Package ‘rgbif’

January 11, 2024

Title Interface to the Global Biodiversity Information Facility API

Description A programmatic interface to the Web Service methods provided by the Global Biodiversity Information Facility (GBIF: <https://www.gbif.org/developer/summary>). GBIF is a database of species occurrence records from sources all over the globe. rgbif includes functions for searching for taxonomic names, retrieving information on data providers, getting species occurrence records, getting counts of occurrence records, and using the GBIF tile map service to make rasters summarizing huge amounts of data.

Version 3.7.9

License MIT + file LICENSE

URL https://github.com/ropensci/rgbif (devel), https://docs.ropensci.org/rgbif/ (documentation)

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

LazyLoad true

Encoding UTF-8

Language en-US

Imports xml2, ggplot2, crul (>= 0.7.4), data.table, whisker, magrittr, jsonlite (>= 1.6), oai (>= 0.2.2), tibble, lazyeval, R6, stats, wk

Suggests testthat, png, terra, magick, protolite (>= 2.0), sf, vcr (>= 1.2.0), knitr, rmarkdown

RoxygenNote 7.2.3

X-schema.org-applicationCategory Biodiversity

X-schema.org-keywords GBIF, specimens, API, web-services, occurrences, species, taxonomy

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

NeedsCompilation no
Author  Scott Chamberlain [aut] (https://orcid.org/0000-0003-1444-9135),
       Damiano Oldoni [aut] (https://orcid.org/0000-0003-3445-7562),
       Vijay Barve [ctb] (https://orcid.org/0000-0002-4852-2567),
       Peter Desmet [ctb] (https://orcid.org/0000-0002-8442-8025),
       Laurens Geffert [ctb],
       Dan McGlinn [ctb] (https://orcid.org/0000-0003-3445-7562),
       Karthik Ram [ctb] (https://orcid.org/0000-0002-0233-1757),
       rOpenSci [fnd] (https://ropensci.org/),
       John Waller [aut, cre] (https://orcid.org/0000-0002-7302-5976)
Maintainer  John Waller <jwaller@gbif.org>
Repository  CRAN
Date/Publication  2024-01-11 08:40:02 UTC

R topics documented:

rgbif-package .................................................  3
check_wkt ..................................................  4
count_facet ..................................................  5
dataset ......................................................  6
datasets ......................................................  8
dataset_doi ...................................................  9
dataset_gridded .............................................. 10
dataset_list_funs ........................................... 11
dataset_search .............................................. 12
dataset_uuid_funs .......................................... 16
derived_dataset ............................................. 17
downloads .................................................... 20
download_predicate_dsl .................................... 22
elevation ..................................................... 27
enumeration .................................................. 30
gbif_bbox2wkt ............................................... 31
gbif_citation ............................................... 32
gbif_geocode ............................................... 33
gbif_issues .................................................. 34
gbif_issues_lookup ......................................... 34
gbif_names .................................................. 35
gbif_oai ...................................................... 36
gbif_photos .................................................. 38
installations ............................................... 39
lit_search ................................................... 40
map_fetch .................................................... 43
mvt_fetch .................................................... 47
name_backbone .............................................. 49
name_backbone_checklist .................................. 52
name_issues ................................................ 55
name_lookup ................................................ 56
name_parse .................................................. 61
Description

rgbif: A programmatic interface to the Web Service methods provided by the Global Biodiversity Information Facility.

About

This package gives you access to data from GBIF https://www.gbif.org/ via their API.
check_wkt

Documentation for the GBIF API

- summary https://www.gbif.org/developer/summary - Summary of the GBIF API
- registry https://www.gbif.org/developer/registry - Metadata on datasets, and contributing organizations
- species names https://www.gbif.org/developer/species - Species names and metadata
- occurrences https://www.gbif.org/developer/occurrence - Occurrences
- maps https://www.gbif.org/developer/maps - Maps - these APIs are not implemented in rgbif, and are meant more for integration with web based maps.

Author(s)

Scott Chamberlain
Karthik Ram
Dan Mcglinn
Vijay Barve
John Waller

---

check_wkt  Check input WKT

Description

Check input WKT

Usage

check_wkt(wkt = NULL, skip_validate = FALSE)

Arguments

wkt (character) one or more Well Known Text objects
skip_validate (logical) whether to skip wk::wk_problems call or not. Default: FALSE

Examples

## Not run:
check_wkt('POLYGON((30.1 10.1, 10 20, 20 60, 60 60, 30.1 10.1))')
check_wkt('POINT(30.1 10.1)')
check_wkt('LINESTRING(3 4,10 50,20 25)')

# check many passed in at once
check_wkt(c('POLYGON((30.1 10.1, 10 20, 20 60, 60 60, 30.1 10.1))',
'POINT(30.1 10.1)'))

# bad WKT
count_facet

Facetted count occurrence search.

Description

Facetted count occurrence search.

Usage

count_facet(keys = NULL, by = "country", countries = 10, removezeros = FALSE)

Arguments

keys (numeric) GBIF keys, a vector. optional
by (character) One of georeferenced, basisOfRecord, country, or publishingCountry. default: country
countries (numeric) Number of countries to facet on, or a vector of country names. default: 10
removezeros (logical) remove zeros or not? default: FALSE

Examples

## Not run:
# Select number of countries to facet on
count_facet(by='country', countries=3, removezeros = TRUE)
# Or, pass in country names
count_facet(by='country', countries='AR', removezeros = TRUE)

spplist <- c('Geothlypis trichas','Tiaris olivacea','Pterodroma axillaris','Calidris ferruginea','Pterodroma macroptera','Gallirallus australis','Falco cenchroides','Telespiza cantans','Oreomystis bairdi','Cistothorus palustris')
keys <- sapply(spplist,
  function(x) name_backbone(x, rank="species")$usageKey)
count_facet(keys, by='country', countries=3, removezeros = TRUE)
count_facet(keys, by='country', countries=3, removezeros = FALSE)
count_facet(by='country', countries=20, removezeros = TRUE)
count_facet(keys, by='basisOfRecord', countries=5, removezeros = TRUE)
dataset

Search for more obscure dataset metadata.

Description

Search for more obscure dataset metadata.

Usage

dataset(
country = NULL,
type = NULL,
identifierType = NULL,
identifier = NULL,
machineTagNamespace = NULL,
machineTagName = NULL,
machineTagValue = NULL,
modified = NULL,
query = NULL,
deleted = FALSE,
limit = NULL,
start = NULL,
curlopts = list()
)

Arguments

country
  The 2-letter country code (as per ISO-3166-1) of the country publishing the dataset.

type
  The primary type of the dataset. Available values: OCCURRENCE, CHECKLIST, METADATA, SAMPLING_EVENT, MATERIAL_ENTITY.

identifierType
  An identifier type for the identifier parameter. Available values: URL, LSID, HANDLER, DOI, UUID, FTP, URI, UNKNOWN, GBIF_PORTAL, GBIF_NODE, GBIF_PARTICIPANT, GRSCICOLL_ID, GRSCICOLL_URI, IH_IRN, ROR, GRID, CITES, SYMBIOTA_UUID, WIKIDATA, NCBI_BIOCOLLECTION.
### dataset

**identifier**  
An identifier of the type given by the identifierType parameter.

**machineTagNamespace**  
Filters for entities with a machine tag in the specified namespace.

**machineTagName**  
Filters for entities with a machine tag with the specified name (use in combination with the machineTagNamespace parameter).

**machineTagValue**  
Filters for entities with a machine tag with the specified value (use in combination with the machineTagNamespace and machineTagName parameters).

**modified**  
The modified date of the dataset. Accepts ranges and a " can be used as a wildcard, e.g.: modified=2023-04-01,

**query**  
Simple full text search parameter. The value for this parameter can be a simple word or a phrase. Wildcards are not supported.

**deleted**  
Logical specifying whether to return only deleted datasets.

**limit**  
Controls the number of results in the page.

**start**  
Determines the start for the search results.

**curl::options**  
options passed on to crul::HttpClient.

### Details

This function allows you to search for some more obscure dataset metadata that might not be possible with dataset_search(). For example, searching through registry machinetags.

### Value

A list.

### Examples

```r
## Not run:
dataset(limit=3)
dataset(country="US",limit=3)
dataset(type="CHECKLIST",limit=3)
dataset(identifierType = "URL",limit=3)
dataset(identifier = 168,limit=3)
dataset(machineTagNamespace = "metasync.gbif.org",limit=3)
dataset(machineTagName = "datasetTitle",limit=3)
dataset(machineTagValue = "Borkhart",limit=3)
dataset(modified = "2023-04-01", limit=3)
dataset(q = "dog", limit=3)
dataset(deleted=TRUE,limit=3)

## End(Not run)
```
datasets

Search for datasets and dataset metadata.

Description
Search for datasets and dataset metadata.

Usage

datasets(
  data = "all",
  type = NULL,
  uuid = NULL,
  query = NULL,
  id = NULL,
  limit = 100,
  start = NULL,
  curlopts = list()
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The type of data to get. One or more of: 'organization', 'contact', 'endpoint', 'identifier', 'tag', 'machinetag', 'comment', 'constituents', 'document', 'metadata', 'deleted', 'duplicate', 'subDataset', 'withNoEndpoint', or the special 'all'. Default: all</td>
</tr>
<tr>
<td>type</td>
<td>Type of dataset. Options: include occurrence, checklist, metadata, or sampling_event.</td>
</tr>
<tr>
<td>uuid</td>
<td>UUID of the data node provider. This must be specified if data is anything other than all</td>
</tr>
<tr>
<td>query</td>
<td>Query term(s). Only used when data=all</td>
</tr>
<tr>
<td>id</td>
<td>A metadata document id.</td>
</tr>
<tr>
<td>limit</td>
<td>Number of records to return. Default: 100. Maximum: 1000.</td>
</tr>
<tr>
<td>start</td>
<td>Record number to start at. Default: 0. Use in combination with limit to page through results.</td>
</tr>
<tr>
<td>curlopts</td>
<td>list of named curl options passed on to HttpClient. see curl::curl_options for curl options</td>
</tr>
</tbody>
</table>

Value
A list.

References

https://www.gbif.org/developer/registry#datasets
Examples

```r
## Not run:
datasets(limit=5)
datasets(type="occurrence", limit=10)
datasets(uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
datasets(data=
/quotesingle.Var
contact
/quotesingle.Var
, uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
datasets(data=
/quotesingle.Var
metadata
/quotesingle.Var
, uuid="a6998220-7e3a-485d-9cd6-73076bd85657",
id=598)
datasets(data=c('deleted','duplicate'))
datasets(data=c('deleted','duplicate'), limit=1)

# curl options
datasets(data=c('deleted','duplicate'), curlopts = list(verbosel=TRUE))

## End(Not run)
```

---

**dataset_doi**

*Get a GBIF dataset from a doi*

**Description**

Get a GBIF dataset from a doi

**Usage**

```r
dataset_doi(doi = NULL, limit = 20, start = NULL, curlopts = list())
```

**Arguments**

- `doi` the doi of the dataset you wish to lookup.
- `limit` Controls the number of results in the page.
- `start` Determines the offset for the search results.
- `curlopts` options passed on to `cru::HttpClient`.

**Details**

This function allows for dataset lookup using a doi. Be aware that some doi have more than one dataset associated with them.

**Value**

A list.
**dataset_gridded**

Check if a dataset is gridded

## Not run:
```r
dataset_doi(‘10.15468/igasai’)
```

## End(Not run)

### Description

Check if a dataset is gridded

### Usage

```r
dataset_gridded(
  uuid = NULL,
  min_dis = 0.05,
  min_per = 50,
  min_dis_count = 30,
  return = “logical”,
  warn = TRUE
)
```

### Arguments

- **uuid** (vector) A character vector of GBIF datasetkey uuids.
- **min_dis** (numeric) (default 0.02) Minimum distance in degrees to accept as gridded.
- **min_per** (integer) (default 50%) Minimum percentage of points having same nearest neighbor distance to be considered gridded.
- **min_dis_count** (default 30) Minimum number of unique points to accept an assessment of 'grid-dyness'.
- **return** (character) (default "logical"). Choice of "data" will return a data.frame of more information or "logical" will return just TRUE or FALSE indicating whether a dataset is considered 'gridded'.
- **warn** (logical) indicates whether to warn about missing values or bad values.

### Details

Gridded datasets are a known problem at GBIF. Many datasets have equally-spaced points in a regular pattern. These datasets are usually systematic national surveys or data taken from some atlas (“so-called rasterized collection designs”). This function uses the percentage of unique lat-long points with the most common nearest neighbor distance to identify gridded datasets.


I recommend keeping the default values for the parameters.
**dataset_list_funs**

**Value**

A logical vector indicating whether a dataset is considered gridded. Or if `return="data"`, a `data.frame` of more information.

**Examples**

```r
## Not run:
dataset_gridded("9070a460-0c6e-11dd-84d2-b8a03c50a862")
dataset_gridded(c("9070a460-0c6e-11dd-84d2-b8a03c50a862",
"13b70480-bd69-11dd-b15f-b8a03c50a862"))
```

## End(Not run)

**dataset_list_funs**  
_List datasets that are deleted or have no endpoint._

**Description**

List datasets that are deleted or have no endpoint.

**Usage**

```r
dataset_duplicate(limit = 20, start = NULL, curlopts = list())
dataset_noendpoint(limit = 20, start = NULL, curlopts = list())
```

**Arguments**

- `limit` Controls the number of results in the page.
- `start` Determines the start for the search results.
- `curlopts` options passed on to `crul::HttpClient`.

**Details**

Get a list of deleted datasets or datasets with no endpoint. You get the full and no parameters aside from `limit` and `start` are accepted.

**Value**

A list.
Examples

```r
## Not run:
dataset_noendpoint(limit=3)

## End(Not run)
```

dataset_search  

Search for dataset metadata.

Description

Search for dataset metadata.

Usage

```r
dataset_search(
  query = NULL,
  type = NULL,
  publishingCountry = NULL,
  subtype = NULL,
  license = NULL,
  keyword = NULL,
  publishingOrg = NULL,
  hostingOrg = NULL,
  endorsingNodeKey = NULL,
  decade = NULL,
  projectId = NULL,
  hostingCountry = NULL,
  networkKey = NULL,
  doi = NULL
)
```

```r
dataset_search(
  query = NULL,
  type = NULL,
  publishingCountry = NULL,
  subtype = NULL,
  license = NULL,
  keyword = NULL,
  publishingOrg = NULL,
  hostingOrg = NULL,
  endorsingNodeKey = NULL,
  decade = NULL,
  projectId = NULL,
  hostingCountry = NULL,
  networkKey = NULL,
  doi = NULL,
)```
dataset_search

```r
facet = NULL,
facetLimit = NULL,
facetOffset = NULL,
facetMincount = NULL,
facetMultiselect = NULL,
limit = 100,
start = NULL,
description = FALSE,
curlopts = list()
)
```

dataset_suggest(
query = NULL,
type = NULL,
publishingCountry = NULL,
subtype = NULL,
license = NULL,
keyword = NULL,
publishingOrg = NULL,
hostingOrg = NULL,
endorsingNodeKey = NULL,
dercade = NULL,
projectId = NULL,
hostingCountry = NULL,
networkKey = NULL,
doi = NULL,
limit = 100,
start = NULL,
description = FALSE,
curlopts = list()
)

Arguments

query Simple full text search parameter. The value for this parameter can be a simple word or a phrase. Wildcards are not supported.
type The primary type of the dataset. Available values: "OCCURRENCE", "CHECK-LIST", "METADATA", "SAMPLING_EVENT", "MATERIAL_ENTITY".
publishingCountry Filters datasets by their owning organization’s country given as a ISO 639-1 (2 letter) country code.
subtype The sub-type of the dataset. Available values: "TAXONOMIC_AUTHORITY", "NOMENCLATOR_AUTHORITY", "INVENTORY_THEMATIC", "INVENTORY_REGIONAL", "GLOBAL_SPECIES_DATASET", "DERIVED_FROM_OCCURRENCE", "SPECIMEN", "OBSERVATION".
license The dataset’s licence. Available values: "CC0_1_0", "CC_BY_4_0", "CC_BY_NC_4_0", "UNSPECIFIED", "UNSUPPORTED".
**dataset_search()** searches and returns metadata on GBIF datasets from the registry. This function does not search occurrence data, only metadata on the datasets that contain or may contain occurrence data. It also searches over other dataset types, such as checklist and metadata datasets. Only a sample of results is returned.

**dataset_export()** function will download a tibble of the results of a dataset_search(). This function is primarily useful if you want the full results of a dataset_search().

Use `dataset_search(facet="x",limit=0)$facets` to get simple group by counts for different parameters.

- **keyword** Filters datasets by a case insensitive plain text keyword. The search is done on the merged collection of tags, the dataset keywordCollections and temporalCoverages.
- **publishingOrg** Filters datasets by their publishing organization UUID key.
- **hostingOrg** Filters datasets by their hosting organization UUID key.
- **endorseNodeKey** Node UUID key that endorsed this dataset’s publisher.
- **decade** Filters datasets by their temporal coverage broken down to decades. Decades are given as a full year, e.g. 1880, 1960, 2000, etc., and will return datasets wholly contained in the decade as well as those that cover the entire decade or more. Ranges can be used like this “1800,1900”.
- **projectId** Filter or facet based on the project ID of a given dataset. A dataset can have a project id if it is the result of a project. Multiple datasets can have the same project id.
- **hostingCountry** Filters datasets by their hosting organization’s country given as a ISO 639-1 (2 letter) country code.
- **networkKey** Filters network UUID associated to a dataset.
- **doi** DOI of the dataset.
- **facet** A facet name used to retrieve the most frequent values for a field.
- **facetLimit** Facet parameters allow paging requests using the parameters facetOffset and facetLimit.
- **facetOffset** Facet parameters allow paging requests using the parameters facetOffset and facetLimit.
- **facetMincount** Used in combination with the facet parameter.
- **facetMultiselect** Used in combination with the facet parameter.
- **limit** Controls the number of results in the page. Using too high a value will be overwritten with the default maximum threshold, depending on the service. Sensible defaults are used so this may be omitted.
- **start** Determines the offset for the search results. A limit of 20 and offset of 40 will get the third page of 20 results. Some services have a maximum offset.
- **description** Logical whether to return descriptions.
- **curlOptions** options passed on to `curl::HttpClient`.

**Details**

dataset_search() searches and returns metadata on GBIF datasets from the registry. This function does not search occurrence data, only metadata on the datasets that contain or may contain occurrence data. It also searches over other dataset types, such as checklist and metadata datasets. Only a sample of results is returned.

dataset_export() function will download a tibble of the results of a dataset_search(). This function is primarily useful if you want the full results of a dataset_search().

Use `dataset_search(facet="x",limit=0)$facets` to get simple group by counts for different parameters.
**dataset_search**

**Value**

A list for `dataset_search()`. A tibble for `dataset_export()`.

**References**


**Examples**

