Package ‘rcol’

February 5, 2021

Type Package
Title Catalogue of Life Client
Description Client for the Catalogue of Life ('CoL')
   (<https://www.catalogueoflife.org/>); based on the new
   'CoL' service, not the old one. Catalogue of Life is a database of
taxonomic names. Includes functions for each
   of the API methods, including searching for names, and more.
Version 0.1.0
License MIT + file LICENSE
URL https://docs.ropensci.org/rcol/ (docs),
       https://github.com/ropensci/rcol
BugReports https://github.com/ropensci/rcol/issues
LazyData true
Encoding UTF-8
Language en-US
Imports crul, tibble, jsonlite, data.table, glue
Suggests testthat, vcr
RoxygenNote 7.1.1
X-schema.org-applicationCategory Biology
X-schema.org-keywords biology, science, API, web, api-client,
taxonomy, species
X-schema.org-isPartOf https://ropensci.org
NeedsCompilation no
Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>),
       rOpenSci [fnd] (https://ropensci.org/)
Maintainer Scott Chamberlain <myrmecocystus@gmail.com>
Repository CRAN
Date/Publication 2021-02-05 08:30:03 UTC
R topics documented:

- rcol-package .................................................. 2
- cp_children ................................................... 2
- cp_classification ............................................ 3
- cp_datasets ................................................... 4
- cp_ds ........................................................... 5
- cp_importer .................................................... 6
- cp_name_match ............................................... 7
- cp_nu_search .................................................. 8
- cp_nu_suggest ................................................. 11
- cp_parser ....................................................... 12
- cp_vocab ....................................................... 12

Index

rcol-package rcol

Description

Catalogue of Life (CoL) Client

Note

CoL API docs: https://api.catalogueoflife.org/

Author(s)

Scott Chamberlain

Description

Children

Usage

   cp_children(dataset_key, taxon_id, ...)

Arguments

- dataset_key  (character/integer/numeric) dataset identifier
- taxon_id     (character/integer/numeric) taxon identifier
- ...         curl options passed on to curl::verb-GET
Value

- result (data.frame/tibble): results, a zero row data.frame if no results found
- meta (data.frame/tibble): number of results found

Examples

```r
cchk <- function(x) {
  z <- tryCatch(crul::ok(x), error = function(e) e)
  if (inherits(z, "error")) FALSE else z
}
if (chk("https://api.catalogueoflife.org/version")) {
  z <- cp_children(dataset_key=1000, taxon_id='1')
  z
  z$result
  if (NROW(z$result) > 0) {
    z$result$scientificName
    z$result$created
  }
}
```

---

**Description**

Classification

**Usage**

```r
cp_classification(dataset_key, taxon_id, ...)
```

**Arguments**

- **dataset_key** (character/integer/numeric) dataset identifier
- **taxon_id** (character/integer/numeric) taxon identifier
- ... curl options passed on to `crul::verb-GET`

**Value**

a data.frame/tibble with results, a zero row data.frame if no results found
Examples

```r
if (cp_up("/dataset/1000/taxon/10/classification")) {
  cp_classification(dataset_key=1000, taxon_id=10)
}
```

```r
## Not run:
cp_classification(dataset_key=1000, taxon_id=20)
cp_classification(dataset_key=3,
  taxon_id="6565450e-1cf2-4dc2-acbb-db728e42e635")

## End(Not run)
```

### cp_datasets

**Datasets**

#### Description

Datasets

#### Usage

```r
cp_datasets(q = NULL, start = 0, limit = 10, ...)
cp_dataset(dataset_keys, ...)
```

#### Arguments

- `q` (character) main query string. optional
- `start` (integer) requested number of offset records. Default: 0
- `limit` (integer) requested number of maximum records to be returned. Default: 10; max: 1000
- `...` curl options passed on to verb-GET
- `dataset_keys` (character) one or more dataset keys. required

#### Details

for `cp_dataset()`, separate http requests are made for each dataset key. unfortunately, the output of `cp_dataset()` is a list for each dataset key because the nested structure of the data is hard to rectangularize

