() or taxmap() composed of taxon IDs. Each classification is constructed by concatenating the taxon ids of the given taxon and its supertaxa.

```
obj$id_classifications(sep = ";")
id_classifications(obj, sep = ";")
```

Arguments

obj	(taxonomy() or taxmap())
sep	(character of length 1) The character(s) to place between taxon IDs

Value

character

See Also

Other taxonomy data functions: classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Get classifications of IDs for each taxon
id_classifications(ex_taxmap)

# Use a different seperator
id_classifications(ex_taxmap, sep = '|')
```

`internodes` *Get "internode" taxa*

Description

Return the "internode" taxa for a `taxonomy()` or `taxmap()` object. An internode is any taxon with a single immediate supertaxon and a single immediate subtaxon. They can be removed from a tree without any loss of information on the relative relationship between remaining taxa. Can also be used to get the internodes of a subset of taxa.

```
obj$internodes(subset = NULL, value = "taxon_indexes")
internodes(obj, subset = NULL, value = "taxon_indexes")
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object containing taxon information to be queried.
`subset`	Taxon IDs, TRUE/FALSE vector, or taxon indexes used to subset the tree prior to determining internodes. Default: All taxa in `obj` will be used. Any variable name that appears in `all_names()` can be used as if it was a vector on its own. Note that internodes are determined after the filtering, so a given taxon might be a internode on the unfiltered tree, but not a internode on the filtered tree.
`value`	What data to return. This is usually the name of column in a table in `obj$data`. Any result of `all_names()` can be used, but it usually only makes sense to use data that corresponds to taxa 1:1, such as `taxon_ranks()`. By default, taxon indexes are returned.

Value

`character`

See Also

Other taxonomy indexing functions: `branches()`, `leaves()`, `roots()`, `stems()`, `subtaxa()`, `supertaxa()`

Examples

```
## Not run:

# Return indexes of branch taxa
internodes(ex_taxmap)

# Return indexes for a subset of taxa
internodes(ex_taxmap, subset = 2:17)
internodes(ex_taxmap, subset = n_obs > 1)

# Return something besides taxon indexes
internodes(ex_taxmap, value = "taxon_names")
```

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

`is_branch` *Test if taxa are branches*

Description

Test if taxa are branches in a `taxonomy()` or `taxmap()` object. Branches are taxa in the interior of the tree that are not `roots()`, `stems()`, or `leaves()`.

```
obj$is_branch()
is_branch(obj)
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object.

Value

A `logical` of length equal to the number of taxa.

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_internode()`, `is_leaf()`, `is_root()`, `is_stem()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_subtaxa()`, `n_supertaxa_1()`, `n_supertaxa()`, `taxon_ids()`, `taxon_indexes()`, `taxon_names()`, `taxon_ranks()`

Examples

```
# Test which taxon IDs correspond to branches
is_branch(ex_taxmap)

# Filter out branches
filter_taxa(ex_taxmap, ! is_branch)
```

is_internode *Test if taxa are "internodes"*

Description

Test if taxa are "internodes" in a `taxonomy()` or `taxmap()` object. An internode is any taxon with a single immediate supertaxon and a single immediate subtaxon. They can be removed from a tree without any loss of information on the relative relationship between remaining taxa.

```
obj$is_internode()
is_internode(obj)
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object.

Value

A `logical` of length equal to the number of taxa.

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_branch()`, `is_leaf()`, `is_root()`, `is_stem()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_subtaxa()`, `n_supertaxa_1()`, `n_supertaxa()`, `taxon_ids()`, `taxon_indexes()`, `taxon_names()`, `taxon_ranks()`

Examples

```
# Test for which taxon IDs correspond to internodes
is_internode(ex_taxmap)

# Filter out internodes
filter_taxa(ex_taxmap, ! is_internode)
```

is_leaf *Test if taxa are leaves*

Description

Test if taxa are leaves in a `taxonomy()` or `taxmap()` object. Leaves are taxa without subtaxa, typically species.

```
obj$is_leaf()
is_leaf(obj)
```

Arguments

obj	The taxonomy() or taxmap() object.

Value

A logical of length equal to the number of taxa.

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Test which taxon IDs correspond to leaves
is_leaf(ex_taxmap)

# Filter out leaves
filter_taxa(ex_taxmap, ! is_leaf)
```

---

is_root *Test if taxa are roots*

Description

Test if taxa are roots in a taxonomy() or taxmap() object. Roots are taxa without supertaxa, typically things like "Bacteria", or "Life".

```
obj$is_root()
is_root(obj)
```

Arguments

obj	The taxonomy() or taxmap() object.

Value

A logical of length equal to the number of taxa.

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Test for which taxon IDs correspond to roots
is_root(ex_taxmap)

# Filter out roots
filter_taxa(ex_taxmap, ! is_root)
```

`is_stem` *Test if taxa are stems*

Description

Test if taxa are stems in a `taxonomy()` or `taxmap()` object. Stems are taxa from the `roots()` taxa to the first taxon with more than one subtaxon. These can usually be filtered out of the taxonomy without removing any information on how the remaining taxa are related.

```
obj$is_stem()
is_stem(obj)
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object.

Value

A `logical` of length equal to the number of taxa.

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_branch()`, `is_internode()`, `is_leaf()`, `is_root()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_subtaxa()`, `n_supertaxa_1()`, `n_supertaxa()`, `taxon_ids()`, `taxon_indexes()`, `taxon_names()`, `taxon_ranks()`

Examples

```
# Test which taxon IDs correspond to stems
is_stem(ex_taxmap)

# Filter out stems
filter_taxa(ex_taxmap, ! is_stem)
```

leaves *Get leaf taxa*

Description

Return the leaf taxa for a taxonomy() or taxmap() object. Leaf taxa are taxa with no subtaxa.

```
obj$leaves(subset = NULL, recursive = TRUE, simplify = FALSE, value = "taxon_indexes")
leaves(obj, subset = NULL, recursive = TRUE, simplify = FALSE, value = "taxon_indexes")
```

Arguments

obj	The taxonomy() or taxmap() object containing taxon information to be queried.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find leaves for. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own.
recursive	(logical or numeric) If FALSE, only return the leaves if they occur one rank below the target taxa. If TRUE, return all of the leaves for each taxon. Positive numbers indicate the number of recursions (i.e. number of ranks below the target taxon to return). 1 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
simplify	(logical) If TRUE, then combine all the results into a single vector of unique values.
value	What data to return. This is usually the name of column in a table in obj$data. Any result of all_names(obj) can be used, but it usually only makes sense to data that corresponds to taxa 1:1, such as taxon_ranks(). By default, taxon indexes are returned.

Value

character

See Also

Other taxonomy indexing functions: branches(), internodes(), roots(), stems(), subtaxa(), supertaxa()

Examples

```
# Return indexes of leaf taxa
leaves(ex_taxmap)

# Return indexes for a subset of taxa
leaves(ex_taxmap, subset = 2:17)
leaves(ex_taxmap, subset = taxon_names == "Plantae")

# Return something besides taxon indexes
```

```
leaves(ex_taxmap, value = "taxon_names")
leaves(ex_taxmap, subset = taxon_ranks == "genus", value = "taxon_names")

# Return a vector of all unique values
leaves(ex_taxmap, value = "taxon_names", simplify = TRUE)

# Only return leaves for their direct supertaxa
leaves(ex_taxmap, value = "taxon_names", recursive = FALSE)
```

`leaves_apply` *Apply function to leaves of each taxon*

Description

Apply a function to the leaves of each taxon. This is similar to using `leaves()` with `lapply()` or `sapply()`.

```
obj$leaves_apply(func, subset = NULL, recursive = TRUE,
  simplify = FALSE, value = "taxon_indexes", ...)
leaves_apply(obj, func, subset = NULL, recursive = TRUE,
  simplify = FALSE, value = "taxon_indexes", ...)
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object containing taxon information to be queried.
`func`	(`function`) The function to apply.
`subset`	Taxon IDs, TRUE/FALSE vector, or taxon indexes to use. Default: All taxa in `obj` will be used. Any variable name that appears in `all_names()` can be used as if it was a vector on its own.
`recursive`	(`logical` or `numeric`) If `FALSE`, only return the leaves if they occur one rank below the target taxa. If `TRUE`, return all of the leaves for each taxon. Positive numbers indicate the number of recursions (i.e. number of ranks below the target taxon to return). `1` is equivalent to `FALSE`. Negative numbers are equivalent to `TRUE`.
`simplify`	(`logical`) If `TRUE`, then combine all the results into a single vector of unique values.
`value`	What data to give to the function. Any result of `all_names(obj)` can be used, but it usually only makes sense to use data that has an associated taxon id.
`...`	Extra arguments are passed to the function `func`.

Examples

```
# Count number of leaves under each taxon or its subtaxa
leaves_apply(ex_taxmap, length)

# Count number of leaves under each taxon
leaves_apply(ex_taxmap, length, recursive = FALSE)

# Converting output of leaves to upper case
leaves_apply(ex_taxmap, value = "taxon_names", toupper)

# Passing arguments to the function
leaves_apply(ex_taxmap, value = "taxon_names", paste0, collapse = ", ")
```

---

lookup_tax_data *Convert one or more data sets to taxmap*

Description

Looks up taxonomic data from NCBI sequence IDs, taxon IDs, or taxon names that are present in a table, list, or vector. Also can incorporate additional associated datasets.

Usage

```
lookup_tax_data(
  tax_data,
  type,
  column = 1,
  datasets = list(),
  mappings = c(),
  database = "ncbi",
  include_tax_data = TRUE,
  use_database_ids = TRUE,
  ask = TRUE
)
```

Arguments

tax_data	A table, list, or vector that contain sequence IDs, taxon IDs, or taxon names. • tables: The `column` option must be used to specify which column contains the sequence IDs, taxon IDs, or taxon names. • lists: There must be only one item per list entry unless the `column` option is used to specify what item to use in each list entry. • vectors: simply a vector of sequence IDs, taxon IDs, or taxon names.
type	What type of information can be used to look up the classifications. Takes one of the following values:

- `"seq_id"`: A database sequence ID with an associated classification (e.g. NCBI accession numbers).
- `"taxon_id"`: A reference database taxon ID (e.g. a NCBI taxon ID)
- `"taxon_name"`: A single taxon name (e.g. "Homo sapiens" or "Primates")
- `"fuzzy_name"`: A single taxon name, but check for misspellings first. Only use if you think there are misspellings. Using `"taxon_name"` is faster.

`column` (`character` or `integer`) The name or index of the column that contains information used to lookup classifications. This only applies when a table or list is supplied to `tax_data`.

`datasets` Additional lists/vectors/tables that should be included in the resulting `taxmap` object. The `mappings` option is use to specify how these data sets relate to the `tax_data` and, by inference, what taxa apply to each item.

`mappings` (named `character`) This defines how the taxonomic information in `tax_data` applies to data in `datasets`. This option should have the same number of inputs as `datasets`, with values corresponding to each dataset. The names of the character vector specify what information in `tax_data` is shared with info in each `dataset`, which is specified by the corresponding values of the character vector. If there are no shared variables, you can add `NA` as a placeholder, but you could just leave that data out since it is not benefiting from being in the taxmap object. The names/values can be one of the following:

- For tables, the names of columns can be used.
- `"{{index}}"` : This means to use the index of rows/items
- `"{{name}}"` : This means to use row/item names.
- `"{{value}}"` : This means to use the values in vectors or lists. Lists will be converted to vectors using `unlist()`.