```r
## Not run:
# search metadata on all datasets and return a sample
dataset_search()
# dataset_export() # download info on all +90K datasets

dataset_search(publishingCountry = "US")
dataset_search(type = "OCCURRENCE")
dataset_search(keyword = "bird")
dataset_search(subtype = "TAXONOMIC_AUTHORITY")
dataset_search(license = "CC0_1_0")
dataset_search(query = "frog")
dataset_search(publishingCountry = "UA")
dataset_search(publishingOrg = "e2e717bf-551a-4917-bdc9-4fa0f342c530")
dataset_search(hostingOrg = "7ce8ae0-9e92-11dc-8738-b8a03c50a862")
dataset_search(decade="1890,1990",limit=5)
dataset_search(projectId = "GRIIS")
dataset_search(hostingCountry = "NO")
dataset_search(networkKey = "99d666b6c-9087-452f-a9d4-f15f2c2d0e7e")
dataset_search(doi="10.15468/aomfnb")

# search multiple values
dataset_search(projectId = "GRIIS;BID-AF2020-140-REG")
dataset_search(hostingCountry = "NO;SE")
dataset_search(doi="10.15468/aomfnb;10.15468/igasai")

# multiple filters
dataset_search(license = "CC0_1_0",subtype = "TAXONOMIC_AUTHORITY")
# dataset_export(license = "CC0_1_0",subtype = "TAXONOMIC_AUTHORITY")

# using dataset export to get all datasets
dataset_export(decade="1800,1900")
dataset_export(projectId="GRIIS")

# get simple group by counts
dataset_search(facet="type",limit=0,facetLimit=5)$facets
dataset_search(facet="publishingCountry",limit=0,facetLimit=5)$facets
dataset_search(facet="license",limit=0,facetLimit=5, facetMincount=10000)

## End(Not run)
dataset_uuid_funs  
Get dataset metadata using a datasetkey

Description
Get dataset metadata using a datasetkey

Usage

```r
dataset_get(uuid = NULL, curlopts = list())
dataset_process(uuid = NULL, limit = 20, start = NULL, curlopts = list())
dataset_networks(uuid = NULL, limit = 20, start = NULL, curlopts = list())
dataset_constituents(uuid = NULL, limit = 20, start = NULL, curlopts = list())
dataset_comment(uuid = NULL, curlopts = list())
dataset_contact(uuid = NULL, curlopts = list())
dataset_endpoint(uuid = NULL, curlopts = list())
dataset_identifier(uuid = NULL, curlopts = list())
dataset_machinetag(uuid = NULL, curlopts = list())
dataset_tag(uuid = NULL, curlopts = list())
dataset_metrics(uuid = NULL, curlopts = list())
```

Arguments

- `uuid`: A GBIF datasetkey uuid.
- `curlopts`: options passed on to `curl::HttpClient`.
- `limit`: Number of records to return.
- `start`: Record number to start at.

Details

`dataset_metrics()` can only be used with checklist type datasets.

Value

A tibble or a list.
derived_dataset

References

https://techdocs.gbif.org/en/openapi/v1/registry

Examples

```r
## Not run:
dataset_get("38b4c89f-584c-41bb-bd8f-cd1def33e92f")
dataset_process("38b4c89f-584c-41bb-bd8f-cd1def33e92f", limit=3)
dataset_networks("3dab037f-a520-4bc3-b888-508755c2eb52")
dataset_constituents("7ddf754f-d193-4cc9-b351-99906754a03b", limit=3)
dataset_comment("2e4cc37b-302e-4f1b-bbbb-1f674ff90e14")
dataset_contact("7ddf754f-d193-4cc9-b351-99906754a03b")
dataset_endpoint("7ddf754f-d193-4cc9-b351-99906754a03b")
dataset_identifier("7ddf754f-d193-4cc9-b351-99906754a03b")
dataset_machinetag("7ddf754f-d193-4cc9-b351-99906754a03b")
dataset_tag("c47f13c1-7427-45a0-9f12-237aad351040")
dataset_metrics("7ddf754f-d193-4cc9-b351-99906754a03b")

## End(Not run)
```

-----

**derived_dataset**

Register a derived dataset for citation.

**Description**

Register a derived dataset for citation.

**Usage**

```r
derived_dataset(
    citation_data = NULL,
    title = NULL,
    description = NULL,
    source_url = NULL,
    gbif_download_doi = NULL,
    user = NULL,
    pwd = NULL,
    curlopts = list()
)
```

```r
derived_dataset_prep(
    citation_data = NULL,
    title = NULL,
    description = NULL,
    source_url = NULL,
    gbif_download_doi = NULL,
    user = NULL,
    pwd = NULL,
```
curlopts = list()
)

Arguments

citation_data (required) A data.frame with two columns. The first column should be GBIF datasetkey uuids and the second column should be occurrence counts from each of your datasets, representing the contribution of each dataset to your final derived dataset.
title (required) The title for your derived dataset.
description (required) A description of the dataset. Perhaps describing how it was created.
source_url (required) A link to where the dataset is stored.
gbif_download_doi (optional) A DOI from an original GBIF download.
user (required) Your GBIF username.
pwd (required) Your GBIF password.
curlopts a list of arguments to pass to curl.

Value
A list.

Usage
Create a citable DOI for a dataset derived from GBIF mediated occurrences.

Use-case (1) your dataset was obtained with occ_search() and never returned a citable DOI, but you want to cite the data in a research paper.

Use-case (2) your dataset was obtained using occ_download() and you got a DOI, but the data underwent extensive filtering using CoordinateCleaner or some other cleaning pipeline. In this case be sure to fill in your original gbif_download_doi.

Use-case (3) your dataset was generated using a GBIF cloud export but you want a DOI to cite in your research paper.

Use derived_dataset to create a custom citable meta-data description and most importantly a DOI link between an external archive (e.g. Zenodo) and the datasets involved in your research or analysis.

All fields (except gbif_download_doi) are required for the registration to work.

We recommend that you run derived_dataset_prep() to check registration details before making it final with derived_dataset().

Authentication

Some rgbif functions require your GBIF credentials.

For the user and pwd parameters, you can set them in one of three ways:

1. Set them in your .Renviron/.bash_profile (or similar) file with the names GBIF_USER, GBIF_PWD, and GBIF_EMAIL
2. Set them in your .Rprofile file with the names gbif_user and gbif_pwd.

3. Simply pass strings to each of the parameters in the function call.

We strongly recommend the first option - storing your details as environment variables - as it’s the most widely used way to store secrets.

You can edit your .Renviron with usethis::edit_r_environ().

After editing, your .Renviron file should look something like this...

GBIF_USER="jwaller"
GBIF_PWD="fakepassword123"
GBIF_EMAIL="jwaller@gbif.org"

See ?Startup for help.

References

Examples

```r
## Not run:
data <- data.frame(
datasetKey = c("3ea36590-9b79-46a8-9300-c9ef0bfed7b8",
"630eb55d-5169-4473-99d6-a93396aeae38",
"806bf7d4-f762-11e1-a439-00145eb45e9a"),
count = c(3, 1, 2781)
)

## If output looks ok, run derived_dataset to register the dataset
derived_dataset_prep(
citation_data = data,
title = "Test for derived dataset",
description = "This data was filtered using a fake protocol",
source_url = "https://zenodo.org/record/4246090#.YPGS2OgzZPY"
)

# derived_dataset(
# citation_data = data,
# title = "Test for derived dataset",
# description = "This data was filtered using a fake protocol",
# source_url = "https://zenodo.org/record/4246090#.YPGS2OgzZPY"
# )

## Example with occ_search and dplyr
library(dplyr)

citation_data <- occ_search(taxonKey=212, limit=20)$data %>%

group_by(datasetKey) %>%
count()
```
You would still need to upload your data to Zenodo or something similar
derived_dataset_prep(
  citation_data = citation_data,
  title = "Bird data downloaded for test",
  description = "This data was downloaded using rgbif::occ_search and was later uploaded to Zenodo. ",
  source_url = "https://zenodo.org/record/4246090#.YPGS2OgzZPY",
  gbif_download_doi = NULL,
)

## End(Not run)

## Downloads interface

### Description

GBIF provides two ways to get occurrence data: through the /occurrence/search route (see occ_search()), or via the /occurrence/download route (many functions, see below). occ_search() is more appropriate for smaller data, while occ_download*() functions are more appropriate for larger data requests.

### Settings

You’ll use occ_download() to kick off a download. You’ll need to give that function settings from your GBIF profile: your user name, your password, and your email. These three settings are required to use the function. You can specify them in one of three ways:

- Pass them to occ_download as parameters
- Use R options: As options either in the current R session using the options() function, or by setting them in your .Rprofile file, after which point they’ll be read in automatically
- Use environment variables: As env vars either in the current R session using the Sys.setenv() function, or by setting them in your .Renviron/.bash_profile or similar files, after which point they’ll be read in automatically

**Beware**

You can not perform that many downloads, so plan wisely. See Rate limiting below.

### Rate limiting

If you try to launch too many downloads, you will receive an 420 “Enhance Your Calm” response. If there is less than 100 in total across all GBIF users, then you can have 3 running at a time. If there are more than that, then each user is limited to 1 only. These numbers are subject to change.
Functions

- **occ_download()** - Start a download
- **occ_download_prep()** - Prepare a download request
- **occ_download_queue()** - Start many downloads in a queue
- **occ_download_cached()** - Check for downloads already in your GBIF account
- **occ_download_wait()** - Re-run occ_download_meta() until ready
- **occ_download_meta()** - Get metadata progress on a single download
- **occ_download_list()** - List your downloads
- **occ_download_cancel()** - Cancel a download
- **occ_download_cancel_staged()** - Cancels any jobs with status RUNNING or PREPARING
- **occ_download_get()** - Retrieve a download
- **occ_download_import()** - Import a download from local file system
- **occ_download_datasets()** - List datasets for a download
- **occ_download_dataset_activity()** - Lists the downloads activity of a dataset

Download query composer methods:

See download_predicate_dsl

Query length

GBIF has a limit of 12,000 characters for a download query. This means that you can have a pretty long query, but at some point it may lead to an error on GBIF’s side and you’ll have to split your query into a few.

Download status

The following statuses can be found with any download:

- **PREPARING**: just submitted by user and awaiting processing (typically only a few seconds)
- **RUNNING**: being created (takes typically 1-15 minutes)
- **FAILED**: something unexpected went wrong
- **KILLED**: user decided to abort the job while it was in PREPARING or RUNNING phase
- **SUCCEEDED**: The download was created and the user was informed
- **FILE_ERASED**: The download was deleted according to the retention policy, see https://www.gbif.org/faq?question=for-how-long-will-does-gbif-store-downloads
download_predicate_dsl

_Description_

Download predicate DSL (domain specific language)

*Usage*

pred(key, value)
pred_gt(key, value)
pred_gte(key, value)
pred_lt(key, value)
pred_lte(key, value)
pred_not(...)
pred_like(key, value)
pred_within(value)
pred_isnull(key)
pred_notnull(key)
pred_or(..., .list = list())
pred_and(..., .list = list())
pred_in(key, value)
pred_default()

*Arguments*

key (character) the key for the predicate. See "Keys" below
value (various) the value for the predicate
..., .list For pred_or() or pred_and(), one or more objects of class occ_predicate, created by any pred* function
predicate methods and their equivalent types

pred* functions are named for the 'type' of operation they do, following the terminology used by GBIF; see https://www.gbif.org/developer/occurrence#predicates

Function names are given, with the equivalent GBIF type value (e.g., pred_gt and greaterThan)

The following functions take one key and one value:

- pred: equals
- pred_lt: lessThan
- pred_lte: lessThanOrEquals
- pred_gt: greaterThan
- pred_gte: greaterThanOrEquals
- pred_like: like

The following function is only for geospatial queries, and only accepts a WKT string:

- pred_within: within

The following function is only for stating the you don’t want a key to be null, so only accepts one key:

- pred_notnull: isNotNull

The following function is only for stating that you want a key to be null.

- pred_isnull: isNull

The following two functions accept multiple individual predicates, separating them by either "and" or "or":

- pred_and: and
- pred_or: or

The not predicate accepts one predicate; that is, this negates whatever predicate is passed in, e.g., not the taxonKey of 12345:

- pred_not: not

The following function is special in that it accepts a single key but many values; stating that you want to search for all the values:

- pred_in: in

The following function will apply commonly used defaults.

- pred_default

Using pred_default() is equivalent to running:

```python
defaults=
    pred_and(  
        pred("HAS_GEOSPATIAL_ISSUE",FALSE),  
        pred("HAS_COORDINATE",TRUE),  
        pred("OCCURRENCE_STATUS","PRESENT"),  
        pred_not(pred_in("BASIS_OF_RECORD",  
            c("FOSSIL_SPECIMEN","LIVING_SPECIMEN")))  
    )
```
What happens internally

Internally, the input to `pred*` functions turns into JSON to be sent to GBIF. For example ...

```
pred_in("taxonKey", c(2480946, 5229208)) gives:
```

```
{
  "type": "in",
  "key": "TAXON_KEY",
  "values": ["2480946", "5229208"]
}
```

```
pred_gt("elevation", 5000) gives:
```

```
{
  "type": "greaterThan",
  "key": "ELEVATION",
  "value": "5000"
}
```

```
pred_or(pred("taxonKey", 2977832), pred("taxonKey", 2977901)) gives:
```

```
{
  "type": "or",
  "predicates": [
    {
      "type": "equals",
      "key": "TAXON_KEY",
      "value": "2977832"
    },
    {
      "type": "equals",
      "key": "TAXON_KEY",
      "value": "2977901"
    }
  ]
}
```

Keys

Acceptable arguments to the `key` parameter are (with the version of the key in parens that must be sent if you pass the query via the body parameter; see below for examples). You can also use the 'ALL_CAPS' version of a key if you prefer. Open an issue in the GitHub repository for this package if you know of a key that should be supported that is not yet.

- `taxonKey` (TAXON_KEY)
- `acceptedTaxonKey` (ACCEPTED_TAXON_KEY)
- `kingdomKey` (KINGDOM_KEY)
- `phylumKey` (PHYLUM_KEY)
• classKey (CLASS_KEY)
• orderKey (ORDER_KEY)
• familyKey (FAMILY_KEY)
• genusKey (GENUS_KEY)
• subgenusKey (SUBGENUS_KEY)
• speciesKey (SPECIES_KEY)
• scientificName (SCIENTIFIC_NAME)
• country (COUNTRY)
• publishingCountry (PUBLISHING_COUNTRY)
• hasCoordinate (HAS_COORDINATE)
• hasGeospatialIssue (HAS_GEOSpatial_Issue)
• typeStatus (TYPE_STATUS)
• recordNumber (RECORD_NUMBER)
• lastInterpreted (LAST_INTERPRETED)
• modified (MODIFIED)
• continent (CONTINENT)
• geometry (GEOMETRY)
• basisOfRecord (BASIS_OF_RECORD)
• datasetKey (DATASET_KEY)
• datasetID/datasetId (DATASET_ID)
• eventDate (EVENT_DATE)
• catalogNumber (CATALOG_NUMBER)
• otherCatalogNumbers (OTHER_CATALOG_NUMBERS)
• year (YEAR)
• month (MONTH)
• decimalLatitude (DECIMAL_LATITUDE)
• decimalLongitude (DECIMAL_LONGITUDE)
• elevation (ELEVATION)
• depth (DEPTH)
• institutionCode (INSTITUTION_CODE)
• collectionCode (COLLECTION_CODE)
• issue (ISSUE)
• mediatype (MEDIA_TYPE)
• recordedBy (RECORDED_BY)
• recordedById/recordedByID (RECORDED_BY_ID)
• establishmentMeans (ESTABLishment_MEANS)
• coordinateUncertaintyInMeters (COORDINATE_UNCERTAINTY_IN_METERS)
• gadm (GADM_GID) (for the Database of Global Administrative Areas)
• level0Gid (GADM_LEVEL_0_GID)
• level1Gid (GADM_LEVEL_1_GID)
• level2Gid (GADM_LEVEL_2_GID)
• level3Gid (GADM_LEVEL_3_GID)
• stateProvince (STATE_PROVINCE)
• occurrenceStatus (OCCURRENCE_STATUS)
• publishingOrg (PUBLISHING_ORG)
• occurrenceId/occurrenceID (OCCURRENCE_ID)
• eventId/eventID (EVENT_ID)
• parentEventId/parentEventID (PARENT_EVENT_ID)
• identifiedBy (IDENTIFIED_BY)
• identifiedById/identifiedByID (IDENTIFIED_BY_ID)
• license (LICENSE)
• locality (LOCALITY)
• pathway (PATHWAY)
• preparations (PREPARATIONS)
• networkKey (NETWORK_KEY)
• organismId/organismID (ORGANISM_ID)
• organismQuantity (ORGANISM_QUANTITY)
• organismQuantityType (ORGANISM_QUANTITY_TYPE)
• protocol (PROTOCOL)
• relativeOrganismQuantity (RELATIVE_ORGANISM_QUANTITY)
• repatriated (REPATRIATED)
• sampleSizeUnit (SAMPLE_SIZE_UNIT)
• sampleSizeValue (SAMPLE_SIZE_VALUE)
• samplingProtocol (SAMPLING_PROTOCOL)
• verbatimScientificName (VERBATIM_SCIENTIFIC_NAME)
• taxonId/taxonId (TAXON_ID)
• taxonomicStatus (TAXONOMIC_STATUS)
• waterBody (WATER_BODY)
• iucnRedListCategory (IUCN_RED_LIST_CATEGORY)
• degreeOfEstablishment (DEGREE_OF_ESTABLISHMENT)
• isInCluster (IS_IN_CLUSTER)
• lifeStage (LIFE_STAGE)
• distanceFromCentroidInMeters (DISTANCE_FROM_CENTROID_IN_METERS)
elevation

Get elevation for lat/long points from a data.frame or list of points.

description

Uses the GeoNames web service

usage

elevation(
  input = NULL,
  latitude = NULL,
  longitude = NULL,
  latlong = NULL,
)
```r
elevation_model = "srtm3",
username = Sys.getenv("GEONAMES_USER"),
key,
curlopts,
...
)
```

**Arguments**

- **input**
  A data.frame of lat/long data. There must be columns decimalLatitude and decimalLongitude.