#### Value

list with two slots

- `result` (data.frame/tibble): results, a zero row data.frame if no results found
- `meta` (data.frame/tibble): number of results found
### Examples

```r
if (cp_up("/dataset")) {
  cp_datasets(limit = 1)
}
## Not run:
cp_datasets(q = "life")
cp_dataset(dataset_keys = 1000)
cp_dataset(dataset_keys = c(3, 1000, 1014))
## End(Not run)
```

---

### cp_ds

**Datasets API route catch all method**

**Description**

Datasets API route catch all method

**Usage**

```r
cp_ds(route, ..., .list = list())
```

**Arguments**

- `route` (character) an API route. the /dataset route part is added internally; so just include the route following that. required.
- `...` named parameters, passed on to `glue::glue()`. required. param names must match must match names given in the route. For example, if you have route = \{key\}/name\{id\}, then you need to pass in a key and an id parameter. The names in the route (here, key and id) don’t have to match the names in the API route you are trying to use - they just need to match the named parameters you pass in. Having said that, it may be easier to remember what you’re doing if you match the names to the route parts.
- `.list` a named list. instead of passing in named parameters through ..., you can pre-prepare a named list and give to this parameter

**Details**

There are A LOT of datasets API routes. Instead of making an R function for each route, we have R functions for some of the "more important" routes, then `cp_ds()` will allow you to make requests to the remainder of the datasets API routes.

**Value**

output varies depending on the route requested, but output will always be a named list. when no results found, an error message will be returned
Not supported dataset routes

Some dataset routes do not return JSON so we don’t support those. Thus far, the only route we don’t support is /dataset/\{key\}/logo

Examples

```r
## Not run:
cp_ds(route = "{key}/tree", key = "1000")
cp_ds(route = "{key}/tree", key = "1014")
cp_ds(route = "{key}/name/{id}", key = 1005, id = 100003)

# pass a named list to the .list parameter
args <- list(key = 1005, id = 100003)
cp_ds("(key)/name/[id]", .list = args)

## End(Not run)
```

---

**cp_importer**

**Importer metrics**

**Description**

Importer metrics

**Usage**

```r
cp_importer(
  dataset_key = NULL,
  state = NULL,
  running = FALSE,
  start = 0,
  limit = 10,
  ...
)
```

**Arguments**

- **dataset_key** (character) a dataset key to filter by. optional
- **state** (character) filter listed import metrics by their state, e.g. the last failed import. one of: downloading, processing, inserting, unchanged, finished, canceled, failed. optional
- **running** (logical) if only a list of running imports should be returned. default: FALSE. optional
- **start** (integer) requested number of offset records. Default: 0
- **limit** (integer) requested number of maximum records to be returned. Default: 10; max: 1000
- ... curl options passed on to verb-GET
**Value**

A named list, with slots `offset` (integer), `limit` (integer), `total` (integer), `result` (list), `empty` (boolean), and `last` (boolean). The `result` slot is a list itself, with any number of results as named lists.

**Examples**

```r
if (cp_up("/importer?limit=1")) {
  cp_importer(limit = 1)
}
```

---

**Description**

Match name against the name index.

**Usage**

```r
cp_name_match(
  q = NULL,
  rank = NULL,
  code = NULL,
  trusted = NULL,
  ver_bose = NULL,
  start = 0,
  limit = 10,
  ...
)
```