`database` (`character`) The name of a database to use to look up classifications. Options include "ncbi", "itis", "eol", "col", "tropicos", and "nbn".

`include_tax_data`
(`TRUE`/`FALSE`) Whether or not to include `tax_data` as a dataset, like those in `datasets`.

`use_database_ids`
(`TRUE`/`FALSE`) Whether or not to use downloaded database taxon ids instead of arbitrary, automatically-generated taxon ids.

`ask` (`TRUE`/`FALSE`) Whether or not to prompt the user for input. Currently, this would only happen when looking up the taxonomy of a taxon name with multiple matches. If `FALSE`, taxa with multiple hits are treated as if they do not exist in the database. This might change in the future if we can find an elegant way of handling this.

Failed Downloads

If you have invalid inputs or a download fails for another reason, then there will be a "unknown" taxon ID as a placeholder and failed inputs will be assigned to this ID. You can remove these using `filter_taxa()` like so: `filter_taxa(result,taxon_ids != "unknown")`. Add `drop_obs = FALSE` if you want the input data, but want to remove the taxon.

See Also

Other parsers: extract_tax_data(), parse_edge_list(), parse_tax_data()

Examples

```
## Not run:

  # Look up taxon names in vector from NCBI
  lookup_tax_data(c("homo sapiens", "felis catus", "Solanaceae"),
                  type = "taxon_name")

  # Look up taxon names in list from NCBI
  lookup_tax_data(list("homo sapiens", "felis catus", "Solanaceae"),
                  type = "taxon_name")

  # Look up taxon names in table from NCBI
  my_table <- data.frame(name = c("homo sapiens", "felis catus"),
                         decency = c("meh", "good"))
  lookup_tax_data(my_table, type = "taxon_name", column = "name")

  # Look up taxon names from NCBI with fuzzy matching
  lookup_tax_data(c("homo sapienss", "feles catus", "Solanacese"),
                  type = "fuzzy_name")

  # Look up taxon names from a different database
  lookup_tax_data(c("homo sapiens", "felis catus", "Solanaceae"),
                  type = "taxon_name", database = "ITIS")

  # Prevent asking questions for ambiguous taxon names
  lookup_tax_data(c("homo sapiens", "felis catus", "Solanaceae"),
                  type = "taxon_name", database = "ITIS", ask = FALSE)

  # Look up taxon IDs from NCBI
  lookup_tax_data(c("9689", "9694", "9643"), type = "taxon_id")

  # Look up sequence IDs from NCBI
  lookup_tax_data(c("AB548412", "FJ358423", "DQ334818"),
                  type = "seq_id")

  # Make up new taxon IDs instead of using the downloaded ones
  lookup_tax_data(c("AB548412", "FJ358423", "DQ334818"),
                  type = "seq_id", use_database_ids = FALSE)


  # --- Parsing multiple datasets at once (advanced) ---
  # The rest is one example for how to classify multiple datasets at once.

  # Make example data with taxonomic classifications
  species_data <- data.frame(tax = c("Mammalia;Carnivora;Felidae",
                                     "Mammalia;Carnivora;Felidae",
                                     "Mammalia;Carnivora;Ursidae"),
                             species = c("Panthera leo",
```

```
                                        "Panthera tigris",
                                        "Ursus americanus"),
                              species_id = c("A", "B", "C"))

  # Make example data associated with the taxonomic data
  # Note how this does not contain classifications, but
  # does have a varaible in common with "species_data" ("id" = "species_id")
  abundance <- data.frame(id = c("A", "B", "C", "A", "B", "C"),
                          sample_id = c(1, 1, 1, 2, 2, 2),
                          counts = c(23, 4, 3, 34, 5, 13))

  # Make another related data set named by species id
  common_names <- c(A = "Lion", B = "Tiger", C = "Bear", "Oh my!")

  # Make another related data set with no names
  foods <- list(c("ungulates", "boar"),
                c("ungulates", "boar"),
                c("salmon", "fruit", "nuts"))

  # Make a taxmap object with these three datasets
  x = lookup_tax_data(species_data,
                      type = "taxon_name",
                      datasets = list(counts = abundance,
                                      my_names = common_names,
                                      foods = foods),
                      mappings = c("species_id" = "id",
                                   "species_id" = "{{name}}",
                                   "{{index}}" = "{{index}}"),
                      column = "species")

  # Note how all the datasets have taxon ids now
  x$data

  # This allows for complex mappings between variables that other functions use
  map_data(x, my_names, foods)
  map_data(x, counts, my_names)

## End(Not run)
```

---

map_data *Create a mapping between two variables*

Description

Creates a named vector that maps the values of two variables associated with taxa in a `taxonomy()` or `taxmap()` object. Both values must be named by taxon ids.

```
obj$map_data(from, to, warn = TRUE)
map_data(obj, from, to, warn = TRUE)
```

Arguments

obj	The taxonomy() or taxmap() object.
from	The value used to name the output. There will be one output value for each value in from. Any variable that appears in all_names() can be used as if it was a variable on its own.
to	The value returned in the output. Any variable that appears in all_names() can be used as if it was a variable on its own.
warn	If TRUE, issue a warning if there are multiple unique values of to for each value of from.

Value

A vector of to values named by values in from.

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Mapping between two variables in `all_names(ex_taxmap)`
map_data(ex_taxmap, from = taxon_names, to = n_legs > 0)

# Mapping with external variables
x = c("d" = "looks like a cat", "h" = "big scary cats",
      "i" = "smaller cats", "m" = "might eat you", "n" = "Meow! (Feed me!)")
map_data(ex_taxmap, from = taxon_names, to = x)
```

---

map_data_ *Create a mapping without NSE*

Description

Creates a named vector that maps the values of two variables associated with taxa in a taxonomy() or taxmap() object without using Non-Standard Evaluation (NSE). Both values must be named by taxon ids. This is the same as map_data() without NSE and can be useful in some odd cases where NSE fails to work as expected.

```
obj$map_data(from, to)
map_data(obj, from, to)
```

Arguments

obj	The taxonomy() or taxmap() object.
from	The value used to name the output. There will be one output value for each value in from.
to	The value returned in the output.

Value

A vector of to values named by values in from.

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
x = c("d" = "looks like a cat", "h" = "big scary cats",
      "i" = "smaller cats", "m" = "might eat you", "n" = "Meow! (Feed me!)")
map_data_(ex_taxmap, from = ex_taxmap$taxon_names(), to = x)
```

---

mutate_obs *Add columns to taxmap() objects*

Description

Add columns to tables in obj$data in taxmap() objects. See dplyr::mutate() for the inspiration for this function and more information. Calling the function using the obj$mutate_obs(...) style edits "obj" in place, unlike most R functions. However, calling the function using the mutate_obs(obj,...) imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$mutate_obs(data, ...)
mutate_obs(obj, data, ...)
```

Arguments

obj	An object of type taxmap()
data	Dataset name, index, or a logical vector that indicates which dataset in obj$data to add columns to.
...	One or more named columns to add. Newly created columns can be referenced in the same function call. Any variable name that appears in all_names() can be used as if it was a vector on its own.
target	DEPRECIATED. use "data" instead.

Value

An object of type taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), arrange_taxa(), filter_obs(), filter_taxa(), sample_frac_obs(), sample_frac_taxa(), sample_n_obs(), sample_n_taxa(), select_obs(), transmute_obs()

Examples

```
# Add column to existing tables
mutate_obs(ex_taxmap, "info",
           new_col = "Im new",
           newer_col = paste0(new_col, "er!"))

# Create columns in a new table
mutate_obs(ex_taxmap, "new_table",
           nums = 1:10,
           squared = nums ^ 2)

# Add a new vector
mutate_obs(ex_taxmap, "new_vector", 1:10)

# Add a new list
mutate_obs(ex_taxmap, "new_list", list(1, 2))
```

---

n_leaves *Get number of leaves*

Description

Get number of leaves for each taxon in an object of type taxonomy() or taxmap()

```
obj$n_leaves()
n_leaves(obj)
```

Arguments

obj	(taxonomy() or taxmap())

Value

numeric

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Get number of leaves for each taxon
n_leaves(ex_taxmap)

# Filter taxa based on number of leaves
filter_taxa(ex_taxmap, n_leaves > 0)
```

---

n_leaves_1 *Get number of leaves*

Description

Get number of leaves for each taxon in an object of type taxonomy() or taxmap(), not including leaves of subtaxa etc.

```
obj$n_leaves_1()
n_leaves_1(obj)
```

Arguments

obj	(taxonomy() or taxmap())

Value

numeric

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Get number of leaves for each taxon
n_leaves_1(ex_taxmap)

# Filter taxa based on number of leaves
filter_taxa(ex_taxmap, n_leaves_1 > 0)
```

n_obs *Count observations in* taxmap()

Description

Count observations for each taxon in a data set in a taxmap() object. This includes observations for the specific taxon and the observations of its subtaxa. "Observations" in this sense are the items (for list/vectors) or rows (for tables) in a dataset. By default, observations in the first data set in the taxmap() object is used. For example, if the data set is a table, then a value of 3 for a taxon means that their are 3 rows in that table assigned to that taxon or one of its subtaxa.

```
obj$n_obs(data)
n_obs(obj, data)
```

Arguments

obj	(taxmap())
data	Dataset name, index, or a logical vector that indicates which dataset in obj$data to add columns to.
target	DEPRECIATED. use "data" instead.

Value

numeric

See Also

Other taxmap data functions: n_obs_1()

Examples

```
# Get number of observations for each taxon in first dataset
n_obs(ex_taxmap)

# Get number of observations in a specified data set
n_obs(ex_taxmap, "info")
n_obs(ex_taxmap, "abund")

# Filter taxa using number of observations in the first table
filter_taxa(ex_taxmap, n_obs > 1)
```

n_obs_1 *Count observation assigned in* taxmap()

Description

Count observations for each taxon in a data set in a taxmap() object. This includes observations for the specific taxon but NOT the observations of its subtaxa. "Observations" in this sense are the items (for list/vectors) or rows (for tables) in a dataset. By default, observations in the first data set in the taxmap() object is used. For example, if the data set is a table, then a value of 3 for a taxon means that their are 3 rows in that table assigned to that taxon.

```
obj$n_obs_1(data)
n_obs_1(obj, data)
```

Arguments

obj	(taxmap())
data	Dataset name, index, or a logical vector that indicates which dataset in obj$data to add columns to.
target	DEPRECIATED. use "data" instead.

Value

numeric

See Also

Other taxmap data functions: n_obs()

Examples

```
# Get number of observations for each taxon in first dataset
n_obs_1(ex_taxmap)

# Get number of observations in a specified data set
n_obs_1(ex_taxmap, "info")
n_obs_1(ex_taxmap, "abund")

# Filter taxa using number of observations in the first table
filter_taxa(ex_taxmap, n_obs_1 > 0)
```

n_subtaxa *Get number of subtaxa*

Description

Get number of subtaxa for each taxon in an object of type `taxonomy()` or `taxmap()`

```
obj$n_subtaxa()
n_subtaxa(obj)
```

Arguments

`obj`	(`taxonomy()` or `taxmap()`)

Value

`numeric`

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_branch()`, `is_internode()`, `is_leaf()`, `is_root()`, `is_stem()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_supertaxa_1()`, `n_supertaxa()`, `taxon_ids()`, `taxon_indexes()`, `taxon_names()`, `taxon_ranks()`

Examples

```
# Count number of subtaxa within each taxon
n_subtaxa(ex_taxmap)

# Filter taxa based on number of subtaxa
#  (this command removed all leaves or "tips" of the tree)
filter_taxa(ex_taxmap, n_subtaxa > 0)
```

n_subtaxa_1 *Get number of subtaxa*

Description

Get number of subtaxa for each taxon in an object of type `taxonomy()` or `taxmap()`, not including subtaxa of subtaxa etc. This does not include subtaxa assigned to subtaxa.