- **latitude**
  A vector of latitude’s. Must be the same length as the longitude vector.

- **longitude**
  A vector of longitude’s. Must be the same length as the latitude vector.

- **latlong**
  A vector of lat/long pairs. See examples.

- **elevation_model**
  (character) one of srtm3 (default), srtm1, astergdem, or gtopo30. See "Elevation models" below for more.

- **username**
  (character) Required. An GeoNames user name. See Details.

- **key, curlopts**
  defunct. see docs

- **...**
  curl options passed on to `curl::verb-GET` see `curl::curl_options()` for curl options

**Value**

A new column named `elevation_geonames` in the supplied data.frame or a vector with elevation of each location in meters. Note that data from GBIF can already have a column named `elevation`, thus the column we add is named differently.

**GeoNames user name**

To get a GeoNames user name, register for an account at [http://www.geonames.org/login](http://www.geonames.org/login) - then you can enable your account for the GeoNames webservice on your account page ([http://www.geonames.org/manageaccount](http://www.geonames.org/manageaccount)). Once you are enabled to use the webservice, you can pass in your username to the `username` parameter. Better yet, store your username in your `.Renviron` file, or similar (e.g., `.zshrc` or `.bash_profile` files) and read it in via `Sys.getenv()` as in the examples below. By default we do `Sys.getenv("GEONAMES_USER")` for the `username` parameter.

**Elevation models**

- **srtm3**:
  - sample area: ca 90m x 90m
  - result: a single number giving the elevation in meters according to srtm3, ocean areas have been masked as "no data" and have been assigned a value of -32768

- **srtm1**:
  - sample area: ca 30m x 30m
- result: a single number giving the elevation in meters according to srtm1, ocean areas have been masked as "no data" and have been assigned a value of -32768

- astergdem (Aster Global Digital Elevation Model V2 2011):
  - sample area: ca 30m x 30m, between 83N and 65S latitude
  - result: a single number giving the elevation in meters according to aster gdem, ocean areas have been masked as "no data" and have been assigned a value of -32768

- gtopo30:
  - sample area: ca 1km x 1km
  - result: a single number giving the elevation in meters according to gtopo30, ocean areas have been masked as "no data" and have been assigned a value of -9999

References

GeoNames http://www.geonames.org/export/web-services.html

Examples

```r
## Not run:
user <- Sys.getenv("GEONAMES_USER")
occ_key <- name_suggest('Puma concolor')$key[1]
dat <- occ_search(taxonKey = occ_key, limit = 300, hasCoordinate = TRUE)
head( elevation(dat$data, username = user) )

# Pass in a vector of lat's and a vector of long's
elevation(latitude = dat$data$decimalLatitude[1:10],
          longitude = dat$data$decimalLongitude[1:10],
          username = user, verbose = TRUE)

# Pass in lat/long pairs in a single vector
pairs <- list(c(31.8496,-110.576060), c(29.15503,-103.59828))
elevation(latlong=pairs, username = user)

# Pass on curl options
pairs <- list(c(31.8496,-110.576060), c(29.15503,-103.59828))
elevation(latlong=pairs, username = user, verbose = TRUE)

# different elevation models
lats <- dat$data$decimalLatitude[1:5]
lons <- dat$data$decimalLongitude[1:5]
elevation(latitude = lats, longitude = lons, elevation_model = "srtm3")
elevation(latitude = lats, longitude = lons, elevation_model = "srtm1")
elevation(latitude = lats, longitude = lons, elevation_model = "astergdem")
elevation(latitude = lats, longitude = lons, elevation_model = "gtopo30")

## End(Not run)
```
Description

Many parts of the GBIF API make use of enumerations, i.e. controlled vocabularies for specific topics - and are available via these functions

Usage

erenumeration(x = NULL, curlopts = list())

erenumeration_country(curlopts = list())

Arguments

- **x** A given enumeration.
- **curlopts** list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Value

enumeration returns a character vector, while enumeration_country returns a data.frame.

Examples

```r
## Not run:
# basic enumeration
enumeration()
enumeration("NameType")
enumeration("MetadataType")
enumeration("TypeStatus")

# country enumeration
enumeration_country()

# curl options
enumeration(curlopts = list(verbose=TRUE))

## End(Not run)
```
gbif_bbox2wkt

Convert a bounding box to a Well Known Text polygon, and a WKT to a bounding box

Description

Convert a bounding box to a Well Known Text polygon, and a WKT to a bounding box

Usage

gbif_bbox2wkt(minx = NA, miny = NA, maxx = NA, maxy = NA, bbox = NULL)

gbif_wkt2bbox(wkt = NULL)

Arguments

minx (numeric) Minimum x value, or the most western longitude
miny (numeric) Minimum y value, or the most southern latitude
maxx (numeric) Maximum x value, or the most eastern longitude
maxy (numeric) Maximum y value, or the most northern latitude
bbox (numeric) A vector of length 4, with the elements: minx, miny, maxx, maxy
wkt (character) A Well Known Text object.

Value

gbif_bbox2wkt returns an object of class character, a Well Known Text string of the form 'POLYGON((minx miny, maxx miny, maxx maxy, minx maxy, minx miny))'.

gbif_wkt2bbox returns a numeric vector of length 4, like c(minx, miny, maxx, maxy)

Examples

## Not run:
# Convert a bounding box to a WKT
## Pass in a vector of length 4 with all values
gbif_bbox2wkt(bbox=c(-125.0,38.4,-121.8,40.9))

## Or pass in each value separately
gbif_bbox2wkt(minx=-125.0, miny=38.4, maxx=-121.8, maxy=40.9)

# Convert a WKT object to a bounding box
wkt <- "POLYGON((-125 38.4,-125 40.9,-121.8 40.9,-121.8 38.4,-125 38.4))"

gbif_wkt2bbox(wkt)

## End(Not run)
gbif_citation

*Get citation for datasets used*

**Description**

Get citation for datasets used

**Usage**

gbif_citation(x)

**Arguments**

x (character) Result of call to `occ_search()`, `occ_data()`, `occ_download_get()`, `occ_download_meta()`, a dataset key, or occurrence key (character or numeric)

**Details**

Returns a set of citations, one for each dataset. We pull out unique dataset keys and get citations, so the length of citations may not be equal to the number of records you pass in.

Currently, this function gives back citations at the dataset level, not at the individual occurrence level. If occurrence keys are passed in, then we track down the dataset the key is from, and get the citation for the dataset.

**Value**

list with S3 class assigned, used by a print method to pretty print citation information. Though you can unclass the output or just index to the named items as needed.

**Examples**

```r
## Not run:

# character class inputs
## pass in a dataset key
gbif_citation(x='0ec3229f-2b53-484e-817a-de8ceb1f2e2b')
## pass in an occurrence key
# gbif_citation(x='1101144669')

# pass in an occurrence key as a numeric (won't work for a dataset key)
# gbif_citation(x=1101144669)

# Downloads
## occ_download_get()
# dl <- occ_download(pred("country", "BG"), pred_gte("year", 2020))
# occ_download_meta(dl) # wait until status = succeeded
# dl <- occ_download_get(dl, overwrite = TRUE)
# gbif_citation(dl)
```
## occ_download_meta()

```r
# key <- "0000122-171020152545675"
# res <- occ_download_meta(key)
# gbif_citation(res)
```

## End(Not run)

---

### gbif_geocode

*Geocode lat-lon point(s) with GBIF's set of geo-polygons (experimental)*

#### Description

Geocode lat-lon point(s) with GBIF’s set of geo-polygons (experimental)

#### Usage

```r
gbif_geocode(latitude = NULL, longitude = NULL)
```

#### Arguments

- `latitude`: a vector of numeric latitude values between -90 and 90.
- `longitude`: a vector of numeric longitude values between -180 and 180.

#### Value

A data.frame of results from the GBIF geocoding service.

- `latitude`: The input latitude
- `longitude`: The input longitude
- `index`: The original input rownumber
- `id`: The polygon id from which the geocode comes from
- `type`: One of the following: "Political" (county codes), "IHO" (marine regions), "SeaVox" (marine regions), "WGSRPD" (tdwg regions), "EEZ", (in national waters) or "GADM0","GADM1","GADM2","GADM3"
- `title`: The name of the source polygon
- `distance`: distance to the polygon boarder

This function uses the GBIF geocoder API which is not guaranteed to be stable and is undocumented. As such, this may return different data over time, may be rate-limited or may stop working if GBIF change the service. Use this function with caution.

#### References

Examples

```r
## Not run:
# one pair
gbif_geocode(0,0)
# or multiple pairs of points
gbif_geocode(c(0,50), c(0,20))

## End(Not run)
```

---

**gbif_issues**

*List all GBIF issues and their codes.*

---

**Description**

Returns a data.frame of all GBIF issues with the following columns:

- **code**: issue short code, e.g. `gass84`
- **code**: issue full name, e.g. `GEODETIC_DATUM_ASSUMED_WGS84`
- **description**: issue description
- **type**: issue type, either related to occurrence or name

**Usage**

```r
gbif_issues()
```

**Source**

https://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/NameUsageIssue.html

---

**gbif_issues_lookup**

*Lookup issue definitions and short codes*

---

**Description**

Lookup issue definitions and short codes

**Usage**

```r
gbif_issues_lookup(issue = NULL, code = NULL)
```
gbif_names

Arguments

issue Full name of issue, e.g. CONTINENT_COUNTRY_MISMATCH
code An issue short code, e.g. 'ccm'

Examples

gbif_issues_lookup(issue = 'CONTINENT_COUNTRY_MISMATCH')
gbif_issues_lookup(code = 'ccm')
gbif_issues_lookup(issue = 'COORDINATE_INVALID')
gbif_issues_lookup(code = 'cdiv')

gbif_names View highlighted terms in name results from GBIF.

Description

View highlighted terms in name results from GBIF.

Usage

gbif_names(input, output = NULL, browse = TRUE)

Arguments

input Input output from occ_search
output Output folder path. If not given uses temporary folder.
browse (logical) Browse output (default: TRUE)

Examples

## Not run:
# browse=FALSE returns path to file
gbif_names(name_lookup(query='snake', hl=TRUE), browse=FALSE)

(out <- name_lookup(query='canada', hl=TRUE, limit=5))
gbif_names(out)
gbif_names(name_lookup(query='snake', hl=TRUE))
gbif_names(name_lookup(query='bird', hl=TRUE))

# or not highlight
gbif_names(name_lookup(query='bird', limit=200))

## End(Not run)
Description

GBIF registry data via OAI-PMH

Usage

gbib_oai_identify(...)

gbib_oai_list_identifiers(
    prefix = "oai_dc",
    from = NULL,
    until = NULL,
    set = NULL,
    token = NULL,
    as = "df",
    ...
)

gbib_oai_list_records(
    prefix = "oai_dc",
    from = NULL,
    until = NULL,
    set = NULL,
    token = NULL,
    as = "df",
    ...
)

gbib_oai_list_metadataformats(id = NULL, ...)

gbib_oai_list_sets(token = NULL, as = "df", ...)

gbib_oai_get_records(ids, prefix = "oai_dc", as = "parsed", ...)

Arguments

... Curl options passed on to http::GET
prefix (character) A string to specify the metadata format in OAI-PMH requests issued to the repository. The default ("oai_dc") corresponds to the mandatory OAI unqualified Dublin Core metadata schema.
from (character) string giving datestamp to be used as lower bound for datestamp-based selective harvesting (i.e., only harvest records with datestamps in the given range). Dates and times must be encoded using ISO 8601. The trailing Z must be used when including time. OAI-PMH implies UTC for data/time specifications.
until (character) Datestamp to be used as an upper bound, for datestamp-based selective harvesting (i.e., only harvest records with datestamps in the given range).

set (character) A set to be used for selective harvesting (i.e., only harvest records in the given set).

token (character) A token previously provided by the server to resume a request where it last left off. 50 is max number of records returned. We will loop for you internally to get all the records you asked for.

as (character) What to return. One of "df" (for data.frame; default), "list" (get a list), or "raw" (raw text). For gbif_oai_get_records, one of "parsed" or "raw"

id, ids (character) The OAI-PMH identifier for the record. Optional.

Details

These functions only work with GBIF registry data, and do so via the OAI-PMH protocol (https://www.openarchives.org/OAI/openarchivesprotocol.html)

Value

raw text, list or data.frame, depending on requested output via as parameter

Examples

```r
## Not run:
gbif_oai_identify()

today <- format(Sys.Date(), "%Y-%m-%d")
gbif_oai_list_identifiers(from = today)
gbif_oai_list_identifiers(set = "country:NL")

gbif_oai_list_records(from = today)
gbif_oai_list_records(set = "country:NL")

gbif_oai_list_metadataformats()
gbif_oai_list_metadataformats(id = "9c4e36c1-d3f9-49ce-8ec1-8c434f9e6eb")

gbif_oai_list_sets()
gbif_oai_list_sets(as = "list")

gbif_oai_get_records("9c4e36c1-d3f9-49ce-8ec1-8c434f9e6eb")
ids <- c("9c4e36c1-d3f9-49ce-8ec1-8c434f9e6eb",
         "e0f1bb8a-2d81-4b2a-9194-d92848d3b82e")
gbif_oai_get_records(ids)

## End(Not run)
```
gbif_photos

View photos from GBIF.

Description

View photos from GBIF.

Usage

\[
gbif\_photos(\text{input, output = NULL, which = "table", browse = TRUE})
\]

Arguments

- **input**: Input output from occ_search
- **output**: Output folder path. If not given uses temporary folder.
- **which**: One of map or table (default).
- **browse**: (logical) Browse output (default: TRUE)

Details

The max number of photos you can see when which="map" is ~160, so cycle through if you have more than that.

BEWARE

The maps in the table view may not show up correctly if you are using RStudio

Examples

```r
## Not run:
res <- occ_search(mediaType = "StillImage", limit = 100)
gbif_photos(res)
gbif_photos(res, which='map')

res <- occ_search(scientificName = "Aves", mediaType = 'StillImage',
                limit=150)
gbif_photos(res)
gbif_photos(res, output = '~/barfoo')
```

## End(Not run)
installations

Installations metadata.

Description

Installations metadata.

Usage

installations(
  data = "all",
  uuid = NULL,
  query = NULL,
  identifier = NULL,
  identifierType = NULL,
  limit = 100,
  start = NULL,
  curlOpts = list()
)

Arguments

data The type of data to get. One or more of: 'contact', 'endpoint', 'dataset', 'comment', 'deleted', 'nonPublishing', or the special 'all'. Default: 'all'

uuid UUID of the data node provider. This must be specified if data is anything other than 'all'.

query Query nodes. Only used when data='all'. Ignored otherwise.

identifier The value for this parameter can be a simple string or integer, e.g. identifier=120. This parameter doesn't seem to work right now.

identifierType Used in combination with the identifier parameter to filter identifiers by identifier type. See details. This parameter doesn't seem to work right now.

limit Number of records to return. Default: 100. Maximum: 1000.

start Record number to start at. Default: 0. Use in combination with limit to page through results.

curlOpts list of named curl options passed on to HttpClient. See curl::curl_options for curl options

Details

identifierType options:

- DOI No description.
- FTP No description.
- GBIF_NODE Identifies the node (e.g: DK for Denmark, sp2000 for Species 2000).
• GBIF_PARTICIPANT Participant identifier from the GBIF IMS Filemaker system.
• GBIF_PORTAL Indicates the identifier originated from an auto_increment column in the portal.data_provider or portal.data_resource table respectively.
• HANDLER No description.
• LSID Reference controlled by a separate system, used for example by DOI.
• SOURCE_ID No description.
• UNKNOWN No description.
• URI No description.
• URL No description.
• UUID No description.

References

https://www.gbif.org/developer/registry#installations

Examples

```r
## Not run:
installations(limit=5)
installations(query="france", limit = 25)
installations(uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4")
installations(data='contact', uuid="2e029a0c-87af-42e6-87d7-f38a50b78201")
installations(data='endpoint', uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4")
installations(data='dataset', uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4")
installations(data='deleted', limit = 25)
installations(data='deleted', limit=2)
installations(data=c('deleted', 'nonPublishing'), limit=2)
installations(identifierType='DOI', limit=2)

# Pass on curl options
installations(data='deleted', curl_opts = list(verbos=TRUE))

## End(Not run)
```

lit_search

Search for literature that cites GBIF mediated data

Description

Search for literature that cites GBIF mediated data
Usage

```r
lit_search(
  q = NULL,
  countriesOfResearcher = NULL,
  countriesOfCoverage = NULL,
  literatureType = NULL,
  relevance = NULL,
  year = NULL,
  topics = NULL,
  datasetKey = NULL,
  publishingOrg = NULL,
  peerReview = NULL,
  openAccess = NULL,
  downloadKey = NULL,
  doi = NULL,
  journalSource = NULL,
  journalPublisher = NULL,
  flatten = TRUE,
  limit = NULL,
  curlopts = list()
)
```  

lit_count(...)