**Arguments**

- `q` (character) scientific name to match
- `rank` (character) rank to restrict matches to. One of: domain, realm, subrealm, superkingdom, kingdom, subkingdom, infrakingdom, superphylum, phylum, subphylum, infraphylum, superclass, class, subclass, infraclass, parclass, superdivision, division, subdivision, infradivision, superlegion, legion, sublegion, infralegion, supercohort, cohort, subcohort, infracohort, gigaorder, magnorder, grandorder, mirorder, superorder, order, nanorder, hypoorder, minorder, suborder, infraorder, parvorder, megafamily, grandfamily, superfamily, epifamily, family, subfamily, infrafamily, supertribe, tribe, subtribe, infratribe, suprageneric_name, genus, subgenus, infrageneric_name, supersection, section, subsection, superseries, series, subseries, infrageneric_name, species_aggregate, species, infraspecific_name, grex, subspecies, cultivar_group, convariety, infrasubspecific_name, proles, natio, aberration, morph, variety, subvariety, form, subform, pathovar, biovar, chemovar, morphovar, phagovar, serovar, chemoform, forma_specialis, cultivar, strain, other, unranked
code  (character) nomenclatural code to restrict matches to. one of: bacterial, botanical, cultivars, viral, zoological, phytosociological

trusted  (logical) if TRUE, unmatched name will be inserted into the names index. default: FALSE

ver_bose  (logical) if TRUE, list alternatively considered name matches. default: FALSE

start  (integer) requested number of offset records. Default: 0

limit  (integer) requested number of maximum records to be returned. Default: 10; max: 1000

...  curl options passed on to verb=GET

Details

Matches by the canonical name, it’s authorship and rank. Authorship matching is somewhat loose, but name matching is quite strict and only allows for a few common misspellings frequently found in epithets (silent h, gender suffix, double letters, i/y), but not in uninomials. Suprageneric ranks are all considered to be the same, otherwise a different rank results in a different match.

Value

a named list, with slots name (list), type (character), alternatives (data.frame), and nameKey (integer)

Examples

```r
if (cp_up("/name/matching?q=Apis")) {
  cp_name_match(q="Apis")
}
## Not run:
  cp_name_match(q="Agapostemon")
  cp_name_match(q="Apis mellifera")
  cp_name_match(q="Apis mellifer") # no fuzzy match apparently
## End(Not run)
```

---

Name Usage: Search

Description

Name Usage: Search
Usage

```r
cp_nu_search(
  q = NULL,
  dataset_key = NULL,
  min_rank = NULL,
  max_rank = NULL,
  content = NULL,
  highlight = NULL,
  reverse = NULL,
  fuzzy = NULL,
  type = NULL,
  nomstatus = NULL,
  status = NULL,
  issue = NULL,
  published_in = NULL,
  facet = NULL,
  sortBy = NULL,
  start = 0,
  limit = 10,
  ...
)
```

Arguments

- `q` (character) vector of one or more scientific names
- `dataset_key` (character) dataset key
- `min_rank, max_rank` (character) filter by rank. one of: domain, superkingdom, kingdom, subkingdom, infrakingdom, superphylum, phylum, subphylum, infraphylum, superclass, class, subclass, infraclass, parvclass, superlegion, legion, sublegion, infralegion, supercohort, cohort, subcohort, infracohort, magnorder, superorder, grandorder, order, suborder, infraorder, parvorder, superfAMILY, family, subfamily, infrafamily, superfAMILY, tribe, subtribe, infratribe, suprageneric name, genus, subgenus, infragenus, supersection, section, subsection, superset, series, subseries, infraspecific name, species aggregate, species, infraspecific name, grex, subspecies, cultivar group, congeneric, infrasubspecific name, proles, natio, aberration, morph, variety, subvariety, form, subform, pathovar, biovar, chemovar, morphovar, phagovar, serovar, chemoform, forma specialis, cultivar, strain, other, unranked
- `content` (character) one of: 'scientific_name' or 'authorship'
- `highlight` (logical) TRUE or FALSE. default: NULL
- `reverse` (logical) TRUE or FALSE. default: NULL
- `fuzzy` (logical) TRUE or FALSE. default: NULL
- `type` (character) one of: 'prefix', 'whole_words', 'exact'
- `nomstatus` (character) filter by nomenclatural status. one of: ok, unavailable, illegitimate, variant, conserved, rejected, doubtful, unevaluated
status (character) filter by taxonomic status. one of: accepted, doubtful, ambiguous synonym

issue (character) filter by issue found

published_in (character) reference id to filter names by

facet (character) request a facet to be returned. one of: dataset_key, rank, nom_status, status, issue, type, field. facet limit default: 50

sortBy (character) one of: "relevance", "name", "taxonomic", "index_name_id", or "native"

start (integer) requested number of offset records. Default: 0

limit (integer) requested number of maximum records to be returned. Default: 10; max: 1000