```
obj$n_subtaxa_1()
n_subtaxa_1(obj)
```

Arguments

obj (taxonomy() or taxmap())

Value

numeric

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Count number of immediate subtaxa in each taxon
n_subtaxa_1(ex_taxmap)

# Filter taxa based on number of subtaxa
#  (this command removed all leaves or "tips" of the tree)
filter_taxa(ex_taxmap, n_subtaxa_1 > 0)
```

---

n_supertaxa *Get number of supertaxa*

Description

Get number of supertaxa for each taxon in an object of type taxonomy() or taxmap().

```
obj$n_supertaxa()
n_supertaxa(obj)
```

Arguments

obj (taxonomy() or taxmap())

Value

numeric

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), taxon_ids(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Count number of supertaxa that contain each taxon
n_supertaxa(ex_taxmap)

# Filter taxa based on the number of supertaxa
#  (this command removes all root taxa)
filter_taxa(ex_taxmap, n_supertaxa > 0)
```

`n_supertaxa_1` *Get number of supertaxa*

Description

Get number of immediate supertaxa (i.e. not supertaxa of supertaxa, etc) for each taxon in an object of type `taxonomy()` or `taxmap()`. This should always be either 1 or 0.

```
obj$n_supertaxa_1()
n_supertaxa_1(obj)
```

Arguments

`obj`	(`taxonomy()` or `taxmap()`)

Value

`numeric`

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_branch()`, `is_internode()`, `is_leaf()`, `is_root()`, `is_stem()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_subtaxa()`, `n_supertaxa()`, `taxon_ids()`, `taxon_indexes()`, `taxon_names()`, `taxon_ranks()`

Examples

```
# Test for the presence of supertaxa containing each taxon
n_supertaxa_1(ex_taxmap)

# Filter taxa based on the presence of supertaxa
#  (this command removes all root taxa)
filter_taxa(ex_taxmap, n_supertaxa_1 > 0)
```

obs *Get data indexes associated with taxa*

Description

Given a taxmap() object, return data associated with each taxon in a given table included in that taxmap() object.

```
obj$obs(data, value = NULL, subset = NULL,
  recursive = TRUE, simplify = FALSE)
obs(obj, data, value = NULL, subset = NULL,
  recursive = TRUE, simplify = FALSE)
```

Arguments

obj	(taxmap()) The taxmap() object containing taxon information to be queried.
data	Either the name of something in obj$data that has taxon information or a an external object with taxon information. For tables, there must be a column named "taxon_id" and lists/vectors must be named by taxon ID.
value	What data to return. This is usually the name of column in a table in obj$data. Any result of all_names(obj) can be used. If the value used has names, it is assumed that the names are taxon ids and the taxon ids are used to look up the correct values.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find observations for. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own.
recursive	(logical or numeric) If FALSE, only return the observation assigned to the specified input taxa, not subtaxa. If TRUE, return all the observations of every subtaxa, etc. Positive numbers indicate the number of ranks below the each taxon to get observations for 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
simplify	(logical) If TRUE, then combine all the results into a single vector of unique observation indexes.

Value

If simplify = FALSE, then a list of vectors of observation indexes are returned corresponding to the data argument. If simplify = TRUE, then the observation indexes for all data taxa are returned in a single vector.

Examples

```
# Get indexes of rows corresponding to each taxon
obs(ex_taxmap, "info")

# Get only a subset of taxon indexes
```

```
obs(ex_taxmap, "info", subset = 1:2)

# Get only a subset of taxon IDs
obs(ex_taxmap, "info", subset = c("b", "c"))

# Get only a subset of taxa using logical tests
obs(ex_taxmap, "info", subset = taxon_ranks == "genus")

# Only return indexes of rows assinged to each taxon explicitly
obs(ex_taxmap, "info", recursive = FALSE)

# Lump all row indexes in a single vector
obs(ex_taxmap, "info", simplify = TRUE)

# Return values from a dataset instead of indexes
obs(ex_taxmap, "info", value = "name")
```

---

obs_apply *Apply function to observations per taxon*

Description

Apply a function to data for the observations for each taxon. This is similar to using `obs()` with `lapply()` or `sapply()`.

```
obj$obs_apply(data, func, simplify = FALSE, value = NULL,
  subset = NULL, recursive = TRUE, ...)
obs_apply(obj, data, func, simplify = FALSE, value = NULL,
  subset = NULL, recursive = TRUE, ...)
```

Arguments

`obj`	The `taxmap()` object containing taxon information to be queried.
`data`	Either the name of something in `obj$data` that has taxon information or a an external object with taxon information. For tables, there must be a column named "taxon_id" and lists/vectors must be named by taxon ID.
`func`	(`function`) The function to apply.
`simplify`	(`logical`) If `TRUE`, convert lists to vectors.
`value`	What data to give to the function. This is usually the name of column in a table in `obj$data`. Any result of `all_names(obj)` can be used, but it usually only makes sense to use columns in the dataset specified by the `data` option. By default, the indexes of observation in `data` are returned.
`subset`	Taxon IDs, TRUE/FALSE vector, or taxon indexes to use. Default: All taxa in `obj` will be used. Any variable name that appears in `all_names()` can be used as if it was a vector on its own.

recursive	(logical or numeric) If FALSE, only return the observation assigned to the specified input taxa, not subtaxa. If TRUE, return all the observations of every subtaxa, etc. Positive numbers indicate the number of ranks below the each taxon to get observations for 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
...	Extra arguments are passed to the function.

Examples

```
# Find the average number of legs in each taxon
obs_apply(ex_taxmap, "info", mean, value = "n_legs", simplify = TRUE)

# One way to implement `n_obs` and find the number of observations per taxon
obs_apply(ex_taxmap, "info", length, simplify = TRUE)
```

---

parse_dataset *Parse options specifying datasets*

Description

Parse options specifying datasets in taxmap objects

Usage

```
parse_dataset(obj, data, must_be_valid = TRUE, needed = TRUE, rm_na = TRUE)
```

Arguments

obj	The taxmap object.
data	The name/index of datasets in a taxmap object to use. Can also be a logical vector of length equal to the number of datasets.
must_be_valid	If TRUE, all datasets specified must be valid or an error occurs.
needed	If TRUE, at least one dataset must be specified or an error occurs.
rm_na	If TRUE, then invalid datasets do result in NAs in the output.

Value

The indexes for the datasets selected

parse_edge_list *Convert a table with an edge list to taxmap*

Description

Converts a table containing an edge list into a `taxa::taxmap()` object. An "edge list" is two columns in a table, where each row defines a taxon-supertaxon relationship. The contents of the edge list will be used as taxon IDs. The whole table will be included as a data set in the output object.

Usage

```
parse_edge_list(input, taxon_id, supertaxon_id, taxon_name, taxon_rank = NULL)
```

Arguments

`input`	A table containing an edge list encoded by two columns.
`taxon_id`	The name/index of the column containing the taxon IDs.
`supertaxon_id`	The name/index of the column containing the taxon IDs for the supertaxon of the IDs in `taxon_col`.
`taxon_name`	xxx
`taxon_rank`	xxx

See Also

Other parsers: `extract_tax_data()`, `lookup_tax_data()`, `parse_tax_data()`

parse_tax_data *Convert one or more data sets to taxmap*

Description

Reads taxonomic information and associated data in tables, lists, and vectors and stores it in a `taxa::taxmap()` object. Taxonomic classifications must be present.

Usage

```
parse_tax_data(
  tax_data,
  datasets = list(),
  class_cols = 1,
  class_sep = ";",
  sep_is_regex = FALSE,
  class_key = "taxon_name",
```

```
  class_regex = "(.*)",
  class_reversed = FALSE,
  include_match = TRUE,
  mappings = c(),
  include_tax_data = TRUE,
  named_by_rank = FALSE
)
```

Arguments

tax_data	A table, list, or vector that contains the names of taxa that represent taxonomic classifications. Accepted representations of classifications include: * A list/vector or table with column(s) of taxon names: Something like "Animalia;Chordata;Mammalia;Pri What separator(s) is used (";" in this example) can be changed with the class_sep option. For tables, the classification can be spread over multiple columns and the separator(s) will be applied to each column, although each column could just be single taxon names with no separator. Use the class_cols option to specify which columns have taxon names. * A list in which each entry is a classifications. For example, list(c("Animalia","Chordata","Mammalia","Primates","Hominidae","Homo * A list of data.frames where each represents a classification with one taxon per row. The column that contains taxon names is specified using the class_cols option. In this instance, it only makes sense to specify a single column.
datasets	Additional lists/vectors/tables that should be included in the resulting taxmap object. The mappings option is use to specify how these data sets relate to the tax_data and, by inference, what taxa apply to each item.
class_cols	(character or integer) The names or indexes of columns that contain classifications if the first input is a table. If multiple columns are specified, they will be combined in the order given. Negative column indexes mean "every column besides these columns".
class_sep	(character) One or more separators that delineate taxon names in a classification. For example, if one column had "Homo sapiens" and another had "Animalia;Chordata;Mammalia;Primates;Hominidae", then class_sep = c(" ",";"). All separators are applied to each column so order does not matter.
sep_is_regex	(TRUE/FALSE) Whether or not class_sep should be used as a regular expression.
class_key	(character of length 1) The identity of the capturing groups defined using class_regex. The length of class_key must be equal to the number of capturing groups specified in class_regex. Any names added to the terms will be used as column names in the output. At least one "taxon_name" must be specified. Only "info" can be used multiple times. Each term must be one of those described below: * taxon_name: The name of a taxon. Not necessarily unique, but are interpretable by a particular database. Requires an internet connection. * taxon_rank: The rank of the taxon. This will be used to add rank info into the output object that can be accessed by out$taxon_ranks(). * info: Arbitrary taxon info you want included in the output. Can be used more than once.
class_regex	(character of length 1) A regular expression with capturing groups indicating the locations of data for each taxon in the class term in the key argument. The identity of the information must be specified using the class_key argument.

The class_sep option can be used to split the classification into data for each taxon before matching. If class_sep is NULL, each match of class_regex defines a taxon in the classification.

class_reversed If TRUE, then classifications go from specific to general. For example: Abditomys latidens : Muridae : Rodentia : Mammalia : Chordata.

include_match (logical of length 1) If TRUE, include the part of the input matched by class_regex in the output object.

mappings (named character) This defines how the taxonomic information in tax_data applies to data set in datasets. This option should have the same number of inputs as datasets, with values corresponding to each data set. The names of the character vector specify what information in tax_data is shared with info in each dataset, which is specified by the corresponding values of the character vector. If there are no shared variables, you can add NA as a placeholder, but you could just leave that data out since it is not benefiting from being in the taxmap object. The names/values can be one of the following: * For tables, the names of columns can be used. * "{{index}}" : This means to use the index of rows/items * "{{name}}" : This means to use row/item names. * "{{value}}" : This means to use the values in vectors or lists. Lists will be converted to vectors using unlist().

include_tax_data (TRUE/FALSE) Whether or not to include tax_data as a dataset, like those in datasets.

named_by_rank (TRUE/FALSE) If TRUE and the input is a table with columns named by ranks or a list of vectors with each vector named by ranks, include that rank info in the output object, so it can be accessed by out$taxon_ranks(). If TRUE, taxa with different ranks, but the same name and location in the taxonomy, will be considered different taxa. Cannot be used with the sep, class_regex, or class_key options.

See Also

Other parsers: extract_tax_data(), lookup_tax_data(), parse_edge_list()

Examples