Arguments

- `q` (character) Simple full text search parameter. The value for this parameter can be a simple word or a phrase. Wildcards are not supported.
- `countriesOfResearcher` (character) Country of institution with which author is affiliated, e.g. DK (for Denmark). Country codes are listed in `enumeration_country()`.
- `countriesOfCoverage` (character) Country of focus of study, e.g. BR (for Brazil). Country codes are listed in `enumeration_country()`.
- `literatureType` (character) Type of literature ("JOURNAL", "BOOK_SECTION", "WORKING_PAPER", "REPORT", "GENERIC", "THESIS", "CONFERENCE_PROCEEDINGS", "WEB_PAGE").
- `relevance` (character) How is the publication relate to GBIF. See details ("GBIF_USED", "GBIF_MENTIONED", "GBIF_PUBLISHED", "GBIF_CITED", "GBIF_CITED", "GBIF_PUBLISHED", "GBIF_ACKNOWLEDGED", "GBIF_AUTHOR").
- `year` (integer) Year of publication.
- `topics` (character) Topic of publication.
- `datasetKey` (character) GBIF dataset uuid referenced in publication.
- `publishingOrg` (character) Publisher uuid whose dataset is referenced in publication.
- `peerReview` (logical) Has publication undergone peer-review?
openAccess (logical) Is publication Open Access?
downloadKey (character) Download referenced in publication.
doi (character) Digital Object Identifier (DOI).
journalSource (character) Journal of publication.
journalPublisher (character) Publisher of journal.
flatten (logical) should any lists in the resulting data be flattened into comma-seperated strings?
limit how many records to return. limit=NULL will fetch up to 10,000.
curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options.
... additional parameters passed to lit_search

Details

This function enables you to search for literature indexed by GBIF, including peer-reviewed papers, citing GBIF datasets and downloads. The literature API powers the literature search on GBIF.

The GBIF Secretariat maintains an ongoing literature tracking programme, which identifies research uses and citations of biodiversity information accessed through GBIF’s global infrastructure.

In the literature database, relevance refers to how publications relate to GBIF following these definitions:

- GBIF_USED : makes substantive use of data in a quantitative analysis (e.g. ecological niche modelling)
- GBIF_CITED : cites a qualitative fact derived in data (e.g. a given species is found in a given country)
- GBIF_DISCUSSED : discusses GBIF as an infrastructure or the use of data
- GBIF_PRIMARY : GBIF is the primary source of data (no longer applied)
- GBIF_ACKNOWLEDGED : acknowledges GBIF (but doesn’t use or cite data)
- GBIF_PUBLISHED : describes or talks about data published to GBIF
- GBIF_AUTHOR : authored by GBIF staff
- GBIF_MENTIONED : unspecifically mentions GBIF or the GBIF portal
- GBIF_FUNDED : funded by GBIF or a GBIF-managed funding programme

The following arguments can take multiple values:

- relevance
- countriesOfResearcher
- countriesOfCoverage
- literatureType
- topics
- datasetKey
map_fetch

- publishingOrg
- downloadKey
- doi
- journalSource
- journalPublisher

If flatten=TRUE, then data will be returned as flat data.frame with no complex column types (i.e. no lists or data.frames).

limit=NULL will return up to 10,000 records. The maximum value for limit is 10,000. If no filters are used, only the first 1,000 records will be returned, limit must be explicitly set to limit=10000, to get the first 10,000 records in this case.

lit_count() is a convenience wrapper, which will return the number of literature references for a certain lit_search() query. This is the same as running lit_search()$meta$count.

Value

A named list with two values: $data and $meta. $data is a data.frame of literature references.

Examples

```r
## Not run:
lit_search(q="bats")$data
lit_search(datasetKey="50c9509d-22c7-4a22-a47d-8c48425ef4a7")
lit_search(year=2020)
lit_search(year="2011,2020") # year ranges
lit_search(relevance=c("GBIF_CITED","GBIF_USED")) # multiple values
lit_search(relevance=c("GBIF_USED","GBIF_CITED"),
          topics=c("EVOLUTION","PHYLOGENETICS"))
lit_count() # total number of literature referencing GBIF
lit_count(peerReview=TRUE) # number of citations of iNaturalist
lit_count(datasetKey="50c9509d-22c7-4a22-a47d-8c48425ef4a7") # number of peer-reviewed articles used GBIF mediated data
lit_count(peerReview=TRUE,literatureType="JOURNAL",relevance="GBIF_USED")

# Typically what is meant by "literature that uses GBIF"
lit_search(peerReview=TRUE,literatureType="JOURNAL",relevance="GBIF_USED")
lit_count(peerReview=TRUE,literatureType="JOURNAL",relevance="GBIF_USED")

## End(Not run)
```
Description

This function is a wrapper for the GBIF mapping api version 2.0. The mapping API is a web map tile service making it straightforward to visualize GBIF content on interactive maps, and overlay content from other sources. It returns tile maps with number of GBIF records per area unit that can be used in a variety of ways, for example in interactive leaflet web maps. Map details are specified by a number of query parameters, some of them optional. Full documentation of the GBIF mapping api can be found at https://www.gbif.org/developer/maps

Usage

map_fetch(
  source = "density",
  x = 0:1,
  y = 0,
  z = 0,
  format = "@1x.png",
  srs = "EPSG:4326",
  bin = NULL,
  hexPerTile = NULL,
  squareSize = NULL,
  style = NULL,
  taxonKey = NULL,
  datasetKey = NULL,
  country = NULL,
  publishingOrg = NULL,
  publishingCountry = NULL,
  year = NULL,
  basisOfRecord = NULL,
  return = "png",
  base_style = NULL,
  plot_terra = TRUE,
  curl opts = list(http_version = 2),
  ...
)

Arguments

source (character) Either density for fast, precalculated tiles, or adhoc for any search. Default: density

x (integer sequence) the column. Default: 0:1

y (integer sequence) the row. Default: 0

z (integer) the zoom. Default: 0

format (character) The data format, one of:
  • @Hx.png for a 256px raster tile
  • @1x.png for a 512px raster tile (the default)
  • @2x.png for a 1024px raster tile
map_fetch

- @3x.png for a 2048px raster tile
- @4x.png for a 4096px raster tile

srs
(character) Spatial reference system. One of:
- EPSG:3857 (Web Mercator)
- EPSG:4326 (WGS84 plate care?)
- EPSG:3575 (Arctic LAEA on 10 degrees E)
- EPSG:3031 (Antarctic stereographic)

bin
(character) square or hex to aggregate occurrence counts into squares or hexagons. Points by default.

hexPerTile
(integer) sets the size of the hexagons (the number horizontally across a tile).

squareSize
(integer) sets the size of the squares. Choose a factor of 4096 so they tessalate correctly: probably from 8, 16, 32, 64, 128, 256, 512.

style
(character) for raster tiles, choose from the available styles. Defaults to classic.point for source="density" and "scaled.circle" for source="adhoc".

taxonKey
(integer/numeric/character) search by taxon key, can only supply 1.

datasetKey
(character) search by taxon key, can only supply 1.

country
(character) search by taxon key, can only supply 1.
publishingOrg
(character) search by taxon key, can only supply 1.
publishingCountry
(character) search by taxon key, can only supply 1.

year
(integer) integer that limits the search to a certain year or, if passing a vector of integers, multiple years, for example 1984 or c(2016, 2017, 2018) or 2010:2015 (years 2010 to 2015). optional

basisOfRecord
(character) one or more basis of record states to include records with that basis of record. The full list is: c("OBSERVATION", "HUMAN_OBSERVATION", "MACHINE_OBSERVATION", "MATERIAL_SAMPLE", "PRESERVED_SPECIMEN", "FOSSIL_SPECIMEN", "LIVING_SPECIMEN", "LITERATURE", "UNKNOWN").

return
(character) Either "png" or "terra".

base_style
(character) The style of the base map.

plot_terra
(logical) Set whether the terra map be default plotted.
curl_opts
options passed on to curl::HttpClient

... additional arguments passed to the adhoc interface.

Details

The default settings, return='png', will return a magick-image png. This image will be a composite image of the the occurrence tiles fetched and a base map. This map is primarily useful as a high quality image of occurrence records.

The args x and y can both be integer sequences. For example, x=0:3 or y=0:1. Note that the tile index starts at 0. Higher values of z will will produce more tiles that can be fetched and stitched together. Selecting a too high value for x or y will produce a blank image.

Setting return='terra' will return a terra::SpatRaster object. This is primarily useful if you were interested in the underlying aggregated occurrence density data.

See the article
map_fetch

Value

a magick-image or terra::SpatRaster object.

Author(s)

John Waller and Laurens Geffert <laurensgeffert@gmail.com>

References

https://www.gbif.org/developer/maps
https://api.gbif.org/v2/map/demo.html
https://api.gbif.org/v2/map/demo13.html

See Also

mvt_fetch()

Examples

## Not run:

# all occurrences
map_fetch()
# get artic map
map_fetch(srs='EPSG:3031')
# only preserved specimens
map_fetch(basisOfRecord="PRESERVEDSPECIMEN")

# Map of occ in Great Britain
map_fetch(z=3,y=1,x=7:8,country="GB")
# Penguins with artic projection
map_fetch(srs='EPSG:3031',taxonKey=2481660,style='glacier.point',
          base_style="gbif-dark")

# occ from a long time ago
map_fetch(year=1600)
# polygon style
map_fetch(style="iNaturalist.poly",bin="hex")
# iNaturalist dataset plotted
map_fetch(datasetKey="50c9509d-22c7-4a22-a47d-8c48425ef4a7",
          style="iNaturalist.poly")

# use source="adhoc" for more filters
map_fetch(z=1,
          source="adhoc",
          iucn_red_list_category="CR",
          style="scaled.circles",
          base_style="gbif-light")

# cropped map of Hawaii
map_fetch(z=5,x=3:4,y=12,source="adhoc",gadmGid="USA.12.1")
mvt_fetch

Fetch Map Vector Tiles (MVT)

Description
This function is a wrapper for the GBIF mapping api version 2.0. The mapping API is a web map tile service making it straightforward to visualize GBIF content on interactive maps, and overlay content from other sources. It returns maps vector tiles with number of GBIF records per area unit that can be used in a variety of ways, for example in interactive leaflet web maps. Map details are specified by a number of query parameters, some of them optional. Full documentation of the GBIF mapping api can be found at https://www.gbif.org/developer/maps

Usage
mvt_fetch(
  source = "density",
  x = 0,
  y = 0,
  z = 0,
  srs = "EPSG:4326",
  bin = NULL,
  hexPerTile = NULL,
  squareSize = NULL,
  style = "classic.point",
  taxonKey = NULL,
  datasetKey = NULL,
  country = NULL,
  publishingOrg = NULL,
  publishingCountry = NULL,
  year = NULL,
  basisOfRecord = NULL,
  ...
)

Arguments

  source  (character) Either density for fast, precalculated tiles, or adhoc for any search. Default: density
  x        (integer) the column. Default: 0
  y        (integer) the row. Default: 0
  z        (integer) the zoom. Default: 0
**mvt_fetch**

`srs` (character) Spatial reference system for the output (input srs for mvt from GBIF is always EPSG:3857). One of:
- EPSG:3857 (Web Mercator)
- EPSG:4326 (WGS84 plate care?)
- EPSG:3575 (Arctic LAEA on 10 degrees E)
- EPSG:3031 (Antarctic stereographic)

`bin` (character) square or hex to aggregate occurrence counts into squares or hexagons. Points by default. optional

`hexPerTile` (integer) sets the size of the hexagons (the number horizontally across a tile). optional

`squareSize` (integer) sets the size of the squares. Choose a factor of 4096 so they tessalate correctly: probably from 8, 16, 32, 64, 128, 256, 512. optional

`style` (character) for raster tiles, choose from the available styles. Defaults to classic.point. optional. THESE DON’T WORK YET.

`taxonKey` (integer/numeric/character) search by taxon key, can only supply 1. optional

`datasetKey` (character) search by taxon key, can only supply 1. optional

`country` (character) search by taxon key, can only supply 1. optional

`publishingOrg` (character) search by taxon key, can only supply 1. optional

`publishingCountry` (character) search by taxon key, can only supply 1. optional

`year` (integer) integer that limits the search to a certain year or, if passing a vector of integers, multiple years, for example 1984 or c(2016, 2017, 2018) or 2010:2015 (years 2010 to 2015). optional

`basisOfRecord` (character) one or more basis of record states to include records with that basis of record. The full list is: c("OBSERVATION", "HUMAN_OBSERVATION", "MACHINE_OBSERVATION", "MATERIAL_SAMPLE", "PRESERVED_SPECIMEN", "FOSSIL_SPECIMEN", "LIVING_SPECIMEN", "LITERATURE", "UNKNOWN"). optional

... curl options passed on to `crl::HttpClient`

**Details**

This function uses the arguments passed on to generate a query to the GBIF web map API. The API returns a web tile object as png that is read and converted into an R raster object. The break values or nbreaks generate a custom colour palette for the web tile, with each bin corresponding to one grey value. After retrieval, the raster is reclassified to the actual break values. This is a somewhat hacky but nonetheless functional solution in the absence of a GBIF raster API implementation.

We add extent and set the projection for the output. You can reproject after retrieving the output.

**Value**

an sf object

**References**

https://www.gbif.org/developer/maps
name_backbone

Lookup names in the GBIF backbone taxonomy.

Description

Lookup names in the GBIF backbone taxonomy.

See Also

map_fetch()

Examples

```r
## Not run:
if (
  requireNamespace("sf", quietly = TRUE) &&
  requireNamespace("protolite", quietly = TRUE)
) {
  x <- mvt_fetch(taxonKey = 2480498, year = 2007:2011)
  x

  # gives an sf object
  class(x)

  # different srs
  y <- mvt_fetch(taxonKey = 2480498, year = 2010, srs = "EPSG:3857")
  y
  z <- mvt_fetch(taxonKey = 2480498, year = 2010, srs = "EPSG:3031", verbose = TRUE)
  z
  z <- mvt_fetch(taxonKey = 2480498, year = 2010, srs = "EPSG:3575")
  z

  # bin
  x <- mvt_fetch(taxonKey = 212, year = 1998, bin = "hex",
              hexPerTile = 30, style = "classic-noborder.poly")
  x

  # query with basisOfRecord
  mvt_fetch(taxonKey = 2480498, year = 2010,
            basisOfRecord = "HUMAN_OBSERVATION")
  mvt_fetch(taxonKey = 2480498, year = 2010,
            basisOfRecord = c("HUMAN_OBSERVATION", "LIVING_SPECIMEN"))
}
## End(Not run)
```
name_backbone

Arguments

name (character) Full scientific name potentially with authorship (required)
rank (character) The rank given as our rank enum. (optional)
kingdom (character) If provided default matching will also try to match against this if no
direct match is found for the name alone. (optional)
phylum (character) If provided default matching will also try to match against this if no
direct match is found for the name alone. (optional)
class (character) If provided default matching will also try to match against this if no
direct match is found for the name alone. (optional)
order (character) If provided default matching will also try to match against this if no
direct match is found for the name alone. (optional)
family (character) If provided default matching will also try to match against this if no
direct match is found for the name alone. (optional)
name_backbone

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genus</td>
<td>(character) If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)</td>
</tr>
<tr>
<td>strict</td>
<td>(logical) If TRUE it (fuzzy) matches only the given name, but never a taxon in the upper classification (optional)</td>
</tr>
<tr>
<td>verbose</td>
<td>(logical) should the function give back more (less reliable) results. See function name_backbone_verbose()</td>
</tr>
<tr>
<td>start</td>
<td>Record number to start at. Default: 0. Use in combination with limit to page through results.</td>
</tr>
<tr>
<td>limit</td>
<td>Number of records to return. Default: 100. Maximum: 1000.</td>
</tr>
<tr>
<td>curlopts</td>
<td>list of named curl options passed on to HttpClient. see curl::curl_options for curl options</td>
</tr>
</tbody>
</table>

Details

If you don’t get a match, GBIF gives back a data.frame with columns synonym, confidence, and matchType='NONE'.

Value

For name_backbone, a data.frame for a single taxon with many columns. For name_backbone_verbose, a larger number of results in a data.frame the results of resulting from fuzzy matching. You will also get back your input name, rank, kingdom, phylum ect. as columns input_name, input_rank, input_kingdom ect. so you can check the results.

References

https://www.gbif.org/developer/species#searching

Examples

```r
## Not run:
name_backbone(name='Helianthus annuus', kingdom='plants')
name_backbone(name='Helianthus', rank='genus', kingdom='plants')
name_backbone(name='Poa', rank='genus', family='Poaceae')

# Verbose - gives back alternatives
# Strictness
name_backbone_verbose(name='Poa', kingdom='plants',
                        strict=FALSE)
name_backbone_verbose(name='Helianthus annuus', kingdom='plants',
                        strict=TRUE)

# Non-existent name - returns list of length 3 stating no match
name_backbone(name='Aso')
name_backbone(name='Oenante')

# Pass on curl options
name_backbone(name='Oenante', curlopts = list(verbise=TRUE))
```

## End(Not run)
name_backbone_checklist

Lookup names in the GBIF backbone taxonomy in a checklist.

Description

Lookup names in the GBIF backbone taxonomy in a checklist.

Usage

name_backbone_checklist(
  name_data = NULL,
  rank = NULL,
  kingdom = NULL,
  phylum = NULL,
  class = NULL,
  order = NULL,
  family = NULL,
  genus = NULL,
  strict = FALSE,
  verbose = FALSE,
  curlopts = list()
)

Arguments

name_data (data.frame or vector) see details.
rank (character) default value (optional).
knight = NULL, (character) default value (optional).
phylum = NULL, (character) default value (optional).
class = NULL, (character) default value (optional).
order = NULL, (character) default value (optional).
family = NULL, (character) default value (optional).
genus = NULL, (character) default value (optional).
strict = TRUE will not attempt to fuzzy match or return higherrankmatches.
verbose = FALSE, (logical) If true it shows alternative matches which were considered but then rejected.
curlopts list of named curl options passed on to HttpClient, see curl::curl_options for curl options
Details

This function is an alternative for name_backbone(), which will work with a list of names (a vector or a data.frame). The data.frame should have the following column names, but only the 'name' column is required. If only one column is present, then that column is assumed to be the 'name' column.