... curl options passed on to verb=GET

Value

list with two slots

- result (data.frame/tibble): results, a zero row data.frame if no results found
- meta (data.frame/tibble): number of results found

Examples

```r
if (cp_up("/nameusage/search?q=Apis")) {
  cp_nu_search(q="Apis", limit = 1)
}
## Not run:
cp_nu_search(q="Agapostemon")
cp_nu_search(q="Agapostemon", dataset_key = 3)
cp_nu_search(q="Agapostemon", min_rank = "genus")
cp_nu_search(q="Agapostemon", nomstatus = "doubtful")
cp_nu_search(q="Agapostemon", status = "accepted")
cp_nu_search(q="Bombus", facet = "rank")
cp_nu_search(q="Agapostemon", dataset_key = 3, hasField="uninomial")

x <- cp_nu_search(q="Poa")
x
x$result
x$result$usage
x$result$usage$name

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

**Name Usage: Suggest**

#### Description

Name Usage: Suggest

#### Usage

```r
cp_nu_suggest(
  q,
  dataset_key,
  fuzzy = FALSE,
  min_rank = NULL,
  max_rank = NULL,
  sort = NULL,
  reverse = FALSE,
  accepted = FALSE,
  limit = 10,
  ...
)
```

#### Arguments

- **q** (character) main query string. required
- **dataset_key** (character) a dataset key. required
- **fuzzy** (logical) Whether or not to do fuzzy search (default: FALSE)
- **min_rank, max_rank** (character) See rank options in `cp_name_match()`
- **sort** (character) one of name, taxonomic, index_name_id, native, relevance
- **reverse** (logical) reverse order i assume (default: FALSE)
- **accepted** (logical) limit to accepted names (default: FALSE)
- **limit** (integer) requested number of maximum records to be returned. Default: 10; max: 1000
- **...** curl options passed on to `crl::verb-GET`

#### Value

a data.frame/tibble of results. a zero row data.frame if no results

#### Examples

```r
if (cp_up("/dataset/3/nameusage/suggest?q=Apis")) {
  cp_nu_suggest(q="Apis", 3)
}
```
### cp_parser  
Name Parser

**Description**
Name Parser

**Usage**
```r
cp_parser(names, ...)
```

**Arguments**
- `names` (character) one or more scientific names to parse
- `...` curl options passed on to `crul::verb-POST`

**Value**
tibble, with one row for each parsed name

**Examples**
```r
## Not run:
cp_parser(names = "Apis mellifera")
cp_parser(names = c("Apis mellifera", "Homo sapiens var. sapiens"))
## End(Not run)
```

### cp_vocab  
CoL Vocabularies

**Description**
CoL Vocabularies

**Usage**
```r
cp_vocab(vocab, ...)
```

**Arguments**
- `vocab` (character) a vocabulary name
- `...` curl options passed on to `crul::verb-GET`

**Value**
character vector of words
Examples

```r
## Not run:
cp_vocab("rank")
cp_vocab("datasetorigin")
cp_vocab("datasettype")
cp_vocab("matchtype")
cp_vocab("taxonomicstatus")

## End(Not run)
```
Index

* package
  rcol-package, 2

cp_children, 2
cp_classification, 3
cp_dataset (cp_datasets), 4
cp_datasets, 4
cp_ds, 5
cp_importer, 6
cp_name_match, 7
cp_name_match(), 11
cp_nu_search, 8
cp_nu_suggest, 11
cp_parser, 12
cp_vocab, 12
crul::verb-GET, 2, 3, 11, 12
crul::verb-POST, 12
glue::glue(), 5
rcol (rcol-package), 2
rcol-package, 2