```
# Read a vector of classifications
my_taxa <- c("Mammalia;Carnivora;Felidae",
             "Mammalia;Carnivora;Felidae",
             "Mammalia;Carnivora;Ursidae")
parse_tax_data(my_taxa, class_sep = ";")

# Read a list of classifications
my_taxa <- list("Mammalia;Carnivora;Felidae",
               "Mammalia;Carnivora;Felidae",
               "Mammalia;Carnivora;Ursidae")
parse_tax_data(my_taxa, class_sep = ";")

# Read classifications in a table in a single column
species_data <- data.frame(tax = c("Mammalia;Carnivora;Felidae",
```

```
                                  "Mammalia;Carnivora;Felidae",
                                  "Mammalia;Carnivora;Ursidae"),
                           species_id = c("A", "B", "C"))
parse_tax_data(species_data, class_sep = ";", class_cols = "tax")

# Read classifications in a table in multiple columns
species_data <- data.frame(lineage = c("Mammalia;Carnivora;Felidae",
                                       "Mammalia;Carnivora;Felidae",
                                       "Mammalia;Carnivora;Ursidae"),
                           species = c("Panthera leo",
                                       "Panthera tigris",
                                       "Ursus americanus"),
                           species_id = c("A", "B", "C"))
parse_tax_data(species_data, class_sep = c(" ", ";"),
               class_cols = c("lineage", "species"))

# Read classification tables with one column per rank
species_data <- data.frame(class = c("Mammalia", "Mammalia", "Mammalia"),
                           order = c("Carnivora", "Carnivora", "Carnivora"),
                           family = c("Felidae", "Felidae", "Ursidae"),
                           genus = c("Panthera", "Panthera", "Ursus"),
                           species = c("leo", "tigris", "americanus"),
                           species_id = c("A", "B", "C"))
 parse_tax_data(species_data, class_cols = 1:5)
 parse_tax_data(species_data, class_cols = 1:5,
                named_by_rank = TRUE) # makes `taxon_ranks()` work

# Classifications with extra information
my_taxa <- c("Mammalia_class_1;Carnivora_order_2;Felidae_genus_3",
             "Mammalia_class_1;Carnivora_order_2;Felidae_genus_3",
             "Mammalia_class_1;Carnivora_order_2;Ursidae_genus_3")
parse_tax_data(my_taxa, class_sep = ";",
               class_regex = "(.+)_(.+)_([0-9]+)",
               class_key = c(my_name = "taxon_name",
                             a_rank = "taxon_rank",
                             some_num = "info"))


 # --- Parsing multiple datasets at once (advanced) ---
 # The rest is one example for how to classify multiple datasets at once.

 # Make example data with taxonomic classifications
 species_data <- data.frame(tax = c("Mammalia;Carnivora;Felidae",
                                    "Mammalia;Carnivora;Felidae",
                                    "Mammalia;Carnivora;Ursidae"),
                            species = c("Panthera leo",
                                        "Panthera tigris",
                                        "Ursus americanus"),
                            species_id = c("A", "B", "C"))

 # Make example data associated with the taxonomic data
 # Note how this does not contain classifications, but
 # does have a varaible in common with "species_data" ("id" = "species_id")
```

```
abundance <- data.frame(id = c("A", "B", "C", "A", "B", "C"),
                        sample_id = c(1, 1, 1, 2, 2, 2),
                        counts = c(23, 4, 3, 34, 5, 13))

# Make another related data set named by species id
common_names <- c(A = "Lion", B = "Tiger", C = "Bear", "Oh my!")

# Make another related data set with no names
foods <- list(c("ungulates", "boar"),
              c("ungulates", "boar"),
              c("salmon", "fruit", "nuts"))

# Make a taxmap object with these three datasets
x = parse_tax_data(species_data,
                   datasets = list(counts = abundance,
                                   my_names = common_names,
                                   foods = foods),
                   mappings = c("species_id" = "id",
                                "species_id" = "{{name}}",
                                "{{index}}" = "{{index}}"),
                   class_cols = c("tax", "species"),
                   class_sep = c(" ", ";"))

# Note how all the datasets have taxon ids now
x$data

# This allows for complex mappings between variables that other functions use
map_data(x, my_names, foods)
map_data(x, counts, my_names)
```

`pick` *Pick taxa*

Description

Pick out specific taxa, while others are dropped

Usage

```
pick(.data, ...)
```

Arguments

`.data`	Input, object of class `Hierarchy`, or `hierarchies`
`...`	quoted rank names (e.g., family) via `ranks()`, taxon names (e.g., Poa annua) via `nms()`, or taxonomic IDs (e.g., 93036) via `ids()`. You can't pass in arbitrary strings or numbers.

Details

supports `Hierarchy` and `hierarchies` objects

Value

an object of the same class as passed in

See Also

See filtering-helpers, including for more explanation of how this function works.

Examples

```
# ranks
ex_hierarchy1
ex_hierarchy1 %>% pick(ranks("family"))
ex_hierarchy1 %>% pick(ranks("family", "genus"))
# taxon names
ex_hierarchy1 %>% pick(nms('Poa'))
ex_hierarchy1 %>% pick(nms("Poaceae", "Poa"))
# taxon ids
ex_hierarchy1 %>% pick(ids(4479))
ex_hierarchy1 %>% pick(ids(4479, 4544))
# mixed: ids and names
ex_hierarchy1 %>% pick(ranks("family"), ids(4544))

## single taxonomic group
ex_hierarchy1 %>% pick(ranks("family"))
pick(ex_hierarchy1, ranks("family"))
### more than 1 - remake res object above first
ex_hierarchy1 %>% pick(ranks("family", "genus"))


# hierarchies
# single taxonomic group
invisible(lapply(ex_hierarchies, print))
ex_hierarchies %>% pick(ranks("family")) %>% lapply(., print) %>% invisible

## more than one taxonomic group
invisible(lapply(ex_hierarchies, print))
ex_hierarchies %>% pick(ranks("family", "genus")) %>% lapply(., print) %>%
  invisible
```

---

pop *Pop taxa out*

Description

Pop out taxa, that is, drop them

Usage

```
pop(.data, ...)
```

Arguments

`.data`	Input, object of class `Hierarchy`, or `hierarchies`
`...`	quoted rank names (e.g., family) via `ranks()`, taxon names (e.g., Poa annua) via `nms()`, or taxonomic IDs (e.g., 93036) via `ids()`. You can't pass in arbitrary strings or numbers.

Details

supports `Hierarchy` and `hierarchies` objects

Value

an object of the same class as passed in

See Also

See filtering-helpers, including for more explanation of how this function works.

Examples

```
# With Hierarchy class object
ex_hierarchy1
## ranks
pop(ex_hierarchy1, ranks("family"))
ex_hierarchy1 %>% pop(ranks("family"))
ex_hierarchy1 %>% pop(ranks("family", "genus"))
## taxon names
ex_hierarchy1 %>% pop(nms("Poa"))
ex_hierarchy1 %>% pop(nms("Poaceae", "Poa"))
## taxon ids
ex_hierarchy1 %>% pop(ids(4479))
ex_hierarchy1 %>% pop(ids(4479, 4544))
## mixed: ids and names
ex_hierarchy1 %>% pop(ranks("family"), ids(4544))

# With hierarchies class object
# single taxonomic group
invisible(lapply(ex_hierarchies, print))
ex_hierarchies %>% pop(ranks("family")) %>% lapply(., print) %>% invisible
## more than one taxonomic group
invisible(lapply(ex_hierarchies, print))
ex_hierarchies %>% pop(ranks("family", "genus")) %>% lapply(., print) %>%
  invisible
```

print_tree *Print a text tree*

Description

Print a text-based tree of a `taxonomy()` or `taxmap()` object.

Arguments

`obj`	A `taxonomy` or `taxmap` object
`value`	What data to return. Default is taxon names. Any result of `all_names()` can be used, but it usually only makes sense to use data with one value per taxon, like taxon names.

Examples

```
print_tree(ex_taxmap)
```

ranks_ref *Lookup-table for IDs of taxonomic ranks*

Description

Composed of two columns:

- rankid - the ordered identifier value. lower values mean higher rank
- ranks - all the rank names that belong to the same level, with different variants that mean essentially the same thing

remove_redundant_names *Remove redundant parts of taxon names*

Description

Remove the names of parent taxa in the beginning of their children's names in a `taxonomy` or `taxmap` object. This is useful for removing genus names in species binomials.

```
obj$remove_redundant_names()
remove_redundant_names(obj)
```

Arguments

obj	A taxonomy or taxmap object

Value

A taxonomy or taxmap object

Examples

```
# Remove genus named from species taxa
species_data <- c("Carnivora;Felidae;Panthera;Panthera leo",
                  "Carnivora;Felidae;Panthera;Panthera tigris",
                  "Carnivora;Ursidae;Ursus;Ursus americanus")
obj <-  parse_tax_data(species_data, class_sep = ";")
remove_redundant_names(obj)
```

---

replace_taxon_ids *Replace taxon ids*

Description

Replace taxon ids in a taxmap() or taxonomy() object.

```
obj$replace_taxon_ids(new_ids)
replace_taxon_ids(obj, new_ids)
```

Arguments

obj	The taxonomy() or taxmap() object.
new_ids	A vector of new ids, one per taxon. They must be unique and in the same order as the corresponding ids in obj$taxon_ids().

Value

A taxonomy() or taxmap() object with new taxon ids

Examples

```
# Replace taxon IDs with numbers
replace_taxon_ids(ex_taxmap, seq_len(length(ex_taxmap$taxa)))

# Make taxon IDs capital letters
replace_taxon_ids(ex_taxmap, toupper(taxon_ids(ex_taxmap)))
```

roots *Get root taxa*

Description

Return the root taxa for a taxonomy() or taxmap() object. Can also be used to get the roots of a subset of taxa.