- **name**: (required)
- **rank**: (optional)
- **kingdom**: (optional)
- **phylum**: (optional)
- **class**: (optional)
- **order**: (optional)
- **family**: (optional)
- **genus**: (optional)

The input columns will be returned as "verbatim_name", "verbatim_rank", "verbatim_phylum" ect. A column of "verbatim_index" will also be returned giving the index of the input.

The following aliases for the 'name' column will work (any case or with '_' will work):

- "scientificName", "ScientificName", "scientific_name" ...
- "sci_name", "sciname", "SCI_NAME" ...
- "names", "NAMES" ...
- "species", "SPECIES" ...
- "species_name", "speciesname" ...
- "sp_name", "SP_NAME", "spname" ...
- "taxon_name", "taxonname", "TAXON NAME" ...

If more than one aliases is present and no column is named 'name', then the left-most column with an acceptable aliased name above is used.

If **verbose=TRUE**, a column called **is_alternative** will be returned, which species if a name was originally a first choice or not. **is_alternative=TRUE** means the name was not is not considered to be the best match by GBIF.

Default values for rank, kingdom, phylum, class, order, family, and genus can can be supplied. If a default value is supplied, the values for these fields are ignored in name_data, and the default value is used instead. This is most useful if you have a list of names and you know they are all plants, insects, birds, ect. You can also input multiple values, if they are the same length as list of names you are trying to match.

This function can also be used with a character vector of names. In that case no column names are needed of course.

This function is very similar to the GBIF species-lookup tool. [https://www.gbif.org/tools/species-lookup](https://www.gbif.org/tools/species-lookup).

If you have 1000s of names to match, it can take some minutes to get back all of the matches. I have tested it with 60K names. Scientific names with author details usually get better matches.

See also article Working With Taxonomic Names.
Value

A data.frame of matched names.

Examples

## Not run:
library(rgbif)

name_data <- data.frame(
  scientificName = c(  
    "Cirsium arvense (L.) Scop.", # a plant
    "Calopteryx splendens (Harris, 1780)", # an insect
    "Puma concolor (Linnaeus, 1771)", # a big cat
    "Ceylonosticta alwisi (Priyadarshana & Wijewardhane, 2016)", # newly discovered insect
    "Puma concoloar (Linnaeus, 1771)", # a mis-spelled big cat
    "Fake species (John Waller 2021)", # a fake species
    "Calopteryx" # Just a Genus
  ),
  description = c(  
    "a plant",
    "an insect",
    "a big cat",
    "newly discovered insect",
    "a mis-spelled big cat",
    "a fake species",
    "just a GENUS"
  ),
  kingdom = c(  
    "Plantae",
    "Animalia",
    "Animalia",
    "Animalia",
    "Animalia",
    "Johnlia",
    "Animalia"
  )
)

name_backbone_checklist(name_data)

# return more than 1 result per name
name_backbone_checklist(name_data, verbose=TRUE)

# works with just vectors too
name_list <- c(  
  "Cirsium arvense (L.) Scop.",
  "Calopteryx splendens (Harris, 1780)",
  "Puma concolor (Linnaeus, 1771)",
  "Ceylonosticta alwisi (Priyadarshana & Wijewardhane, 2016)",
  "Puma concuolor",
  "Fake species (John Waller 2021)",
  "Calopteryx")
name_issues

name_backbone_checklist(name_list)
name_backbone_checklist(name_list, verbose=TRUE)
name_backbone_checklist(name_list, strict=TRUE)

# default values
name_backbone_checklist(c("Aloe arborecens Mill.",
  "Cirsium arvense (L.) Scop."), kingdom="Plantae")
name_backbone_checklist(c("Aloe arborecens Mill.",
  "Calopteryx splendens (Harris, 1780)"), kingdom=c("Plantae", "Animalia"))

## End(Not run)

name_issues

Parse and examine further GBIF name issues on a dataset.

Description

Parse and examine further GBIF name issues on a dataset.

Usage

name_issues(.data, ..., mutate = NULL)

Arguments

.data         Output from a call to name_usage()
...           Named parameters to only get back (e.g. bbmn), or to remove (e.g. -bbmn).
mutate        (character) One of:

  • split  Split issues into new columns.
  • expand Expand issue abbreviated codes into descriptive names. for downloads datasets, this is not super useful since the issues come to you as expanded already.
  • split_expand  Split into new columns, and expand issue names.

For split and split_expand, values in cells become y ("yes") or n ("no")

References

https://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/NameUsageIssue.html
## name_lookup

Lookup names in all taxonomies in GBIF.

### Description

This service uses fuzzy lookup so that you can put in partial names and you should get back those things that match. See examples below.

Faceting: If facet=FALSE or left to the default (NULL), no faceting is done. And therefore, all parameters with facet in their name are ignored (facetOnly, facetMincount, facetMultiselect).

### Usage

```r
name_lookup(
    query = NULL,
    rank = NULL,
    higherTaxonKey = NULL,
    status = NULL,
    isExtinct = NULL,
)```

Examples

```r
## Not run:  
# what do issues mean, can print whole table  
head(gbif_issues())  
# or just name related issues  
gbif_issues()[which(gbif_issues()$type %in% c("name")),]  
# or search for matches  
gbif_issues()[gbif_issues()$code %in% c('bbmn','clasna','scina'),]  
# compare out data to after name_issues use  
(aa <- name_usage(name = "Lupus"))  
aa %>% name_issues("clasna")  

## or parse issues in various ways  
### remove data rows with certain issue classes  
aa %>% name_issues(-clasna, -scina)  
### expand issues to more descriptive names  
aa %>% name_issues(mutate = "expand")  
### split and expand  
aa %>% name_issues(mutate = "split_expand")  
### split, expand, and remove an issue class  
aa %>% name_issues(-bbmn, mutate = "split_expand")  
## Or you can use name_issues without %>%  
name_issues(aa, -bbmn, mutate = "split_expand")  
## End(Not run)  
```
name_lookup

habitat = NULL,
nameType = NULL,
datasetKey = NULL,
origin = NULL,
nomenclaturalStatus = NULL,
limit = 100,
start = 0,
facet = NULL,
facetMincount = NULL,
facetMultiselect = NULL,
type = NULL,
hl = NULL,
issue = NULL,
verbose = FALSE,
return = NULL,
curlopts = list()
)

Arguments
query Query term(s) for full text search.
rank CLASS, CULTIVAR, CULTIVAR_GROUP, DOMAIN, FAMILY, FORM, GENUS, INFORMAL, INFRAGENERIC_NAME, INFRAORDER, INFRASPECIFIC_NAME, INFRAUNITSPECIFIC_NAME, KINGDOM, ORDER, PHYLUM, SECTION, SERIES, SPECIES, STRAIN, SUBCLASS, SUBCLASS, SUBFAMILY, SUBFAMILY, SUBGENUS, SUBKINGDOM, SUBORDER, SUBPHYLUM, SUBSECTION, SUBSERIES, SUBSPECIES, SUBTRIBE, SUBVARIETY, SUPERCLASS, SUPERFAMILY, SUPERORDER, SUPERPHYLUM, SUPRAGENERIC_NAME, TRIBE, UNRANKED, VARIETY
higherTaxonKey Filters by any of the higher Linnean rank keys. Note this is within the respective checklist and not searching nub keys across all checklists. This parameter accepts many inputs in a vector (passed in the same request).
status Filters by the taxonomic status as one of:
  • ACCEPTED
  • DETERMINATION_SYNONYM Used for unknown child taxa referred to via spec, ssp, ...
  • DOUBTFUL Treated as accepted, but doubtful whether this is correct.
  • HETEROTYPIQUE_SYNONYM More specific subclass of SYNONYM.
  • HOMOTYPIC_SYNONYM More specific subclass of SYNONYM.
  • INTERMEDIATE_RANK_SYNONYM Used in nub only.
  • MISEPLPLIED More specific subclass of SYNONYM.
  • PROPARTE_SYNONYM More specific subclass of SYNONYM.
  • SYNONYM A general synonym, the exact type is unknown.

isExtinct (logical) Filters by extinction status (e.g. isExtinct=TRUE)
habitat (character) Filters by habitat. One of: marine, freshwater, or terrestrial
nameType Filters by the name type as one of:
• BLACKLISTED surely not a scientific name.
• CANDIDATUS Candidatus is a component of the taxonomic name for a bacterium that cannot be maintained in a Bacteriology Culture Collection.
• CULTIVAR a cultivated plant name.
• DOUBTFUL doubtful whether this is a scientific name at all.
• HYBRID a hybrid formula (not a hybrid name).
• INFORMAL a scientific name with some informal addition like "cf." or indetermined like Abies spec.
• SCINAME a scientific name which is not well formed.
• VIRUS a virus name.
• WELLFORMED a well formed scientific name according to present nomenclatural rules.

datasetKey Filters by the dataset’s key (a uuid)

origin (character) Filters by origin. One of:

• SOURCE
• DENORMED_CLASSIFICATION
• VERBATIM_ACCEPTED
• EX_AUTHOR_SYNONYM
• AUTONYM
• BASIONYM_PLACEHOLDER
• MISSING_ACCEPTED
• IMPLICIT_NAME
• PROPARTE
• VERBATIM_BASIONYM

nomenclaturalStatus Not yet implemented, but will eventually allow for filtering by a nomenclatural status enum.

limit Number of records to return. Hard maximum limit set by GBIF API: 99999.

start Record number to start at. Default: 0.

facet A vector/list of facet names used to retrieve the 100 most frequent values for a field. Allowed facets are: datasetKey, higherTaxonKey, rank, status, isExtinct, habitat, and nameType. Additionally threat and nomenclaturalStatus are legal values but not yet implemented, so data will not yet be returned for them.

facetMincount Used in combination with the facet parameter. Set facetMincount to exclude facets with a count less than x, e.g. http://bit.ly/2osAUQB only shows the type values 'CHECKLIST' and 'OCCURRENCE' because the other types have counts less than 10000

facetMultiselect (logical) Used in combination with the facet parameter. Set facetMultiselect=TRUE to still return counts for values that are not currently filtered, e.g. http://bit.ly/2JAYmaC still shows all type values even though type is being filtered by type=CHECKLIST.

type Type of name. One of occurrence, checklist, or metadata.
hl (logical) Set hl=TRUE to highlight terms matching the query when in fulltext search fields. The highlight will be an emphasis tag of class gbifH1 e.g. query='plant', hl=TRUE. Fulltext search fields include: title, keyword, country, publishing country, publishing organization title, hosting organization title, and description. One additional full text field is searched which includes information from metadata documents, but the text of this field is not returned in the response.

issue Filters by issue. Issue has to be related to names. Type gbif_issues() to get complete list of issues.

verbose (logical) If TRUE, all data is returned as a list for each element. If FALSE (default) a subset of the data that is thought to be most essential is organized into a data.frame.

return Defunct. All components are returned; index to the one(s) you want.

curlOpts list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Value

An object of class gbif, which is a S3 class list, with slots for metadata (meta), the data itself (data), the taxonomic hierarchy data (hierarchies), and vernacular names (names). In addition, the object has attributes listing the user supplied arguments and type of search, which is, differently from occurrence data, always equals to ‘single’ even if multiple values for some parameters are given. meta is a list of length four with offset, limit, endOfRecords and count fields. data is a tibble (aka data.frame) containing all information about the found taxa. hierarchies is a list of data.frame’s, one per GBIF key (taxon), containing its taxonomic classification. Each data.frame contains two columns: rankkey and name. names returns a list of data.frame’s, one per GBIF key (taxon), containing all vernacular names. Each data.frame contains two columns: vernacularName and language.

A list of length five:

• metadata
• data: either a data.frame (verbose=FALSE, default) or a list (verbose=TRUE).
• facets
• hierarchies
• names

Repeat parameter inputs

Some parameters can take many inputs, and treated as ‘OR’ (e.g., a or b or c). The following take many inputs:

• rank
• higherTaxonKey
• status
• habitat
• nameType
• datasetKey
• origin
References

https://www.gbif.org/developer/species#searching

Examples

```r
## Not run:
# Look up names like mammalia
name_lookup(query='mammalia', limit = 20)

# Start with an offset
name_lookup(query='mammalia', limit=1)
name_lookup(query='mammalia', limit=1, start=2)

# large requests (paging is internally implemented).
# hard maximum limit set by GBIF API: 99999
# name_lookup(query = "Carnivora", limit = 10000)

# Get all data and parse it, removing descriptions which can be quite long
out <- name_lookup('Helianthus annuus', rank="species", verbose=TRUE)
lapply(out$data, function(x) {
  x[!names(x) %in% c("descriptions","descriptionsSerialized")]
})

# Search for a genus
name_lookup(query="Cnaemidophorus", rank="genus")
# Limit records to certain number
name_lookup('Helianthus annuus', rank="species", limit=2)

# Query by habitat
name_lookup(habitat = "terrestrial", limit=2)
name_lookup(habitat = "marine", limit=2)
name_lookup(habitat = "freshwater", limit=2)

# Using faceting
name_lookup(facet='status', limit=0, facetMincount='70000')
name_lookup(facet=c('status','higherTaxonKey'), limit=0,
  facetMincount='700000')

name_lookup(facet='nameType', limit=0)
name_lookup(facet='habitat', limit=0)
name_lookup(facet='datasetKey', limit=0)
name_lookup(facet='rank', limit=0)
name_lookup(facet='isExtinct', limit=0)

name_lookup(isExtinct=TRUE, limit=0)

# text highlighting
## turn on highlighting
res <- name_lookup(query='canada', hl=TRUE, limit=5)
res$data
name_lookup(query='canada', hl=TRUE, limit=45)
## and you can pass the output to gbif_names() function
```
res <- name_lookup(query='canada', hl=TRUE, limit=5)
gbif_names(res)

# Lookup by datasetKey (set up sufficient high limit, API maximum: 99999)
# name_lookup(datasetKey='3f8a1297-3259-4700-91fc-acc4170b27ce',
#       # limit = 50000)

# Some parameters accept many inputs, treated as OR
name_lookup(rank = c("family", "genus"))
name_lookup(higherTaxonKey = c("119", "120", "121", "204"))
name_lookup(status = c("misapplied", "synonym"))$data
name_lookup(habitat = c("marine", "terrestrial"))
name_lookup(nameType = c("cultivar", "doubtful"))
name_lookup(datasetKey = c("73605f3a-af85-4ade-bbc5-522bfb90d847",
                        "d7c60346-44b6-400d-ba27-8d3fbeffc8a5"))
name_lookup(datasetKey = "289244ee-e1c1-49aa-b2d7-d379391ce265",
             origin = c("SOURCE", "DENORMED_CLASSIFICATION"))

# Pass on curl options
name_lookup(query='Cnaemidophorus', rank="genus",
             curlopts = list(\#verbose = TRUE))

## End(Not run)

---

**name_parse**

Parse taxon names using the GBIF name parser.

**Description**

Parse taxon names using the GBIF name parser.

**Usage**

```r
name_parse(scientificname, curlopts = list())
```

**Arguments**

- **scientificname**: A character vector of scientific names.
- **curlopts**: list of named curl options passed on to `HttpClient`. See `curl::curl_options` for curl options.

**Value**

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

**Author(s)**

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

Suggest up to 20 name usages.

Description

A quick and simple autocomplete service that returns up to 20 name usages by doing prefix matching against the scientific name. Results are ordered by relevance.

Usage

```r
name_suggest(
  q = NULL,
  datasetKey = NULL,
  rank = NULL,
  fields = NULL,
  start = NULL,
  limit = 100,
  curlopts = list()
)
```

Arguments

- **q** (character, required) Simple search parameter. The value for this parameter can be a simple word or a phrase. Wildcards can be added to the simple word parameters only, e.g. `q=puma`
- **datasetKey** (character) Filters by the checklist dataset key (a uuid, see examples)
name_suggest

rank (character) A taxonomic rank. One of class, cultivar, cultivar_group, domain, family, form, genus, informal, infrageneric_name, infraorder, infraspecific_name, infrasubspecific_name, kingdom, order, phylum, section, series, species, strain, subclass, subfamily, subgenus, superkingdom, suborder, subphylum, subsection, subspecies, subtribe, subvariety, superclass, superfamliy, superorder, superphylum, suprageneric_name, tribe, unranked, or variety.

fields (character) Fields to return in output data.frame (simply prunes columns off)

start Record number to start at. Default: 0. Use in combination with limit to page through results.

limit Number of records to return. Default: 100. Maximum: 1000.

curl opts list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Value

A list, with two elements data (tibble) and hierarchy (list of data.frame's). If 'higherClassificationMap' is one of the fields requested, then hierarchy is a list of data.frame's; if not included, hierarchy is an empty list.

Repeat parameter inputs

Some parameters can take many inputs, and treated as 'OR' (e.g., a or b or c). The following take many inputs:

• rank
• datasetKey

References

https://www.gbif.org/developer/species#searching

Examples

```r
## Not run:
name_suggest(q="Puma concolor")
name_suggest(q="Puma")
name_suggest(q="Puma", rank="genus")
name_suggest(q="Puma", rank="subspecies")
name_suggest(q="Puma", rank="species")
name_suggest(q="Puma", rank="infraspecific_name")

name_suggest(q="Puma", limit=2)
name_suggest(q="Puma", fields=c('key', 'canonicalName'))
name_suggest(q="Puma", fields=c('key', 'canonicalName', 'higherClassificationMap'))

# Some parameters accept many inputs, treated as OR
name_suggest(rank = c("family", "genus"))
name_suggest(datasetKey = c("73605f3a-af85-4ade-bbc5-522bf90d847", "d7c60346-44b6-400d-ba27-8d3fbeffe8a5"))
```
# If 'higherClassificationMap' in fields, a list is returned
ame_suggest(q='Puma', fields=c('key','higherClassificationMap'))

# Pass on curl options
name_suggest(q='Puma', limit=200, curlopts = list(verbos=TRUE))

## End(Not run)

---

**Usage**

lookup_details_for_specific_names_in_all_taxonomies_in_GBIF.

**Arguments**

- **key**: (numeric or character) A GBIF key for a taxon
- **name**: (character) Filters by a case insensitive, canonical namestring, e.g. 'Puma concolor'
- **data**: (character) Specify an option to select what data is returned. See Description below.
- **language**: (character) Language, default is english
- **datasetKey**: (character) Filters by the dataset’s key (a uuid). Must be length=1
- **uuid**: (character) A dataset key

**Description**

Lookup details for specific names in all taxonomies in GBIF.
name_usage

(rank) Taxonomic rank. Filters by taxonomic rank as one of: CLASS, CULTIVAR, CULTIVAR_GROUP, DOMAIN, FAMILY, FORM, GENUS, INFORMAL, INFRAGENERIC_NAME, INFRAORDER, INFRASPECIFIC_NAME, INFRASUBSPECIFIC_NAME, KINGDOM, ORDER, PHYLUM, SECTION, SERIES, SPECIES, STRAIN, SUBCLASS, SUBFAMILY, SUBFORM, SUBGENUS, SUBKINGDOM, SUBORDER, SUBPHYLUM, SUBSECTION, SUBSERIES, SUBSPECIES, SUBTRIBE, SUBVARIETY, SUPERCLASS, SUPERFAMILY, SUPEROORDER, SUPERPHYLUM, SUPRAGENERIC_NAME, TRIBE, UNRANKED, VARIETY

(shotname) A short name for a dataset - it may not do anything

(start) Record number to start at. Default: 0.

(limit) Number of records to return. Default: 100.

(return) Defunct. All components are returned; index to the one(s) you want

(curlopts) list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Details

This service uses fuzzy lookup so that you can put in partial names and you should get back those things that match. See examples below.

This function is different from name_lookup() in that that function searches for names. This function encompasses a bunch of API endpoints, most of which require that you already have a taxon key, but there is one endpoint that allows name searches (see examples below).

Note that data="verbatim" hasn’t been working.

Options for the data parameter are: 'all', 'verbatim', 'name', 'parents', 'children', 'related', 'synonyms', 'descriptions', 'distributions', 'media', 'references', 'speciesProfiles', 'vernacularNames', 'typeSpecimens', 'root', 'iucnRedListCategory'

This function used to be vectorized with respect to the data parameter, where you could pass in multiple values and the function internally loops over each option making separate requests. This has been removed. You can still loop over many options for the data parameter, just use an lapply family function, or a for loop, etc.

See name_issues() for more information about issues in issues column.

Value

An object of class gbif, which is a S3 class list, with slots for metadata (meta) and the data itself (data). In addition, the object has attributes listing the user supplied arguments and type of search, which is, differently from occurrence data, always equals to 'single' even if multiple values for some parameters are given. meta is a list of length four with offset, limit, endOfRecords and count fields. data is a tibble (aka data.frame) containing all information about the found taxa.

Repeat parameter inputs

These parameters used to accept many inputs, but no longer do:

• rank
• name
• language
• datasetKey

References

https://www.gbif.org/developer/species#nameUsages

Examples

## Not run:
# A single name usage
name_usage(key=1)

# Name usage for a taxonomic name
name_usage(name='Puma', rank="GENUS")

# Name usage for all taxa in a dataset
# (set sufficient high limit, but less than 100000)
# name_usage(datasetKey = "9ff7d317-609b-4c08-bd86-3bc404b77c42",
# limit = 10000)
# All name usages
name_usage()

# References for a name usage
name_usage(key=2435099, data='references')

# Species profiles, descriptions
name_usage(key=3119195, data='speciesProfiles')
name_usage(key=3119195, data='descriptions')
name_usage(key=2435099, data='children')

# Vernacular names for a name usage
name_usage(key=3119195, data='vernacularNames')

# Limit number of results returned
name_usage(key=3119195, data='vernacularNames', limit=3)

# Search for names by dataset with datasetKey parameter
name_usage(datasetKey="d7dddbf4-2cf0-4f39-9b2a-bb099caae36c")

# Search for a particular language
name_usage(key=3119195, language="FRENCH", data='vernacularNames')

# get root usage with a uuid
name_usage(data = "root", uuid = "73605f3a-af85-4ade-bbc5-522bfb90d847")

# search by language
name_usage(language = "spanish")

# Pass on curl options
name_usage(name='Puma concolor', limit=300, curlopts = list(verbos=TRUE))
# look up iucn red list category
name_usage(key = 7707728, data = 'iucnRedListCategory')

## End(Not run)

network

Get data about GBIF networks

**Description**

Get data about GBIF networks

**Usage**

```r
network(
  data = "all",
  uuid = NULL,
  query = NULL,
  identifier = NULL,
  identifierType = NULL,
  limit = 100,
  start = NULL,
  curlopts = list()
)
```

network_constituents(uuid = NULL, limit = 100, start = 0)

**Arguments**

data The type of data to get. One or more of: 'contact', 'endpoint', 'identifier', 'tag', 'machineTag', 'comment', 'constituents', or the special 'all'. Default: 'all'

uuid UUID of the data network provider. This must be specified if data is anything other than 'all'. Only 1 can be passed in

query Query nodes. Only used when data='all'. Ignored otherwise.

identifier The value for this parameter can be a simple string or integer, e.g. identifier=120. This parameter doesn’t seem to work right now.

identifierType Used in combination with the identifier parameter to filter identifiers by identifier type. See details. This parameter doesn’t seem to work right now.

limit Number of records to return. Default: 100. Maximum: 1000.

start Record number to start at. Default: 0. Use in combination with limit to page through results.

curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options
Details

identifierType options:

- DOI No description.
- FTP No description.
- GBIF_NODE Identifies the node (e.g: DK for Denmark, sp2000 for Species 2000).
- GBIF_PARTICIPANT Participant identifier from the GBIF IMS Filemaker system.
- GBIF_PORTAL Indicates the identifier originated from an auto_increment column in the portal.data_provider or portal.data_resource table respectively.
- HANDLER No description.
- LSID Reference controlled by a separate system, used for example by DOI.
- SOURCE_ID No description.
- UNKNOWN No description.
- URI No description.
- URL No description.
- UUID No description.

Get various information about GBIF networks. `network_constituents()` is a convenience function that allows you to get all the datasets in a network.

Value

- `network()` returns a list
- `network_constituents()` returns a data.frame of datasets in the network

References

https://www.gbif.org/developer/registry#networks

Examples

```r
## Not run:
network()
network(uuid='2b7c7b4f-4d4f-40d3-94de-c28b6fa054a6')
network_constituents('2b7c7b4f-4d4f-40d3-94de-c28b6fa054a6')

# curl options
network(curlopts = list(verbos=TRUE))
```

## End(Not run)
networks

Networks metadata.

Description

Networks metadata.

Usage

```r
networks(
  data = "all",
  uuid = NULL,
  query = NULL,
  identifier = NULL,
  identifierType = NULL,
  limit = 100,
  start = NULL,
  curlopts = list()
)
```

Arguments

data The type of data to get. One or more of: 'contact', 'endpoint', 'identifier', 'tag', 'machineTag', 'comment', 'constituents', or the special 'all'. Default: 'all'

uuid UUID of the data network provider. This must be specified if data is anything other than 'all'. Only 1 can be passed in

query Query nodes. Only used when data='all'. Ignored otherwise.

identifier The value for this parameter can be a simple string or integer, e.g. identifier=120. This parameter doesn't seem to work right now.

identifierType Used in combination with the identifier parameter to filter identifiers by identifier type. See details. This parameter doesn't seem to work right now.

limit Number of records to return. Default: 100. Maximum: 1000.

start Record number to start at. Default: 0. Use in combination with limit to page through results.

curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Details

identifierType options:

- DOI No description.
- FTP No description.
- GBIF_NODE Identifies the node (e.g: DK for Denmark, sp2000 for Species 2000).
nodes

- GBIF_PARTICIPANT Participant identifier from the GBIF IMS Filemaker system.
- GBIF_PORTAL Indicates the identifier originated from an auto_increment column in the portal.data_provider or portal.data_resource table respectively.
- HANDLER No description.
- LSID Reference controlled by a separate system, used for example by DOI.
- SOURCE_ID No description.
- UNKNOWN No description.
- URI No description.
- URL No description.
- UUID No description.

References

https://www.gbif.org/developer/registry#networks

Examples

```r
## Not run:
networks()
networks(uuid='2b7c7b4f-4d4f-40d3-94de-c28b6fa054a6')

# curl options
networks(curlopts = list(verbose=TRUE))

## End(Not run)
```

## Description

Nodes metadata.

## Usage

```r
nodes(
data = "all",
uuid = NULL,
query = NULL,
identifier = NULL,
identifierType = NULL,
limit = 100,
start = NULL,
isocode = NULL,
curlopts = list()
)
```
**Arguments**

- **data**
  The type of data to get. One or more of: 'organization', 'endpoint', 'identifier', 'tag', 'machineTag', 'comment', 'pendingEndorsement', 'country', 'dataset', 'installation', or the special 'all'. Default: 'all'

- **uuid**
  UUID of the data node provider. This must be specified if data is anything other than 'all'.

- **query**
  Query nodes. Only used when data='all'

- **identifier**
  The value for this parameter can be a simple string or integer, e.g. identifier=120. This parameter doesn’t seem to work right now.

- **identifierType**
  Used in combination with the identifier parameter to filter identifiers by identifier type. See details. This parameter doesn’t seem to work right now.

- **limit**
  Number of records to return. Default: 100. Maximum: 1000.

- **start**
  Record number to start at. Default: 0. Use in combination with limit to page through results.

- **isocode**
  A 2 letter country code. Only used if data='country'.

- **curlopts**
  list of named curl options passed on to HttpClient. see curl::curl_options for curl options

**Details**

**identifierType options:**

- DOI No description.
- FTP No description.
- GBIF_NODE Identifies the node (e.g: DK for Denmark, sp2000 for Species 2000).
- GBIF_PARTICIPANT Participant identifier from the GBIF IMS Filemaker system.
- GBIF_PORTAL Indicates the identifier originated from an auto_increment column in the portal.data_provider or portal.data_resource table respectively.
- HANDLER No description.
- LSID Reference controlled by a separate system, used for example by DOI.
- SOURCE_ID No description.
- UNKNOWN No description.
- URI No description.
- URL No description.
- UUID No description.

**References**

https://www.gbif.org/developer/registry#nodes
Examples

```r
## Not run:
nodes(limit=5)
nodes(uuid="1193638d-32d1-43f0-a855-8727c94299d8")
nodes(data='identifier', uuid="03e816b3-8f58-49ae-bc12-4e18b358d6d9")
nodes(data=c('identifier','organization','comment'),
    uuid="03e816b3-8f58-49ae-bc12-4e18b358d6d9")

uuids = c("8cb55387-7802-40e8-86d6-d357a583c596",
    "02c40d2a-1cbca-4333-90b7-e36e5e97aba8",
    "7a17efec-0a6a-424c-b743-f715852c3c1f",
    "b797ce0f-47e6-4231-b048-6b62ca3b0f55",
    "1193638d-32d1-43f0-a855-8727c94299d8",
    "d3499f89-5bc0-4454-8cdb-60bead228a6d",
    "c9736d-5ff7-4ece-9959-3c744360cdb3",
    "a8b16421-d80b-4ef3-8f22-098b01a89255",
    "8df8d012-8e64-4c88-886e-5213abf4623",
    "b35cf8f1-748d-467a-adca-4f9170f20a4e",
    "03e816b3-8f58-49ae-bc12-4e18b358d6d9",
    "873d1223-70b1-4433-bb21-dd70afe3053b",
    "87df2eb9-5116-4922-9a8a-3e0912276a72",
    "b0df1486-c8a8-469b-84cc-cc5342f9242",
    "0909d601-bda2-42df-9e63-a6d51847ebce",
    "0e0181bf-9c78-4676-bdce-54765e661bb8",
    "109ae14-c252-4a85-96e2-f5f4d5d088f4",
    "169eb292-376b-4cc6-8e31-9c2c432de0ad",
    "7e789bc9-79fc-4e6b-a94e-89dfc45a1788",
    "1f94b3ca-9345-4d85-afe2-4bace93a0fe")

res <- lapply(uuids, function(x) nodes(x, data='identifier')$data)
res <- res[!sapply(res, NROW)==0]
res[1]

# Pass on curl options
nodes(limit=20, curlopts=list(verbosetrue))

## End(Not run)
```

**occ_count**

Get number of occurrence records.

Description

Get number of occurrence records.

Usage

```r
occ_count(..., occurrenceStatus = "PRESENT", curlopts = list())
```
Arguments

... parameters passed to occ_search().

occurrenceStatus
(character) Default is "PRESENT". Specify whether search should return "PRESENT" or "ABSENT" data.

curlopts (list) curl options.

Details

occ_count() is a short convenience wrapper for occ_search(limit=0)$meta$count.
The current version (since rgbif 3.7.6) of occ_count() uses a different GBIF API endpoint from previous versions. This change greatly improves the usability of occ_count(). Legacy parameters georeferenced, type, date, to, from are no longer supported and not guaranteed to work correctly.

Multiple values of the type c("a", "b") will give an error, but "a;b" will work.

Value

The occurrence count of the occ_search() query.

See Also

occ_count_year(), occ_count_country(), occ_count_pub_country(), occ_count_basis_of_record()

Examples

## Not run:
# total occurrences mediated by GBIF
occ_count() # should be > 2 billion!

# number of plant occurrences
occ_count(kingdomKey=name_backbone("Plantea")$usageKey)
occ_count(scientificName = 'Ursus americanus')

occ_count(country="DK") # found in Denmark
occ_count(country="DK;US") # found in Denmark and United States
occ_count(publishingCountry="US") # published by the United States
# number of repatriated eBird records in India
occ_count(repatriated = TRUE,country="IN")

occ_count(taxonKey=212) # number of bird occurrences
# between years 1800-1900
occ_count(basisOfRecord="PRESERVED_SPECIMEN", year="1800,1900")
occ_count(recordedBy="John Waller") # recorded by John Waller
occ_count(decimalLatitude=0, decimalLongitude=0) # exactly on 0,0

# close to a known iso2 centroid
occ_count(distanceFromCentroidInMeters="0,2000")
# close to a known iso2 centroid in Sweden
occ_count(distanceFromCentroidInMeters="0,2000",country="SE")
occ_count(hasCoordinate=TRUE) # with coordinates
occ_count(protocol = "DIGIR") # published using DIGIR format
occ_count(mediaType = 'StillImage') # with images

# number of occurrences iucn status "critically endangered"
occ_count(iucnRedListCategory="CR")
occ_count(verbatimScientificName="Calopteryx splendens;Calopteryx virgo")
occ_count(

# getting a table of counts using the facets interface
# occurrence counts by year
occ_count(facet="year")
occ_count(facet="year",facetLimit=400)

# top scientificNames from Japan
occ_count(facet="scientificName",country="JP")
# top countries publishing specimen bird records between 1850 and 1880
occ_count(facet="scientificName",taxonKey=212,basisOfRecord="PRESERVED_SPECIMEN",year="1850,1880")

# Number of present or absence records of Elephants
occ_count(facet="occurrenceStatus",scientificName="Elephantidae")

# top 100 datasets publishing occurrences to GBIF
occ_count(facet="datasetKey",facetLimit=100)
# top datasets publishing country centroids on GBIF
occ_count(facet="datasetKey",distanceFromCentroidInMeters="0")

# common values for coordinateUncertaintyInMeters for museum specimens
occ_count(facet="coordinateUncertaintyInMeters",basisOfRecord="PRESERVED_SPECIMEN")

# number of iucn listed bird and insect occurrences in Mexico
occ_count(facet="iucnRedListCategory",taxonKey="212;216",country="MX")

# most common latitude values mediated by GBIF
occ_count(facet="decimalLatitude")

# top iNaturalist users publishing research-grade obs to GBIF
occ_count(facet="recordedBy",datasetKey="50c9509d-22c7-4a22-a47d-8c48425ef4a7")
# top 100 iNaturalist users from Ukraine
occ_count(facet="recordedBy",datasetKey="50c9509d-22c7-4a22-a47d-8c48425ef4a7",country="UA",facetLimit=100)

# top institutions publishing specimen occurrences to GBIF
occ_count(facet="institutionCode",basisOfRecord="PRESERVED_SPECIMEN")

## End(Not run)
**Description**

Get quick pre-computed occurrence counts of a limited number of dimensions.

**Usage**

- `occ_count_country(publishingCountry = NULL)`
- `occ_count_pub_country(country = NULL)`
- `occ_count_year(year = NULL)`
- `occ_count_basis_of_record(curlopts = list())`

**Arguments**

- `publishingCountry` (The 2-letter country code (as per ISO-3166-1) the country from which the occurrence was published.)
- `country` (character) The 2-letter country code (ISO-3166-1) in which the occurrence was recorded.
- `year` (The 4 digit year. Supports range queries, `smaller,larger` (e.g., `'1990,1991'`, whereas `1991, 1990` wouldn’t work).)
- `curlopts` (list) curl options.

**Details**

Get quick pre-computed counts of a limited number of dimensions.

- `occ_count_country()` will return a data.frame with occurrence counts by country. By using `occ_count_country(publishingCountry="DK")` will return the occurrence contributions Denmark has made to each country.
- `occ_count_pub_country()` will return a data.frame with occurrence counts by publishing country. Using `occ_count_pub_country(country="DK")`, will return the occurrence contributions each country has made to that focal country=DK.
- `occ_count_year()` will return a data.frame with the total occurrences mediated by GBIF for each year. By using `occ_count_year(year="1800,1900")` will only return counts for that range.
- `occ_count_basis_of_record()` will return a data.frame with total occurrences mediated by GBIF for each basis of record.

**Value**

A data.frame of counts.
See Also

occ_count()

Examples

## Not run:
# total occurrence counts for all countries and iso2 places
occ_count_country()
# the occurrences Mexico has published in other countries
occ_count_country("MX")
# the occurrences Denmark has published in other countries
occ_count_country("DK")

# the occurrences other countries have published in Denmark
occ_count_pub_country("DK")
# the occurrences other countries have published in Mexico
occ_count_pub_country("MX")

# total occurrence counts for each year that an occurrence was
# recorded or collected.
occ_count_year()
# supports ranges
occ_count_year("1800,1900")

# table of occurrence counts by basis of record
occ_count_basis_of_record()

## End(Not run)

---

**occ_data**  

*Search for GBIF occurrences - simplified for speed*

**Description**

Search for GBIF occurrences - simplified for speed

**Usage**