```
obj$roots(subset = NULL, value = "taxon_indexes")
roots(obj, subset = NULL, value = "taxon_indexes")
```

Arguments

obj	The taxonomy() or taxmap() object containing taxon information to be queried.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find roots for. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own.
value	What data to return. This is usually the name of column in a table in obj$data. Any result of all_names(obj) can be used, but it usually only makes sense to data that corresponds to taxa 1:1, such as taxon_ranks(). By default, taxon indexes are returned.

Value

character

See Also

Other taxonomy indexing functions: branches(), internodes(), leaves(), stems(), subtaxa(), supertaxa()

Examples

```
# Return indexes of root taxa
roots(ex_taxmap)

# Return indexes for a subset of taxa
roots(ex_taxmap, subset = 2:17)

# Return something besides taxon indexes
roots(ex_taxmap, value = "taxon_names")
```

sample_frac_obs *Sample a proportion of observations from* taxmap()

Description

Randomly sample some proportion of observations from a taxmap() object. Weights can be specified for observations or their taxa. See dplyr::sample_frac() for the inspiration for this function. Calling the function using the obj$sample_frac_obs(...) style edits "obj" in place, unlike most R functions. However, calling the function using the sample_frac_obs(obj,...) imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$sample_frac_obs(data, size, replace = FALSE,
  taxon_weight = NULL, obs_weight = NULL,
  use_supertaxa = TRUE, collapse_func = mean, ...)
sample_frac_obs(obj, data, size, replace = FALSE,
  taxon_weight = NULL, obs_weight = NULL,
  use_supertaxa = TRUE, collapse_func = mean, ...)
```

Arguments

obj	(taxmap()) The object to sample from.
data	Dataset names, indexes, or a logical vector that indicates which datasets in obj$data to sample. If multiple datasets are sample at once, then they must be the same length.
size	(numeric of length 1) The proportion of observations to sample.
replace	(logical of length 1) If TRUE, sample with replacement.
taxon_weight	(numeric) Non-negative sampling weights of each taxon. If use_supertaxa is TRUE, the weights for each taxon in an observation's classification are supplied to collapse_func to get the observation weight. If obs_weight is also specified, the two weights are multiplied (after taxon_weight for each observation is calculated).
obs_weight	(numeric) Sampling weights of each observation. If taxon_weight is also specified, the two weights are multiplied (after taxon_weight for each observation is calculated).
use_supertaxa	(logical or numeric of length 1) Affects how the taxon_weight is used. If TRUE, the weights for each taxon in an observation's classification are multiplied to get the observation weight. If FALSE just the taxonomic level the observation is assign to it considered. Positive numbers indicate the number of ranks above the each taxon to use. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
collapse_func	(function of length 1) If taxon_weight option is used and supertaxa is TRUE, the weights for each taxon in an observation's classification are supplied to collapse_func to get the observation weight. This function should take numeric vector and return a single number.

...	Additional options are passed to filter_obs().
target	DEPRECIATED. use "data" instead.

Value

An object of type taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), arrange_taxa(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_taxa(), sample_n_obs(), sample_n_taxa(), select_obs(), transmute_obs()

Examples

```
# Sample half of the rows fram a table
sample_frac_obs(ex_taxmap, "info", 0.5)

# Sample multiple datasets at once
sample_frac_obs(ex_taxmap, c("info", "phylopic_ids", "foods"), 0.5)
```

---

sample_frac_taxa *Sample a proportion of taxa from* taxonomy() *or* taxmap()

Description

Randomly sample some proportion of taxa from a taxonomy() or taxmap() object. Weights can be specified for taxa or the observations assigned to them. See dplyr::sample_frac() for the inspiration for this function.

```
obj$sample_frac_taxa(size, taxon_weight = NULL,
  obs_weight = NULL, obs_target = NULL,
  use_subtaxa = TRUE, collapse_func = mean, ...)
sample_frac_taxa(obj, size, taxon_weight = NULL,
  obs_weight = NULL, obs_target = NULL,
  use_subtaxa = TRUE, collapse_func = mean, ...)
```

Arguments

obj	(taxonomy() or taxmap()) The object to sample from.
size	(numeric of length 1) The proportion of taxa to sample.
taxon_weight	(numeric) Non-negative sampling weights of each taxon. If obs_weight is also specified, the two weights are multiplied (after obs_weight for each taxon is calculated).

obs_weight	(numeric) This option only applies to taxmap() objects. Sampling weights of each observation. The weights for each observation assigned to a given taxon are supplied to collapse_func to get the taxon weight. If use_subtaxa is TRUE then the observations assigned to every subtaxa are also used. Any variable name that appears in all_names() can be used as if it was a vector on its own. If taxon_weight is also specified, the two weights are multiplied (after obs_weight for each observation is calculated). obs_target must be used with this option.
obs_target	(character of length 1) This option only applies to taxmap() objects. The name of the data set in obj$data that values in obs_weight corresponds to. Must be used when obs_weight is used.
use_subtaxa	(logical or numeric of length 1) Affects how the obs_weight option is used. If TRUE, the weights for each taxon in an observation's classification are multiplied to get the observation weight. If TRUE just the taxonomic level the observation is assign to it considered. Positive numbers indicate the number of ranks below the target taxa to return. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
collapse_func	(function of length 1) If taxon_weight is used and supertaxa is TRUE, the weights for each taxon in an observation's classification are supplied to collapse_func to get the observation weight. This function should take numeric vector and return a single number.
...	Additional options are passed to filter_taxa().

Value

An object of type taxonomy() or taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), arrange_taxa(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_obs(), sample_n_obs(), sample_n_taxa(), select_obs(), transmute_obs()

Examples

```
# sample half of the taxa
sample_frac_taxa(ex_taxmap, 0.5, supertaxa = TRUE)
```

---

sample_n_obs *Sample n observations from* taxmap()

Description

Randomly sample some number of observations from a `taxmap()` object. Weights can be specified for observations or the taxa they are classified by. Any variable name that appears in `all_names()` can be used as if it was a vector on its own. See `dplyr::sample_n()` for the inspiration for this function. Calling the function using the `obj$sample_n_obs(...)` style edits "obj" in place, unlike most R functions. However, calling the function using the `sample_n_obs(obj,...)` imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$sample_n_obs(data, size, replace = FALSE,
  taxon_weight = NULL, obs_weight = NULL,
  use_supertaxa = TRUE, collapse_func = mean, ...)
sample_n_obs(obj, data, size, replace = FALSE,
  taxon_weight = NULL, obs_weight = NULL,
  use_supertaxa = TRUE, collapse_func = mean, ...)
```

Arguments

`obj`	(`taxmap()`) The object to sample from.
`data`	Dataset names, indexes, or a logical vector that indicates which datasets in `obj$data` to sample. If multiple datasets are sampled at once, then they must be the same length.
`size`	(`numeric` of length 1) The number of observations to sample.
`replace`	(`logical` of length 1) If `TRUE`, sample with replacement.
`taxon_weight`	(`numeric`) Non-negative sampling weights of each taxon. If `use_supertaxa` is `TRUE`, the weights for each taxon in an observation's classification are supplied to `collapse_func` to get the observation weight. If `obs_weight` is also specified, the two weights are multiplied (after `taxon_weight` for each observation is calculated).
`obs_weight`	(`numeric`) Sampling weights of each observation. If `taxon_weight` is also specified, the two weights are multiplied (after `taxon_weight` for each observation is calculated).
`use_supertaxa`	(`logical` or `numeric` of length 1) Affects how the `taxon_weight` is used. If `TRUE`, the weights for each taxon in an observation's classification are multiplied to get the observation weight. Otherwise, just the taxonomic level the observation is assign to it considered. If `TRUE`, use all supertaxa. Positive numbers indicate the number of ranks above each taxon to use. `0` is equivalent to `FALSE`. Negative numbers are equivalent to `TRUE`.
`collapse_func`	(`function` of length 1) If `taxon_weight` option is used and `supertaxa` is `TRUE`, the weights for each taxon in an observation's classification are supplied to `collapse_func` to get the observation weight. This function should take numeric vector and return a single number.
`...`	Additional options are passed to `filter_obs()`.
`target`	DEPRECIATED. use "data" instead.

Value

An object of type taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), arrange_taxa(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_obs(), sample_frac_taxa(), sample_n_taxa(), select_obs(), transmute_obs()

Examples

```
# Sample 2 rows without replacement
sample_n_obs(ex_taxmap, "info", 2)
sample_n_obs(ex_taxmap, "foods", 2)

# Sample with replacement
sample_n_obs(ex_taxmap, "info", 10, replace = TRUE)

# Sample some rows for often then others
sample_n_obs(ex_taxmap, "info", 3, obs_weight = n_legs)

# Sample multiple datasets at once
sample_n_obs(ex_taxmap, c("info", "phylopic_ids", "foods"), 3)
```

---

sample_n_taxa *Sample n taxa from* taxonomy() *or* taxmap()

Description

Randomly sample some number of taxa from a taxonomy() or taxmap() object. Weights can be specified for taxa or the observations assigned to them. See dplyr::sample_n() for the inspiration for this function.

```
obj$sample_n_taxa(size, taxon_weight = NULL,
  obs_weight = NULL, obs_target = NULL,
  use_subtaxa = TRUE, collapse_func = mean, ...)
sample_n_taxa(obj, size, taxon_weight = NULL,
  obs_weight = NULL, obs_target = NULL,
  use_subtaxa = TRUE, collapse_func = mean, ...)
```

Arguments

obj	(taxonomy() or taxmap()) The object to sample from.
size	(numeric of length 1) The number of taxa to sample.
taxon_weight	(numeric) Non-negative sampling weights of each taxon. If obs_weight is also specified, the two weights are multiplied (after obs_weight for each taxon is calculated).

obs_weight (numeric) This option only applies to taxmap() objects. Sampling weights of each observation. The weights for each observation assigned to a given taxon are supplied to collapse_func to get the taxon weight. If use_subtaxa is TRUE then the observations assigned to every subtaxa are also used. Any variable name that appears in all_names() can be used as if it was a vector on its own. If taxon_weight is also specified, the two weights are multiplied (after obs_weight for each observation is calculated). obs_target must be used with this option.

obs_target (character of length 1) This option only applies to taxmap() objects. The name of the data set in obj$data that values in obs_weight corresponds to. Must be used when obs_weight is used.

use_subtaxa (logical or numeric of length 1) Affects how the obs_weight option is used. If TRUE, the weights for each taxon in an observation's classification are multiplied to get the observation weight. If FALSE just the taxonomic level the observation is assign to it considered. Positive numbers indicate the number of ranks below the each taxon to use. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.

collapse_func (function of length 1) If taxon_weight is used and supertaxa is TRUE, the weights for each taxon in an observation's classification are supplied to collapse_func to get the observation weight. This function should take numeric vector and return a single number.

... Additional options are passed to filter_taxa().

Value

An object of type taxonomy() or taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), arrange_taxa(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_obs(), sample_frac_taxa(), sample_n_obs(), select_obs(), transmute_obs()

Examples

```
# Randomly sample three taxa
sample_n_taxa(ex_taxmap, 3)

# Include supertaxa
sample_n_taxa(ex_taxmap, 3, supertaxa = TRUE)

# Include subtaxa
sample_n_taxa(ex_taxmap, 1, subtaxa = TRUE)

# Sample some taxa more often then others
sample_n_taxa(ex_taxmap, 3, supertaxa = TRUE,
              obs_weight = n_legs, obs_target = "info")
```

`select_obs` *Subset columns in a* `taxmap()` *object*

Description

Subsets columns in a `taxmap()` object. Takes and returns a `taxmap()` object. Any variable name that appears in `all_names()` can be used as if it was a vector on its own. See `dplyr::select()` for the inspiration for this function and more information. Calling the function using the `obj$select_obs(...)` style edits "obj" in place, unlike most R functions. However, calling the function using the `select_obs(obj,...)` imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$select_obs(data, ...)
select_obs(obj, data, ...)
```

Arguments

`obj`	An object of type `taxmap()`
`data`	Dataset names, indexes, or a logical vector that indicates which tables in `obj$data` to subset columns in. Multiple tables can be subset at once.
`...`	One or more column names to return in the new object. Each can be one of two things: **expression with unquoted column name** The name of a column in the dataset typed as if it was a variable on its own. `numeric` Indexes of columns in the dataset To match column names with a character vector, use `matches("my_col_name")`. To match a logical vector, convert it to a column index using `which`.
`target`	DEPRECIATED. use "data" instead.

Value

An object of type `taxmap()`

See Also

Other taxmap manipulation functions: `arrange_obs()`, `arrange_taxa()`, `filter_obs()`, `filter_taxa()`, `mutate_obs()`, `sample_frac_obs()`, `sample_frac_taxa()`, `sample_n_obs()`, `sample_n_taxa()`, `transmute_obs()`

Examples

```
# Selecting a column by name
select_obs(ex_taxmap, "info", dangerous)

# Selecting a column by index
select_obs(ex_taxmap, "info", 3)
```

```
# Selecting a column by regular expressions
select_obs(ex_taxmap, "info", matches("^n"))
```

---

span *Span taxa*

Description

Select a range of taxa, either by two names, or relational operators

Usage

```
span(.data, ...)
```

Arguments

`.data`	Input, object of class `Hierarchy`, or `hierarchies`
`...`	quoted rank names (e.g., family) via `ranks()`, taxon names (e.g., Poa annua) via `nms()`, or taxonomic IDs (e.g., 93036) via `ids()`. You can't pass in arbitrary strings or numbers.

Details

supports `Hierarchy` and `hierarchies` objects

Value

an object of the same class as passed in

See Also

See filtering-helpers, including for more explanation of how this function works.

Examples

```
# Hierarchy class
ex_hierarchy1

## ranks
### keep all taxa between family and genus
span(ex_hierarchy1, ranks("family", "genus"))
span(ex_hierarchy1, nms("Poaceae", "Poa"))
span(ex_hierarchy1, ids(4479, 4544))

### keep all taxa between genus and species
span(ex_hierarchy1, ranks("genus", "species"))
```

```
### keep all taxa greater than genus
span(ex_hierarchy1, ranks("> genus"))

### keep all taxa greater than or equal to genus
span(ex_hierarchy1, ranks(">= genus"))

### keep all taxa less than genus
span(ex_hierarchy1, ranks("< genus"))

### keep all taxa less than or equal to genus
span(ex_hierarchy1, ranks("<= genus"))

### same as above, with different dataset
span(ex_hierarchy2, ranks("> genus"))
span(ex_hierarchy2, ranks(">= genus"))
span(ex_hierarchy2, ranks("< genus"))
span(ex_hierarchy2, ranks("<= genus"))

# using taxonomic names
span(ex_hierarchy2, nms("< Felidae"))

# using taxonomic ids
span(ex_hierarchy2, ids("< 9681"))

## Multiple operator statements - useful with larger classifications
ex_hierarchy3
span(ex_hierarchy3, ranks("> genus"), ranks("< phylum"))
span(ex_hierarchy3, ids("> 161994"), ids("< 158852"))


## taxon names
### keep all taxa between Poaceae and Poa
### - matches to ranks first
ex_hierarchy1 %>% span(nms("Poaceae", "Poa"))

## taxon ids
### keep all taxa between 4479 and 4544 taxonomic IDs
### - matches to ranks first
ex_hierarchy1 %>% span(ids(4479, 4544))


# hierarchies class
invisible(lapply(ex_hierarchies, print))
ex_hierarchies %>% span(ranks("family", "genus")) %>% lapply(., print) %>%
invisible
```

---

stems *Get stem taxa*

Description

Return the stem taxa for a taxonomy() or a taxmap() object. Stem taxa are all those from the roots to the first taxon with more than one subtaxon.

```
obj$stems(subset = NULL, simplify = FALSE,
  value = "taxon_indexes", exclude_leaves = FALSE)
stems(obj, subset = NULL, simplify = FALSE,
  value = "taxon_indexes", exclude_leaves = FALSE)
```

Arguments

obj	The taxonomy() or taxmap() object containing taxon information to be queried.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find stems for. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own.
value	What data to return. This is usually the name of column in a table in obj$data. Any result of all_names(obj) can be used, but it usually only makes sense to data that corresponds to taxa 1:1, such as taxon_ranks(). By default, taxon indexes are returned.
simplify	(logical) If TRUE, then combine all the results into a single vector of unique values.
exclude_leaves	(logical) If TRUE, the do not include taxa with no subtaxa.

Value

character

See Also

Other taxonomy indexing functions: branches(), internodes(), leaves(), roots(), subtaxa(), supertaxa()

Examples

```
# Return indexes of stem taxa
stems(ex_taxmap)

# Return indexes for a subset of taxa
stems(ex_taxmap, subset = 2:17)

# Return something besides taxon indexes
stems(ex_taxmap, value = "taxon_names")

# Return a vector instead of a list
stems(ex_taxmap, value = "taxon_names", simplify = TRUE)
```

subtaxa *Get subtaxa*

Description

Return data for the subtaxa of each taxon in an `taxonomy()` or `taxmap()` object.

```
obj$subtaxa(subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE, value = "taxon_indexes")
subtaxa(obj, subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE, value = "taxon_indexes")
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object containing taxon information to be queried.
`subset`	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find subtaxa for. Default: All taxa in `obj` will be used. Any variable name that appears in `all_names()` can be used as if it was a vector on its own.
`recursive`	(`logical` or `numeric`) If `FALSE`, only return the subtaxa one rank below the target taxa. If `TRUE`, return all the subtaxa of every subtaxa, etc. Positive numbers indicate the number of ranks below the immediate subtaxa to return. 1 is equivalent to `FALSE`. Negative numbers are equivalent to `TRUE`. Since the algorithm is optimized for traversing all of large trees, `numeric` values greater than 0 for this option actually take slightly longer to compute than either TRUE or FALSE.
`simplify`	(`logical`) If `TRUE`, then combine all the results into a single vector of unique values.
`include_input`	(`logical`) If `TRUE`, the input taxa are included in the output
`value`	What data to return. This is usually the name of column in a table in `obj$data`. Any result of `all_names()` can be used, but it usually only makes sense to data that corresponds to taxa 1:1, such as `taxon_ranks()`. By default, taxon indexes are returned.

Value

If `simplify = FALSE`, then a list of vectors are returned corresponding to the `target` argument. If `simplify = TRUE`, then the unique values are returned in a single vector.

See Also

Other taxonomy indexing functions: `branches()`, `internodes()`, `leaves()`, `roots()`, `stems()`, `supertaxa()`

Examples

```
# return the indexes for subtaxa for each taxon
subtaxa(ex_taxmap)

# Only return data for some taxa using taxon indexes
subtaxa(ex_taxmap, subset = 1:3)

# Only return data for some taxa using taxon ids
subtaxa(ex_taxmap, subset = c("d", "e"))

# Only return data for some taxa using logical tests
subtaxa(ex_taxmap, subset = taxon_ranks == "genus")

# Only return subtaxa one level below
subtaxa(ex_taxmap, recursive = FALSE)

# Only return subtaxa some number of ranks below
subtaxa(ex_taxmap, recursive = 2)

# Return something besides taxon indexes
subtaxa(ex_taxmap, value = "taxon_names")
```

---

subtaxa_apply *Apply function to subtaxa of each taxon*

Description

Apply a function to the subtaxa for each taxon. This is similar to using `subtaxa()` with `lapply()` or `sapply()`.

```
obj$subtaxa_apply(func, subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE, value = "taxon_indexes", ...)
subtaxa_apply(obj, func, subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE, value = "taxon_indexes", ...)
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object containing taxon information to be queried.
`func`	(`function`) The function to apply.
`subset`	Taxon IDs, TRUE/FALSE vector, or taxon indexes to use. Default: All taxa in `obj` will be used. Any variable name that appears in `all_names()` can be used as if it was a vector on its own.
`recursive`	(`logical` or `numeric`) If `FALSE`, only return the subtaxa one rank below the target taxa. If `TRUE`, return all the subtaxa of every subtaxa, etc. Positive numbers indicate the number of recursions (i.e. number of ranks below the target taxon to return). `1` is equivalent to `FALSE`. Negative numbers are equivalent to `TRUE`.

simplify	(logical) If TRUE, then combine all the results into a single vector of unique values.
include_input	(logical) If TRUE, the input taxa are included in the output
value	What data to give to the function. Any result of all_names(obj) can be used, but it usually only makes sense to use data that has an associated taxon id.
...	Extra arguments are passed to the function.

Examples

```
# Count number of subtaxa in each taxon
subtaxa_apply(ex_taxmap, length)

# Paste all the subtaxon names for each taxon
subtaxa_apply(ex_taxmap, value = "taxon_names",
              recursive = FALSE, paste0, collapse = ", ")
```

---

supertaxa *Get all supertaxa of a taxon*

Description

Return data for supertaxa (i.e. all taxa the target taxa are a part of) of each taxon in a taxonomy() or taxmap() object.

```
obj$supertaxa(subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE,
  value = "taxon_indexes", na = FALSE)
supertaxa(obj, subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE,
  value = "taxon_indexes", na = FALSE)
```

Arguments

obj	The taxonomy() or taxmap() object containing taxon information to be queried.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find supertaxa for. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own.
recursive	(logical or numeric) If FALSE, only return the supertaxa one rank above the target taxa. If TRUE, return all the supertaxa of every supertaxa, etc. Positive numbers indicate the number of recursions (i.e. number of ranks above the target taxon to return). 1 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
simplify	(logical) If TRUE, then combine all the results into a single vector of unique values.
include_input	(logical) If TRUE, the input taxa are included in the output
value	What data to return. Any result of all_names() can be used, but it usually only makes sense to use data that has an associated taxon id.
na	(logical) If TRUE, return NA where information is not available.

Value

If `simplify = FALSE`, then a list of vectors are returned corresponding to the `subset` argument. If `simplify = TRUE`, then unique values are returned in a single vector.

See Also

Other taxonomy indexing functions: `branches()`, `internodes()`, `leaves()`, `roots()`, `stems()`, `subtaxa()`

Examples

```
# return the indexes for supertaxa for each taxon
supertaxa(ex_taxmap)

# Only return data for some taxa using taxon indexes
supertaxa(ex_taxmap, subset = 1:3)

# Only return data for some taxa using taxon ids
supertaxa(ex_taxmap, subset = c("d", "e"))

# Only return data for some taxa using logical tests
supertaxa(ex_taxmap, subset = taxon_ranks == "species")

# Only return supertaxa one level above
supertaxa(ex_taxmap, recursive = FALSE)

# Only return supertaxa some number of ranks above
supertaxa(ex_taxmap, recursive = 2)

# Return something besides taxon indexes
supertaxa(ex_taxmap, value = "taxon_names")
```

---

supertaxa_apply *Apply function to supertaxa of each taxon*

Description

Apply a function to the supertaxa for each taxon. This is similar to using `supertaxa()` with `lapply()` or `sapply()`.

```
obj$supertaxa_apply(func, subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE, value = "taxon_indexes",
  na = FALSE, ...)
supertaxa_apply(obj, func, subset = NULL, recursive = TRUE,
  simplify = FALSE, include_input = FALSE, value = "taxon_indexes",
  na = FALSE, ....)
```

Arguments

obj	The taxonomy() or taxmap() object containing taxon information to be queried.
func	(function) The function to apply.
subset	Taxon IDs, TRUE/FALSE vector, or taxon indexes of taxa to use. Default: All taxa in obj will be used. Any variable name that appears in all_names() can be used as if it was a vector on its own.
recursive	(logical or numeric) If FALSE, only return the supertaxa one rank above the target taxa. If TRUE, return all the supertaxa of every supertaxa, etc. Positive numbers indicate the number of recursions (i.e. number of ranks above the target taxon to return). 1 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
simplify	(logical) If TRUE, then combine all the results into a single vector of unique values.
include_input	(logical) If TRUE, the input taxa are included in the output
value	What data to give to the function. Any result of all_names(obj) can be used, but it usually only makes sense to use data that has an associated taxon id.
na	(logical) If TRUE, return NA where information is not available.
...	Extra arguments are passed to the function.