```r
occ_data(
  taxonKey = NULL,
  scientificName = NULL,
  country = NULL,
  publishingCountry = NULL,
  hasCoordinate = NULL,
  typeStatus = NULL,
  recordNumber = NULL,
  lastInterpreted = NULL,
  continent = NULL,
)```
occ_data

geometry = NULL,
gemn_big = "asis",
gemn_size = 40,
gemn_n = 10,
recordedBy = NULL,
recordedByID = NULL,
identifiedByID = NULL,
basisOfRecord = NULL,
datasetKey = NULL,
eventDate = NULL,
catalogNumber = NULL,
year = NULL,
month = NULL,
decimalLatitude = NULL,
decimalLongitude = NULL,
elevation = NULL,
depth = NULL,
institutionCode = NULL,
collectionCode = NULL,
hasGeospatialIssue = NULL,
issue = NULL,
search = NULL,
mediaType = NULL,
subgenusKey = NULL,
repatriated = NULL,
phylumKey = NULL,
kingdomKey = NULL,
classKey = NULL,
orderKey = NULL,
familyKey = NULL,
genusKey = NULL,
speciesKey = NULL,
establishmentMeans = NULL,
degreeOEEstablishment = NULL,
protocol = NULL,
license = NULL,
organismId = NULL,
publishingOrg = NULL,
stateProvince = NULL,
waterBody = NULL,
locality = NULL,
limit = 500,
start = 0,
skip_validate = TRUE,
occurrenceStatus = "PRESENT",
gadmGid = NULL,
coordinateUncertaintyInMeters = NULL,
verbatimScientificName = NULL,
eventId = NULL,
identifiedBy = NULL,
networkKey = NULL,
verbatimTaxonId = NULL,
ocurrenceId = NULL,
organismQuantity = NULL,
organismQuantityType = NULL,
relativeOrganismQuantity = NULL,
iucnRedListCategory = NULL,
lifeStage = NULL,
isInCluster = NULL,
distanceFromCentroidInMeters = NULL,
curlopts = list(http_version = 2)
)

Arguments

taxonKey  (numeric) A taxon key from the GBIF backbone. All included and synonym taxa are included in the search, so a search for aves with taxonKey=212 will match all birds, no matter which species. You can pass many keys to occ_search(taxonKey=c(1,212)).

scientificName A scientific name from the GBIF backbone. All included and synonym taxa are included in the search.

country (character) The 2-letter country code (ISO-3166-1) in which the occurrence was recorded. enumeration_country().

publishingCountry The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded. See enumeration_country().

hasCoordinate (logical) Return only occurrence records with lat/long data (TRUE) or all records (FALSE, default).

typeStatus Type status of the specimen. One of many options.

recordNumber Number recorded by collector of the data, different from GBIF record number.

lastInterpreted Date the record was last modified in GBIF, in ISO 8601 format: yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Supports range queries, 'smaller,larger' (e.g., '1990,1991', whereas '1991,1990' wouldn't work).

continent The source supplied continent.

• "africa"
• "antarctica"
• "asia"
• "europe"
• "north_america"
• "oceania"
• "south_america"
Continent is not inferred but only populated if provided by the dataset publisher. Applying this filter may exclude many relevant records.

**geometry** (character) Searches for occurrences inside a polygon in Well Known Text (WKT) format. A WKT shape written as either

- "POINT"
- "LINESTRING"
- "LINEARRING"
- "POLYGON"
- "MULTIPOLYGON"

For Example, "POLYGON((37.08 46.86,38.06 46.86,38.06 47.28,37.08 47.28,37.0 46.8))". See also the section **WKT** below.

**geom_big** (character) One of "axe", "bbox", or "asis" (default).

**geom_size** (integer) An integer indicating size of the cell. Default: 40.

**geom_n** (integer) An integer indicating number of cells in each dimension. Default: 10.

**recordedBy** (character) The person who recorded the occurrence.

**recordedByID** (character) Identifier (e.g. ORCID) for the person who recorded the occurrence.

**identifiedByID** (character) Identifier (e.g. ORCID) for the person who provided the taxonomic identification of the occurrence.

**basisOfRecord** (character) The specific nature of the data record. See here.

- "FOSSIL_SPECIMEN"
- "HUMAN_OBSERVATION"
- "MATERIAL_CITATION"
- "MATERIAL_SAMPLE"
- "LIVING_SPECIMEN"
- "MACHINE_OBSERVATION"
- "OBSERVATION"
- "PRESERVED_SPECIMEN"
- "OCCURRENCE"

**datasetKey** (character) The occurrence dataset uuid key. That can be found in the dataset page url. For example, "7e380070-f762-11e1-a439-00145eb45e9a" is the key for [Natural History Museum (London) Collection Specimens](http://example.com).


**catalogNumber** (character) An identifier of any form assigned by the source within a physical collection or digital dataset for the record which may not unique, but should be fairly unique in combination with the institution and collection code.

**year** The 4 digit year. A year of 98 will be interpreted as AD 98. Supports range queries, ‘smaller,larger’ (e.g., ‘1990,1991’, whereas 1991, 1990’ wouldn’t work).

**month** The month of the year, starting with 1 for January. Supports range queries, ‘smaller,larger’ (e.g., ‘1,2’, whereas ‘2,1’ wouldn’t work).
decimalLatitude
Latitude in decimals between -90 and 90 based on WGS84. Supports range queries, 'smaller,larger' (e.g., '25,30', whereas '30,25' wouldn’t work).

decimalLongitude
Longitude in decimals between -180 and 180 based on WGS84. Supports range queries (e.g., '-0.4,-0.2', whereas '-0.2,-0.4' wouldn’t work).

elevation
Elevation in meters above sea level. Supports range queries, 'smaller,larger' (e.g., '5,30', whereas '30,5' wouldn’t work).

depth
Depth in meters relative to elevation. For example 10 meters below a lake surface with given elevation. Supports range queries, 'smaller,larger' (e.g., '5,30', whereas '30,5' wouldn’t work).

institutionCode
An identifier of any form assigned by the source to identify the institution the record belongs to.

collectionCode
(character) An identifier of any form assigned by the source to identify the physical collection or digital dataset uniquely within the text of an institution.

hasGeospatialIssue
(logical) Includes/excludes occurrence records which contain spatial issues (as determined in our record interpretation), i.e. hasGeospatialIssue=TRUE returns only those records with spatial issues while hasGeospatialIssue=FALSE includes only records without spatial issues. The absence of this parameter returns any record with or without spatial issues.

issue
(character) One or more of many possible issues with each occurrence record. Issues passed to this parameter filter results by the issue. One of many options. See here for definitions.

search
(character) Query terms. The value for this parameter can be a simple word or a phrase. For example, search="puma"

mediaType
(character) Media type of "MovingImage", "Sound", or "StillImage".

subgenusKey
(numeric) Subgenus classification key.

repatriated
(character) Searches for records whose publishing country is different to the country where the record was recorded in.

phylumKey
(numeric) Phylum classification key.

kingdomKey
(numeric) Kingdom classification key.

classKey
(numeric) Class classification key.

orderKey
(numeric) Order classification key.

familyKey
(numeric) Family classification key.

genusKey
(numeric) Genus classification key.

speciesKey
(numeric) Species classification key.

establishmentMeans
(character) provides information about whether an organism or organisms have been introduced to a given place and time through the direct or indirect activity of modern humans.

• "Introduced"
• "Native"
• "NativeReintroduced"
• "Vagrant"
• "Uncertain"
• "IntroducedAssistedColonisation"

degreeOfEstablishment
(character) Provides information about degree to which an Organism survives, reproduces, and expands its range at the given place and time. One of many options.

protocol
(character) Protocol or mechanism used to provide the occurrence record. One of many options.

license
(character) The type license applied to the dataset or record.
  • "CC0_1_0"
  • "CC_BY_4_0"
  • "CC_BY_NC_4_0"

organismId
(numeric) An identifier for the Organism instance (as opposed to a particular digital record of the Organism). May be a globally unique identifier or an identifier specific to the data set.

publishingOrg
(character) The publishing organization key (a UUID).

stateProvince
(character) The name of the next smaller administrative region than country (state, province, canton, department, region, etc.) in which the Location occurs.

waterBody
(character) The name of the water body in which the locations occur.

locality
(character) The specific description of the place.

limit
Number of records to return. Default: 500. Note that the per request maximum is 300, but since we set it at 500 for the function, we do two requests to get you the 500 records (if there are that many). Note that there is a hard maximum of 100,000, which is calculated as the limit+start, so start=99,000 and limit=2000 won’t work.

start
Record number to start at. Use in combination with limit to page through results. Note that we do the paging internally for you, but you can manually set the start parameter.

skip_validate
(logical) whether to skip wellknown::validate_wkt call or not. passed down to check_wkt(). Default: TRUE.

occurrenceStatus
(character) Default is "PRESENT". Specify whether search should return "PRESENT" or "ABSENT" data.

gadmGid
(character) The gadm id of the area occurrences are desired from. https://gadm.org/.

coordinateUncertaintyInMeters
A number or range between 0-1,000,000 which specifies the desired coordinate uncertainty. A coordinateUncertainty InMeters=1000 will be interpreted all records with exactly 1000m. Supports range queries, 'smaller,larger' (e.g., '1000,10000', whereas '10000,1000' wouldn’t work).

verbatimScientificName
(character) Scientific name as provided by the source.
eventId (character) identifier(s) for a sampling event.
identifiedBy (character) names of people, groups, or organizations.
networkKey (character) The occurrence network key (a uuid) who assigned the Taxon to the subject.
verbatimTaxonId (character) The taxon identifier provided to GBIF by the data publisher.
ocurrenceId (character) occurrence id from source.
organismQuantity A number or range which specifies the desired organism quantity. An organismQuantity=5 will be interpreted all records with exactly 5. Supports range queries, smaller,larger (e.g., '5,20', whereas '20,5' wouldn’t work).
organismQuantityType (character) The type of quantification system used for the quantity of organisms. For example, "individuals" or "biomass".
relativeOrganismQuantity (numeric) A relativeOrganismQuantity=0.1 will be interpreted all records with exactly 0.1 The relative measurement of the quantity of the organism (a number between 0-1). Supports range queries, "smaller,larger" (e.g., '0.1,0.5', whereas '0.5,0.1' wouldn’t work).
iucnRedListCategory (character) The IUCN threat status category.
  • "NE" (Not Evaluated)
  • "DD" (Data Deficient)
  • "LC" (Least Concern)
  • "NT" (Near Threatened)
  • "VU" (Vulnerable)
  • "EN" (Endangered)
  • "CR" (Critically Endangered)
  • "EX" (Extinct)
  • "EW" (Extinct in the Wild)
lifeStage (character) the life stage of the occurrence. One of many options.
isInCluster (logical) identify potentially related records on GBIF.
distanceFromCentroidInMeters A number or range. A value of "2000,*" means at least 2km from known centroids. A value of "0" would mean occurrences exactly on known centroids. A value of "0,2000" would mean within 2km of centroids. Max value is 5000.
curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Value

An object of class gbif_data, which is a S3 class list, with slots for metadata (meta) and the occurrence data itself (data), and with attributes listing the user supplied arguments and whether it was a "single" or "many" search; that is, if you supply two values of the datasetKey parameter to searches are done, and it’s a "many". meta is a list of length four with offset, limit, endOfRecords and count fields. data is a tibble (aka data.frame)
Multiple values passed to a parameter

There are some parameters you can pass multiple values to in a vector, each value of which produces a different request (multiple different requests = c("a","b")). Some parameters allow multiple values to be passed in the same request (multiple same request = "a;b") in a semicolon separated string (e.g., 'a;b'); if given we’ll do a single request with that parameter repeated for each value given (e.g., foo=a&foo=b if the parameter is foo).

See article Multiple Values.

Hierarchies

Hierarchies are returned with each occurrence object. There is no option to return them from the API. However, within the occ_search function you can select whether to return just hierarchies, just data, all of data and hierarchies and metadata, or just metadata. If all hierarchies are the same we just return one for you.

curl debugging

You can pass parameters not defined in this function into the call to the GBIF API to control things about the call itself using curlopts. See an example below that passes in the verbose function to get details on the http call.

WKT

Examples of valid WKT objects:

- 'POLYGON((-19.5 34.1, 27.8 34.1, 35.9 68.1, -25.3 68.1, -19.5 34.1))'
- 'MULTIPOLYGON(((123 38,-116 38,-116 43,-123 43,-123 38)),((-97 41,-93 41,-93 45,-97 45,-97 41)))'
- 'POINT(-120 40)'
- 'LINESTRING(3 4,10 50,20 25)'

Note that GBIF expects counter-clockwise winding order for WKT. You can supply clockwise WKT, but GBIF treats it as an exclusion, so you get all data not inside the WKT area. occ_download() behaves differently in that you should simply get no data back at all with clockwise WKT.

Long WKT

Options for handling long WKT strings: Note that long WKT strings are specially handled when using occ_search or occ_data. Here are the three options for long WKT strings (> 1500 characters), set one of these three via the parameter geom_big:

- asis - the default setting. This means we don’t do anything internally. That is, we just pass on your WKT string just as we’ve done before in this package.
- axe - this option uses the sf package to chop up your WKT string in to many polygons, which then leads to a separate data request for each polygon piece, then we combine all dat back together to give to you. Note that if your WKT string is not of type polygon, we drop back to asis as there’s no way to chop up linestrings, etc. This option will in most cases be slower than the other two options. However, this polygon splitting approach won’t have the problem
of the disconnect between how many records you want and what you actually get back as with the bbox option.

This method uses sf::st_make_grid and sf::st_intersection, which has two parameters cellsize and n. You can tweak those parameters here by tweaking geom_size and geom_n. geom_size seems to be more useful in toggling the number of WKT strings you get back.

See wkt_parse to manually break make WKT bounding box from a larger WKT string, or break a larger WKT string into many smaller ones.

• bbox - this option checks whether your WKT string is longer than 1500 characters, and if it is we create a bounding box from the WKT, do the GBIF search with that bounding box, then prune the resulting data to only those occurrences in your original WKT string. There is a big caveat however. Because we create a bounding box from the WKT, and the limit parameter determines some subset of records to get, then when we prune the resulting data to the WKT, the number of records you get could be less than what you set with your limit parameter. However, you could set the limit to be high enough so that you get all records back found in that bounding box, then you’ll get all the records available within the WKT.

Counts

There is a slight difference in the way records are counted here vs. results from occ_count. For equivalent outcomes, in this function use hasCoordinate=TRUE, and hasGeospatialIssue=FALSE to have the same outcome using occ_count with isGeoreferenced=TRUE

occ_data vs. occ_search

This does nearly the same thing as occ_search(), but is simplified for speed, and is for the most common use case where user just wants occurrence data, and not other information like taxon hierarchies and media (e.g., images). A lot of time in occ_search() is used parsing data to be more useable downstream. We do less of that in this function.

There are a number of data fields GBIF returns that we drop to speed up processing time within R. These fields take extra time to process because they are deeply nested and so take extra time to check if they are empty or not, and if not, figure out how to parse them into a data.frame. The fields are:

• gadm
• media
• facts
• relations
• extensions
• identifiers
• recordedByIDs
• identifiedByIDs

To get these fields use occ_search() instead.