Examples

```
# Get number of supertaxa that each taxon is contained in
supertaxa_apply(ex_taxmap, length)

# Get classifications for each taxon
# Note; this can be done with `classifications()` easier
supertaxa_apply(ex_taxmap, paste, collapse = ";", include_input = TRUE,
                value = "taxon_names")
```

---

taxa *A class for multiple taxon objects*

Description

Stores one or more taxon() objects. This is just a thin wrapper for a list of taxon() objects.

Usage

```
taxa(..., .list = NULL)
```

Arguments

...	Any number of object of class taxon()
.list	An alternate to the ... input. Any number of object of class taxon(). Cannot be used with ....

Details

This is the documentation for the class called `taxa`. If you are looking for the documentation for the package as a whole: taxa-package.

Value

An `R6Class` object of class `Taxon`

See Also

Other classes: `hierarchies()`, `hierarchy()`, `taxmap()`, `taxon_database()`, `taxon_id()`, `taxon_name()`, `taxon_rank()`, `taxonomy()`, `taxon()`

Examples

```
(a <- taxon(
  name = taxon_name("Poa annua"),
  rank = taxon_rank("species"),
  id = taxon_id(93036)
))
taxa(a, a, a)

# a null set
x <- taxon(NULL)
taxa(x, x, x)

# combo non-null and null
taxa(a, x, a)
```

`taxmap` *Taxmap class*

Description

A class designed to store a taxonomy and associated information. This class builds on the `taxonomy()` class. User defined data can be stored in the list `obj$data`, where `obj` is a taxmap object. Data that is associated with taxa can be manipulated in a variety of ways using functions like `filter_taxa()` and `filter_obs()`. To associate the items of lists/vectors with taxa, name them by `taxon_ids()`. For tables, add a column named `taxon_id` that stores `taxon_ids()`.

Usage

```
taxmap(..., .list = NULL, data = NULL, funcs = list(), named_by_rank = FALSE)
```

Arguments

...	Any number of object of class hierarchy() or character vectors.
.list	An alternate to the ... input. Any number of object of class hierarchy() or character vectors in a list. Cannot be used with ... .
data	A list of tables with data associated with the taxa.
funcs	A named list of functions to include in the class. Referring to the names of these in functions like filter_taxa() will execute the function and return the results. If the function has at least one argument, the taxmap object is passed to it.
named_by_rank	(TRUE/FALSE) If TRUE and the input is a list of vectors with each vector named by ranks, include that rank info in the output object, so it can be accessed by out$taxon_ranks(). If TRUE, taxa with different ranks, but the same name and location in the taxonomy, will be considered different taxa.

Details

To initialize a taxmap object with associated data sets, use the parsing functions parse_tax_data(), lookup_tax_data(), and extract_tax_data().

on initialize, function sorts the taxon list based on rank (if rank information is available), see ranks_ref for the reference rank names and orders

Value

An R6Class object of class taxmap()

See Also

Other classes: hierarchies(), hierarchy(), taxa(), taxon_database(), taxon_id(), taxon_name(), taxon_rank(), taxonomy(), taxon()

Examples

```
# The code below shows how to contruct a taxmap object from scratch.
# Typically, taxmap objects would be the output of a parsing function,
#  not created from scratch, but this is for demostration purposes.

notoryctidae <- taxon(
name = taxon_name("Notoryctidae"),
rank = taxon_rank("family"),
id = taxon_id(4479)
)
notoryctes <- taxon(
  name = taxon_name("Notoryctes"),
  rank = taxon_rank("genus"),
  id = taxon_id(4544)
)
typhlops <- taxon(
  name = taxon_name("typhlops"),
  rank = taxon_rank("species"),
  id = taxon_id(93036)
```

```
)
mammalia <- taxon(
  name = taxon_name("Mammalia"),
  rank = taxon_rank("class"),
  id = taxon_id(9681)
)
felidae <- taxon(
  name = taxon_name("Felidae"),
  rank = taxon_rank("family"),
  id = taxon_id(9681)
)
felis <- taxon(
  name = taxon_name("Felis"),
  rank = taxon_rank("genus"),
  id = taxon_id(9682)
)
catus <- taxon(
  name = taxon_name("catus"),
  rank = taxon_rank("species"),
  id = taxon_id(9685)
)
panthera <- taxon(
  name = taxon_name("Panthera"),
  rank = taxon_rank("genus"),
  id = taxon_id(146712)
)
tigris <- taxon(
  name = taxon_name("tigris"),
  rank = taxon_rank("species"),
  id = taxon_id(9696)
)
plantae <- taxon(
  name = taxon_name("Plantae"),
  rank = taxon_rank("kingdom"),
  id = taxon_id(33090)
)
solanaceae <- taxon(
  name = taxon_name("Solanaceae"),
  rank = taxon_rank("family"),
  id = taxon_id(4070)
)
solanum <- taxon(
  name = taxon_name("Solanum"),
  rank = taxon_rank("genus"),
  id = taxon_id(4107)
)
lycopersicum <- taxon(
  name = taxon_name("lycopersicum"),
  rank = taxon_rank("species"),
  id = taxon_id(49274)
)
tuberosum <- taxon(
  name = taxon_name("tuberosum"),
```

```
  rank = taxon_rank("species"),
  id = taxon_id(4113)
)
homo <- taxon(
  name = taxon_name("homo"),
  rank = taxon_rank("genus"),
  id = taxon_id(9605)
)
sapiens <- taxon(
  name = taxon_name("sapiens"),
  rank = taxon_rank("species"),
  id = taxon_id(9606)
)
hominidae <- taxon(
  name = taxon_name("Hominidae"),
  rank = taxon_rank("family"),
  id = taxon_id(9604)
)
unidentified <- taxon(
  name = taxon_name("unidentified")
)

tiger <- hierarchy(mammalia, felidae, panthera, tigris)
cat <- hierarchy(mammalia, felidae, felis, catus)
human <- hierarchy(mammalia, hominidae, homo, sapiens)
mole <- hierarchy(mammalia, notoryctidae, notoryctes, typhlops)
tomato <- hierarchy(plantae, solanaceae, solanum, lycopersicum)
potato <- hierarchy(plantae, solanaceae, solanum, tuberosum)
potato_partial <- hierarchy(solanaceae, solanum, tuberosum)
unidentified_animal <- hierarchy(mammalia, unidentified)
unidentified_plant <- hierarchy(plantae, unidentified)

info <- data.frame(stringsAsFactors = FALSE,
                   name = c("tiger", "cat", "mole", "human", "tomato", "potato"),
                   n_legs = c(4, 4, 4, 2, 0, 0),
                   dangerous = c(TRUE, FALSE, FALSE, TRUE, FALSE, FALSE))

abund <- data.frame(code = rep(c("T", "C", "M", "H"), 2),
                    sample_id = rep(c("A", "B"), each = 2),
                    count = c(1,2,5,2,6,2,4,0),
                    taxon_index = rep(1:4, 2))

phylopic_ids <- c("e148eabb-f138-43c6-b1e4-5cda2180485a",
                  "12899ba0-9923-4feb-a7f9-758c3c7d5e13",
                  "11b783d5-af1c-4f4e-8ab5-a51470652b47",
                  "9fae30cd-fb59-4a81-a39c-e1826a35f612",
                  "b6400f39-345a-4711-ab4f-92fd4e22cb1a",
                  "63604565-0406-460b-8cb8-1abe954b3f3a")

foods <- list(c("mammals", "birds"),
              c("cat food", "mice"),
              c("insects"),
              c("Most things, but especially anything rare or expensive"),
```

```
              c("light", "dirt"),
              c("light", "dirt"))

reaction <- function(x) {
  ifelse(x$data$info$dangerous,
         paste0("Watch out! That ", x$data$info$name, " might attack!"),
         paste0("No worries; its just a ", x$data$info$name, "."))
}

ex_taxmap <- taxmap(tiger, cat, mole, human, tomato, potato,
                    data = list(info = info,
                                phylopic_ids = phylopic_ids,
                                foods = foods,
                                abund = abund),
                    funcs = list(reaction = reaction))
```

`taxon` *Taxon class*

Description

A class used to define a single taxon. Most other classes in the taxa package include one or more objects of this class.

Usage

```
taxon(name, rank = NULL, id = NULL, authority = NULL)
```

Arguments

`name`	a TaxonName object `taxon_name()` or character string. if character passed in, we'll coerce to a TaxonName object internally, required
`rank`	a TaxonRank object `taxon_rank()` or character string. if character passed in, we'll coerce to a TaxonRank object internally, required
`id`	a TaxonId object `taxon_id()`, numeric/integer, or character string. if numeric/integer/character passed in, we'll coerce to a TaxonId object internally, required
`authority`	(character) a character string, optional

Details

Note that there is a special use case of this function - you can pass `NULL` as the first parameter to get an empty `taxon` object. It makes sense to retain the original behavior where nothing passed in to the first parameter leads to an error, and thus creating a `NULL` taxon is done very explicitly.

Value

An `R6Class` object of class `Taxon`

See Also

Other classes: hierarchies(), hierarchy(), taxa(), taxmap(), taxon_database(), taxon_id(), taxon_name(), taxon_rank(), taxonomy()

Examples

```
(x <- taxon(
  name = taxon_name("Poa annua"),
  rank = taxon_rank("species"),
  id = taxon_id(93036)
))
x$name
x$rank
x$id

# a null taxon object
taxon(NULL)
## with all NULL objects from the other classes
taxon(
  name = taxon_name(NULL),
  rank = taxon_rank(NULL),
  id = taxon_id(NULL)
)
```

---

taxonomy *Taxonomy class*

Description

Stores a taxonomy composed of taxon() objects organized in a tree structure. This differs from the hierarchies() class in how the taxon() objects are stored. Unlike hierarchies(), each taxon is only stored once and the relationships between taxa are stored in an edge list.

Usage

```
taxonomy(..., .list = NULL, named_by_rank = FALSE)
```

Arguments

...	Any number of object of class hierarchy() or character vectors.
.list	An alternate to the ... input. Any number of object of class hierarchy() or character vectors in a list. Cannot be used with ... .
named_by_rank	(TRUE/FALSE) If TRUE and the input is a list of vectors with each vector named by ranks, include that rank info in the output object, so it can be accessed by out$taxon_ranks(). If TRUE, taxa with different ranks, but the same name and location in the taxonomy, will be considered different taxa.

Value

An R6Class object of class Taxonomy

See Also

Other classes: hierarchies(), hierarchy(), taxa(), taxmap(), taxon_database(), taxon_id(), taxon_name(), taxon_rank(), taxon()

Examples

```
# Making a taxonomy object with vectors
taxonomy(c("mammalia", "felidae", "panthera", "tigris"),
         c("mammalia", "felidae", "panthera", "leo"),
         c("mammalia", "felidae", "felis", "catus"))

# Making a taxonomy object from scratch
#   Note: This information would usually come from a parsing function.
#         This is just for demonstration.
x <- taxon(
  name = taxon_name("Notoryctidae"),
  rank = taxon_rank("family"),
  id = taxon_id(4479)
)
y <- taxon(
  name = taxon_name("Notoryctes"),
  rank = taxon_rank("genus"),
  id = taxon_id(4544)
)
z <- taxon(
  name = taxon_name("Notoryctes typhlops"),
  rank = taxon_rank("species"),
  id = taxon_id(93036)
)

a <- taxon(
  name = taxon_name("Mammalia"),
  rank = taxon_rank("class"),
  id = taxon_id(9681)
)
b <- taxon(
  name = taxon_name("Felidae"),
  rank = taxon_rank("family"),
  id = taxon_id(9681)
)

cc <- taxon(
  name = taxon_name("Puma"),
  rank = taxon_rank("genus"),
  id = taxon_id(146712)
)
d <- taxon(
```

```
  name = taxon_name("Puma concolor"),
  rank = taxon_rank("species"),
  id = taxon_id(9696)
)

m <- taxon(
  name = taxon_name("Panthera"),
  rank = taxon_rank("genus"),
  id = taxon_id(146712)
)
n <- taxon(
  name = taxon_name("Panthera tigris"),
  rank = taxon_rank("species"),
  id = taxon_id(9696)
)

(hier1 <- hierarchy(z, y, x, a))
(hier2 <- hierarchy(cc, b, a, d))
(hier3 <- hierarchy(n, m, b, a))

(hrs <- hierarchies(hier1, hier2, hier3))

taxonomy(hier1, hier2, hier3)
```

`taxonomy_table` *Convert taxonomy info to a table*

Description

Convert per-taxon information, like taxon names, to a table of taxa (rows) by ranks (columns).

Arguments

`obj`	A `taxonomy` or `taxmap` object
`subset`	Taxon IDs, TRUE/FALSE vector, or taxon indexes to find supertaxa for. Default: All leaves will be used. Any variable name that appears in `all_names()` can be used as if it was a vector on its own.
`value`	What data to return. Default is taxon names. Any result of `all_names()` can be used, but it usually only makes sense to use data with one value per taxon, like taxon names.
`use_ranks`	Which ranks to use. Must be one of the following: • `NULL` (the default): If there is rank information, use the ranks that appear in the lineage with the most ranks. Otherwise, assume the number of supertaxa corresponds to rank and use placeholders for the rank column names in the output. • `TRUE`: Use the ranks that appear in the lineage with the most ranks. An error will occur if no rank information is available.

- FALSE: Assume the number of supertaxa corresponds to rank and use placeholders for the rank column names in the output. Do not use included rank information.
- character: The names of the ranks to use. Requires included rank information.
- numeric: The "depth" of the ranks to use. These are equal to n_supertaxa + 1.

add_id_col If TRUE, include a taxon ID column.

Value

A tibble of taxa (rows) by ranks (columns).

Examples

```
# Make a table of taxon names
taxonomy_table(ex_taxmap)

# Use a differnt value
taxonomy_table(ex_taxmap, value = "taxon_ids")

# Return a subset of taxa
taxonomy_table(ex_taxmap, subset = taxon_ranks == "genus")

# Use arbitrary ranks names based on depth
taxonomy_table(ex_taxmap, use_ranks = FALSE)
```

---

taxon_database *Taxonomy database class*

Description

Used to store information about taxonomy databases. This is typically used to store where taxon information came from in taxon() objects.

Usage

```
taxon_database(name = NULL, url = NULL, description = NULL, id_regex = NULL)
```

Arguments

name	(character) name of the database
url	(character) url for the database
description	(character) description of the database
id_regex	(character) id regex

Value

An R6Class object of class TaxonDatabase

See Also

database_list

Other classes: hierarchies(), hierarchy(), taxa(), taxmap(), taxon_id(), taxon_name(), taxon_rank(), taxonomy(), taxon()

Examples

```
# create a database entry
(x <- taxon_database(
  "ncbi",
  "http://www.ncbi.nlm.nih.gov/taxonomy",
  "NCBI Taxonomy Database",
  "*"
))
x$name
x$url

# use pre-created database objects
database_list
database_list$ncbi
```

---

taxon_id *Taxon ID class*

Description

Used to store taxon IDs, either arbitrary or from a taxonomy database. This is typically used to store taxon IDs in taxon() objects.

Usage

```
taxon_id(id, database = NULL)
```

Arguments

id	(character/integer/numeric) a taxonomic id, required
database	(database) database class object, optional

Value

An R6Class object of class TaxonId

See Also

Other classes: hierarchies(), hierarchy(), taxa(), taxmap(), taxon_database(), taxon_name(), taxon_rank(), taxonomy(), taxon()

Examples

```
(x <- taxon_id(12345))
x$id
x$database

(x <- taxon_id(
  12345,
  database_list$ncbi
))
x$id
x$database

# a null taxon_name object
taxon_name(NULL)
```

---

taxon_ids *Get taxon IDs*

Description

Return the taxon IDs in a taxonomy() or taxmap() object. They are in the order they appear in the edge list.

```
obj$taxon_ids()
taxon_ids(obj)
```

Arguments

obj	The taxonomy() or taxmap() object.

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_indexes(), taxon_names(), taxon_ranks()

Examples

```
# Return the taxon IDs for each taxon
taxon_ids(ex_taxmap)

# Filter using taxon IDs
filter_taxa(ex_taxmap, ! taxon_ids %in% c("c", "d"))
```

taxon_indexes *Get taxon indexes*

Description

Return the taxon indexes in a `taxonomy()` or `taxmap()` object. They are the indexes of the edge list rows.

```
obj$taxon_indexes()
taxon_indexes(obj)
```

Arguments

obj	The `taxonomy()` or `taxmap()` object.

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_branch()`, `is_internode()`, `is_leaf()`, `is_root()`, `is_stem()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_subtaxa()`, `n_supertaxa_1()`, `n_supertaxa()`, `taxon_ids()`, `taxon_names()`, `taxon_ranks()`

Examples

```
# Return the indexes for each taxon
taxon_indexes(ex_taxmap)

# Use in another function (stupid example; 1:5 would work too)
filter_taxa(ex_taxmap, taxon_indexes < 5)
```

taxon_name *Taxon name class*

Description

Used to store the name of taxa. This is typically used to store where taxon names in `taxon()` objects.

Usage

```
taxon_name(name, database = NULL)
```

Arguments

name	(character) a taxonomic name. required
database	(character) database class object, optional

Value

An R6Class object of class TaxonName

See Also

Other classes: hierarchies(), hierarchy(), taxa(), taxmap(), taxon_database(), taxon_id(), taxon_rank(), taxonomy(), taxon()

Examples

```
(poa <- taxon_name("Poa"))
(undef <- taxon_name("undefined"))
(sp1 <- taxon_name("species 1"))
(poa_annua <- taxon_name("Poa annua"))
(x <- taxon_name("Poa annua L."))

x$name
x$database

(x <- taxon_name(
  "Poa annua",
  database_list$ncbi
))
x$rank
x$database

# a null taxon_name object
taxon_name(NULL)
```

---

taxon_names *Get taxon names*

Description

Return the taxon names in a taxonomy() or taxmap() object. They are in the order they appear in the edge list.

```
obj$taxon_names()
taxon_names(obj)
```

Arguments

obj	The taxonomy() or taxmap() object.

See Also

Other taxonomy data functions: classifications(), id_classifications(), is_branch(), is_internode(), is_leaf(), is_root(), is_stem(), map_data_(), map_data(), n_leaves_1(), n_leaves(), n_subtaxa_1(), n_subtaxa(), n_supertaxa_1(), n_supertaxa(), taxon_ids(), taxon_indexes(), taxon_ranks()

Examples

```
# Return the names for each taxon
taxon_names(ex_taxmap)

# Filter by taxon name
filter_taxa(ex_taxmap, taxon_names == "Felidae", subtaxa = TRUE)
```

`taxon_rank` *Taxon rank class*

Description

Stores the rank of a taxon. This is typically used to store where taxon information came from in `taxon()` objects.

Usage

```
taxon_rank(name, database = NULL)
```

Arguments

`name`	(character) rank name. required
`database`	(character) database class object, optional

Value

An `R6Class` object of class `TaxonRank`

See Also

Other classes: `hierarchies()`, `hierarchy()`, `taxa()`, `taxmap()`, `taxon_database()`, `taxon_id()`, `taxon_name()`, `taxonomy()`, `taxon()`

Examples

```
taxon_rank("species")
taxon_rank("genus")
taxon_rank("kingdom")

(x <- taxon_rank(
  "species",
  database_list$ncbi
))
x$rank
x$database

# a null taxon_name object
taxon_name(NULL)
```

taxon_ranks *Get taxon ranks*

Description

Return the taxon ranks in a `taxonomy()` or `taxmap()` object. They are in the order taxa appear in the edge list.

```
obj$taxon_ranks()
taxon_ranks(obj)
```

Arguments

`obj`	The `taxonomy()` or `taxmap()` object.

See Also

Other taxonomy data functions: `classifications()`, `id_classifications()`, `is_branch()`, `is_internode()`, `is_leaf()`, `is_root()`, `is_stem()`, `map_data_()`, `map_data()`, `n_leaves_1()`, `n_leaves()`, `n_subtaxa_1()`, `n_subtaxa()`, `n_supertaxa_1()`, `n_supertaxa()`, `taxon_ids()`, `taxon_indexes()`, `taxon_names()`

Examples

```
# Get ranks for each taxon
taxon_ranks(ex_taxmap)

# Filter by rank
filter_taxa(ex_taxmap, taxon_ranks == "family", supertaxa = TRUE)
```

transmute_obs *Replace columns in `taxmap()` objects*

Description

Replace columns of tables in `obj$data` in `taxmap()` objects. See `dplyr::transmute()` for the inspiration for this function and more information. Calling the function using the `obj$transmute_obs(...)` style edits "obj" in place, unlike most R functions. However, calling the function using the `transmute_obs(obj,...)` imitates R's traditional copy-on-modify semantics, so "obj" would not be changed; instead a changed version would be returned, like most R functions.

```
obj$transmute_obs(data, ...)
transmute_obs(obj, data, ...)
```

Arguments

obj	An object of type taxmap()
data	Dataset name, index, or a logical vector that indicates which dataset in obj$data to use.
...	One or more named columns to add. Newly created columns can be referenced in the same function call. Any variable name that appears in all_names() can be used as if it was a vector on its own.
target	DEPRECIATED. use "data" instead.

Value

An object of type taxmap()

See Also

Other taxmap manipulation functions: arrange_obs(), arrange_taxa(), filter_obs(), filter_taxa(), mutate_obs(), sample_frac_obs(), sample_frac_taxa(), sample_n_obs(), sample_n_taxa(), select_obs()

Examples

```
# Replace columns in a table with new columns
transmute_obs(ex_taxmap, "info", new_col = paste0(name, "!!!"))
```

Index