Note

Maximum number of records you can get with this function is 100,000. See https://www.gbif.org/developer/occurrence
occ_data

References
https://www.gbif.org/developer/occurrence#search

See Also
downloads(), occ_search()

Examples

## Not run:
(key <- name_backbone(name='Encelia californica')$speciesKey)
occ_data(taxonKey = key, limit = 4)
(res <- occ_data(taxonKey = key, limit = 400))

# Return 20 results, this is the default by the way
(key <- name_suggest(q='Helianthus annuus', rank='species')$data$key[1])
occ_data(taxonKey = key, limit = 20)

# Instead of getting a taxon key first, you can search for a name directly
# However, note that using this approach (with \code{scientificName="..."})
# you are getting synonyms too. The results for using \code{scientificName}
# and \code{taxonKey} parameters are the same in this case, but I wouldn’t
# be surprised if for some names they return different results
occ_data(scientificName = 'Ursus americanus', curlopts=list(verbose=TRUE))
key <- name_backbone(name = 'Ursus americanus', rank='species')$usageKey
occ_data(occ_taxonKey = key)

# Search by dataset key
occ_data(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', limit=10)

## Search by catalog number
occ_data(catalogNumber="49366", limit=10)

## separate requests: use a vector of strings
occ_data(catalogNumber=c("49366","Bird.27847588"), limit=10)

## one request, many instances of same parameter: use semi-colon sep. string
occ_data(catalogNumber="49366;Bird.27847588", limit=10)

# Use paging parameters (limit and start) to page. Note the different results
# for the two queries below.
occ_data(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', start=10, limit=5)
occ_data(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', start=20, limit=5)

# Many dataset keys

## separate requests: use a vector of strings
occ_data(datasetKey=c("50c9509d-22c7-4a22-a47d-8c48425ef4a7",
"7b5d6a48-f762-11e1-a439-00145eb45e9a"), limit=20)

## one request, many instances of same parameter: use semi-colon sep. string
v="50c9509d-22c7-4a22-a47d-8c48425ef4a7;7b5d6a48-f762-11e1-a439-00145eb45e9a"
occ_data(datasetKey = v, limit=20)

# Search by recorder
occ_data(recordedBy="smith", limit=20)
# Many collector names
## separate requests: use a vector of strings
occ_data(recordedBy=c("smith","BJ Stacey"), limit=10)
## one request, many instances of same parameter: use semi-colon sep. string
occ_data(recordedBy="smith;BJ Stacey", limit=10)

# recordedByID
occ_data(recordedByID="https://orcid.org/0000-0003-1691-239X", limit=20)
## many at once
### separate searches
ids <- c("https://orcid.org/0000-0003-1691-239X",
         "https://orcid.org/0000-0001-7569-1828",
         "https://orcid.org/0000-0002-0596-5376")
res <- occ_data(recordedByID=ids, limit=20)
res[[1]]$data$recordedByIDs[[1]]
res[[2]]$data$recordedByIDs[[1]]
res[[3]]$data$recordedByIDs[[1]]
### all in one search
res <- occ_data(recordedByID=paste0(ids, collapse=";"), limit=20)
unique(vapply(res$data$recordedByIDs, "[[", "", "value"))

# identifiedByID
occ_data(identifiedByID="https://orcid.org/0000-0003-4710-2648", limit=20)

# Pass in curl options for extra fun
occ_data(taxonKey=2433407, limit=20, curlopts=list(verbos TRUE))
occ_data(taxonKey=2433407, limit=20, curlopts = list(
    noprogress = FALSE,
    progressfunction = function(down, up) {
        cat(sprintf("up: %d | down %d\n", up, down))
        return(TRUE)
    })
)
# occ_data(taxonKey=2433407, limit=20, curlopts=list(timeout_ms=1))

# Search for many species
splist <- c("Cyanocitta stelleri", "Junco hyemalis", "Aix sponsa")
keys <- sapply(splist, function(x) name_suggest(x)$data$key[1], USE.NAMES=FALSE)
## separate requests: use a vector of strings
occ_data(taxonKey = keys, limit=5)
## one request, many instances of same parameter: use semi-colon sep. string
occ_data(taxonKey = paste0(keys, collapse = ";"), limit=5)

# Search using a synonym name
# Note that you'll see a message printing out that the accepted name will
# be used
occ_data(scientificName = 'Pulsatilla patens', limit=5)

# Search on latitidue and longitude
occ_data(decimalLatitude=40, decimalLongitude=-120, limit = 10)
# Search on a bounding box
## in well known text format
### polygon
occ_data(geometry = "POLYGON((30.1 10.1, 40.10 40, 20 30.1 10.1))", limit = 20)
### multipolygon
wkt <- 'MULTIPOLYGON(((38 -116 43, -123 43, -123 38), (-97 45, -97 41, -49 45, -97 41)))'
occ_data(geometry = gsub("\n\s+", "", wkt), limit = 20)
### polygon and taxonkey
key <- name_suggest(q = "Aesculus hippocastanum")$data$key[1]
occ_data(taxonKey = key, geometry = "POLYGON((30.1 10.1, 40, 20 40, 10 20, 30.1 10.1))", limit = 20)
## or using bounding box, converted to WKT internally
occ_data(geometry = c(-125.0, 38.4, -121.8, 40.9), limit = 20)

## you can search on many geometry objects
### separate requests: use a vector of strings
wkts <- c("POLYGON((-102.2 46, -102.2 43.7, -93.9 43.7, -93.9 46, -102.2 46))", "POLYGON((30.1 10.1, 40, 40, 10 20, 30.1 10.1))")
occ_data(geometry = wkts, limit = 20)
### one request, many instances of same parameter: use semi-colon sep. string
occ_data(geometry = paste0(wkts, collapse = ";"), limit = 20)

# Search on a long WKT string - too long for a GBIF search API request
## By default, a very long WKT string will likely cause a request failure as
## GBIF only handles strings up to about 1500 characters long. You can leave as is, or
## - Alternatively, you can choose to break up your polygon into many, and do a
data request on each piece, and the output is put back together (see below)
## - Or, 2nd alternatively, you could use the GBIF download API
88

occ_data

12.881750097656322 48.22681126957933, 9.190343847656322 48.693079457106684,
8.75890722656322 50.68283120621287, 5.059484472656322 50.40356146487845,
4.268468847656322 52.377558897655156, 1.455968847656322 53.28027243658647,
0.840734472656322 51.62000971578333, 0.577062597656322 49.32721423860726,
-2.5869999023436776 49.49875947592088, -2.4991092773436776 51.18135535408638,
-2.059661523436776 52.5382562473851, -6.69374902343678 51.674549198756,
-5.31169277343678 50.0990218005776, -6.629968652343678 48.75106196817059,
-7.684656152343678 50.1226363382465, -6.190515527343678 51.83776110910459,
-8.915124902343678 54.77719740243195, -12.079187402343678 54.52294465763567,
-13.573328027343678 53.437631551347174,
-11.28811777343678 53.4899552517918,
-9.17896777343678 53.22769021556159))

wkt <- gsub("\n", " ", wkt)

#### Default option with large WKT string fails
# res <- occ_data(geometry = wkt)

#### if WKT too long, with 'geom_big=bbox': makes into bounding box
if (interactive()){
  res <- occ_data(geometry = wkt, geom_big = "bbox")
}

#### Or, use 'geom_big=axe'
(res <- occ_data(geometry = wkt, geom_big = "axe"))

#### manipulate essentially number of polygons that result, so number of requests
#### default geom_size is 40
#### fewer calls
(res <- occ_data(geometry = wkt, geom_big = "axe", geom_size=50))
#### more calls
(res <- occ_data(geometry = wkt, geom_big = "axe", geom_size=30))

# Search on country
occ_data(country='US', limit=20)
occ_data(country='FR', limit=20)
occ_data(country='DE', limit=20)

#### separate requests: use a vector of strings
occ_data(country=c('US','DE'), limit=20)

#### one request, many instances of same parameter: use semi-colon sep. string
occ_data(country = 'US;DE', limit=20)

# Get only occurrences with lat/long data
occ_data(taxonKey=key, hasCoordinate=TRUE, limit=20)

# Get only occurrences that were recorded as living specimens
occ_data(basisOfRecord="LIVING_SPECIMEN", hasCoordinate=TRUE, limit=20)

## multiple values in a vector = a separate request for each value
occ_data(taxonKey=key,
  basisOfRecord=c("OBSERVATION", "HUMAN_OBSERVATION"), limit=20)

## multiple values in a single string, ";" separated = one request including all values
occ_data(taxonKey=key,
  basisOfRecord="OBSERVATION;HUMAN_OBSERVATION", limit=20)
# Get occurrences for a particular eventDate
occ_data(taxonKey=key, eventDate="2013", limit=20)
occ_data(taxonKey=key, year="2013", limit=20)
occ_data(taxonKey=key, month="6", limit=20)

# Get occurrences based on depth
key <- name_backbone(name='Salmo salar', kingdom='animals')$speciesKey
occ_data(taxonKey=key, depth=1, limit=20)

# Get occurrences based on elevation
key <- name_backbone(name='Puma concolor', kingdom='animals')$speciesKey
occ_data(taxonKey=key, elevation=50, hasCoordinate=TRUE, limit=20)

# Get occurrences based on institutionCode
occ_data(institutionCode="TLMF", limit=20)
### separate requests: use a vector of strings
occ_data(institutionCode=c("TLMF","ArtDatabanken"), limit=20)
### one request, many instances of same parameter: use semi-colon sep. string
occ_data(institutionCode = "TLMF;ArtDatabanken", limit=20)

# Get occurrences based on collectionCode
occ_data(collectionCode="Floristic Databases MV - Higher Plants", limit=20)
### separate requests: use a vector of strings
occ_data(collectionCode=c("Floristic Databases MV - Higher Plants",
"Artport".), limit = 20)
### one request, many instances of same parameter: use semi-colon sep. string
occ_data(collectionCode = "Floristic Databases MV - Higher Plants;Artport",
limit = 20)

# Get only those occurrences with spatial issues
occ_data(taxonKey=key, hasGeospatialIssue=TRUE, limit=20)

# Search using a query string
occ_data(search="kingfisher", limit=20)

# search on repatriated - doesn’t work right now
# occ_data(repatriated = "")

# search on phylumKey
occ_data(phylumKey = 7707728, limit = 5)

# search on kingdomKey
occ_data(kingdomKey = 1, limit = 5)

# search on classKey
occ_data(classKey = 216, limit = 5)

# search on orderKey
occ_data(orderKey = 7192402, limit = 5)

# search on familyKey
occ_data(familyKey = 3925, limit = 5)
# search on genusKey
occ_data(genusKey = 1935496, limit = 5)

# search on establishmentMeans
occ_data(establishmentMeans = "INVASIVE", limit = 5)
occ_data(establishmentMeans = "NATIVE", limit = 5)
occ_data(establishmentMeans = "UNCERTAIN", limit = 5)
### separate requests: use a vector of strings
occ_data(establishmentMeans = c("INVASIVE", "NATIVE"), limit = 5)
### one request, many instances of same parameter: use semi-colon sep. string
occ_data(establishmentMeans = "INVASIVE;NATIVE", limit = 5)

# search on protocol
occ_data(protocol = "DIGIR", limit = 5)

# search on license
occ_data(license = "CC_BY_4_0", limit = 5)

# search on organismId
occ_data(organismId = "100", limit = 5)

# search on publishingOrg
occ_data(publishingOrg = "28eb1a3f-1c15-4a95-931a-4af90ecb574d", limit = 5)

# search on stateProvince
occ_data(stateProvince = "California", limit = 5)

# search on waterBody
occ_data(waterBody = "pacific ocean", limit = 5)

# search on locality
occ_data(locality = "Trondheim", limit = 5)
### separate requests: use a vector of strings
res <- occ_data(locality = c("Trondheim", "Hovekilen"), limit = 5)
res$Trondheim$data
res$Hovekilen$data
### one request, many instances of same parameter: use semi-colon sep. string
occ_data(locality = "Trondheim;Hovekilen", limit = 5)

# Range queries
### See Detail for parameters that support range queries
occ_data(depth='50,100', limit = 20)
### this is not a range search, but does two searches for each depth
occ_data(depth=c(50,100), limit = 20)

### Range search with year
occ_data(year='1999,2000', limit=20)

### Range search with latitude
occ_data(decimalLatitude='29.59,29.6', limit = 20)

### Range search with distanceFromCentroidInMeters
occ_data

occ_data(distanceFromCentroidInMeters = "2000,*") # at least 2km from centroids
occ_data(distanceFromCentroidInMeters = "0,2000") # close to centroids within 2km
occ_data(distanceFromCentroidInMeters = 0) # exactly on centroids

# Search by specimen type status
## Look for possible values of the typeStatus parameter looking at the typestatus dataset
occ_data(typeStatus = 'allotype', limit = 20)$data[,c('name','typeStatus')]

# Search by specimen record number
## This is the record number of the person/group that submitted the data, not GBIF's numbers
## You can see that many different groups have record number 1, so not super helpful
occ_data(recordNumber = 1, limit = 20)$data[,c('name','recordNumber','recordedBy')]

# Search by last time interpreted: Date the record was last modified in GBIF
## The lastInterpreted parameter accepts ISO 8601 format dates, including
## yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Range queries are accepted for lastInterpreted
occ_data(lastInterpreted = '2016-04-02', limit = 20)

# Search for occurrences with images
occ_data(mediaType = 'StillImage', limit = 20)
occ_data(mediaType = 'MovingImage', limit = 20)
occ_data(mediaType = 'Sound', limit = 20)

# Search by continent
## One of africa, antarctica, asia, europe, north_america, oceania, or
## south_america
occ_data(continent = 'south_america', limit = 20)$meta
occ_data(continent = 'africa', limit = 20)$meta
occ_data(continent = 'oceania', limit = 20)$meta
occ_data(continent = 'antarctica', limit = 20)$meta
### separate requests: use a vector of strings
occ_data(continent = c('south_america', 'oceania'), limit = 20)
### one request, many instances of same parameter: use semi-colon sep. string
occ_data(continent = 'south_america;oceania', limit = 20)

# Query based on issues - see Details for options
## one issue
x <- occ_data(taxonKey=1, issue='DEPTH_UNLIKELY', limit = 20)
x$data[,c('name','key','decimalLatitude','decimalLongitude','depth')]
## two issues
occ_data(taxonKey=1, issue=c('DEPTH_UNLIKELY','COORDINATE_ROUNDED'), limit = 20)
# Show all records in the Arizona State Lichen Collection that cant be matched to the GBIF
# backbone properly:
occ_data(datasetKey='84c01a0-f762-11e1-a439-00145eb45e9a',
issue=c('TAXON_MATCH_NONE','TAXON_MATCH_HIGERRANK'), limit = 20)

# Parsing output by issue
(res <- occ_data(geometry='POLYGON((30.1 10.1,40 40,20 40,10 20,30.1 10.1)), limit = 50))
## what do issues mean, can print whole table, or search for matches
head(gbif_issues())
gbif_issues()[ gbif_issues()$code %in% c('cdround','cudc','gass84','txmathi',] ]
## or parse issues in various ways
### remove data rows with certain issue classes
library('magrittr')
res %>% occ_issues(gass84)
### split issues into separate columns
res %>% occ_issues(mutate = "split")
### expand issues to more descriptive names
res %>% occ_issues(mutate = "expand")
### split and expand
res %>% occ_issues(mutate = "split_expanding")
### split, expand, and remove an issue class
res %>% occ_issues(-cudc, mutate = "split_expanding")

## End(Not run)

### occ_download

**Spin up a download request for GBIF occurrence data.**

**Description**

Spin up a download request for GBIF occurrence data.

**Usage**

```r
occ_download(
  ...,
  body = NULL,
  type = "and",
  format = "DWCA",
  user = NULL,
  pwd = NULL,
  email = NULL,
  curlopts = list()
)
```

```r
occ_download_prep(
  ...,
  body = NULL,
  type = "and",
  format = "DWCA",
  user = NULL,
  pwd = NULL,
  email = NULL,
  curlopts = list()
)
```

**Arguments**

... For `occ_download()` and `occ_download_prep()`, one or more objects of class `occ_predicate` or `occ_predicate_list`, created by `pred*` functions (see `download_predicate_dsl`). If you use this, don’t use body parameter.
body

if you prefer to pass in the payload yourself, use this parameter. If you use this, don’t pass anything to the dots. Accepts either an R list, or JSON. JSON is likely easier, since the JSON library jsonlite requires that you unbox strings that shouldn’t be auto-converted to arrays, which is a bit tedious for large queries. optional

type

(character) One of equals (=), and (&), or (l), lessThan (<), lessThanOrEquals (<=), greaterThan (>), greaterThanOrEquals (>=), in, within, not (!), like, isNot-Null

format

(character) The download format. One of 'DWCA' (default), 'SIMPLE_CSV', or 'SPECIES_LIST'

user

(character) User name within GBIF’s website. Required. See "Authentication" below

pwd

(character) User password within GBIF’s website. Required. See "Authentication" below

email

(character) Email address to receive download notice done email. Required. See "Authentication" below

curlopts

list of named curl options passed on to HttpClient. see curl::curl_options for curl options

game

When using the geometry parameter, make sure that your well known text (WKT) is formatted as GBIF expects it. They expect WKT to have a counter-clockwise winding order. For example, the following is clockwise POLYGON((-19.5 34.1, -25.3 68.1, 35.9 68.1, 27.8 34.1, -19.5 34.1)), whereas they expect the other order: POLYGON((-19.5 34.1, 27.8 34.1, 35.9 68.1, -25.3 68.1, -19.5 34.1)) note that coordinate pairs are longitude, latitude, longitude first, then latitude

you should not get any results if you supply WKT that has clockwise winding order.

also note that occ_search()/occ_data() behave differently with respect to WKT in that you can supply clockwise WKT to those functions but they treat it as an exclusion, so get all data not inside the WKT area.

Methods

• occ_download_prep: prepares a download request, but DOES NOT execute it. meant for use with occ_download_queue()
• occ_download: prepares a download request and DOES execute it

Authentication

For user, pwd, and email parameters, you can set them in one of three ways:

• Set them in your .Rprofile file with the names gbif_user, gbif_pwd, and gbif_email
• Set them in your .Renviron/.bash_profile (or similar) file with the names GBIF_USER, GBIF_PWD, and GBIF_EMAIL
• Simply pass strings to each of the parameters in the function call
We strongly recommend the second option - storing your details as environment variables as it's the most widely used way to store secrets.
See ?Startup for help.

Query length

GBIF has a limit of 12,000 characters for a download query. This means that you can have a pretty long query, but at some point it may lead to an error on GBIF’s side and you’ll have to split your query into a few.

Note

see downloads for an overview of GBIF downloads methods

References

See the API docs https://www.gbif.org/developer/occurrence#download for more info, and the predicates docs https://www.gbif.org/developer/occurrence#predicates

See Also

Other downloads: download_predicate_dsl, occ_download_cached(), occ_download_cancel(), occ_download_dataset_activity(), occ_download_datasets(), occ_download_get(), occ_download_import(), occ_download_list(), occ_download_meta(), occ_download_queue(), occ_download_wait()

Examples

## Not run:
# occ_download(pred("basisOfRecord", "LITERATURE"))
# occ_download(pred("taxonKey", 3119195), pred_gt("elevation", 5000))
# occ_download(pred_gt("decimalLatitude", 50))
# occ_download(pred_gte("elevation", 9000))
# occ_download(pred_gte('decimalLatitude', 65))
# occ_download(pred("country", "US"))
# occ_download(pred("institutionCode", "TLMF"))
# occ_download(pred("catalogNumber", 217880))

# download format
# z <- occ_download(pred_gte("decimalLatitude", 75), # format = "SPECIES_LIST")

# res <- occ_download(pred("taxonKey", 7264332), pred("hasCoordinate", TRUE))

# pass output directly, or later, to occ_download_meta for more information
# occ_download(pred_gt('decimalLatitude', 75)) %>% occ_download_meta

# Multiple queries
# occ_download(pred_gte("decimalLatitude", 65),
# pred_lte("decimalLatitude", -65), type="or")
# gg <- occ_download(pred("depth", 80), pred("taxonKey", 2343454),
# type="or")
# x <- occ_download(pred_and(pred_within("POLYGON((-14 42, 9 38, -7 26, -14 42))"),
# pred_gte("elevation", 5000)))

# complex example with many predicates
# shows example of how to do date ranges for both year and month
# res <- occ_download(
# pred_gt("elevation", 5000),
# pred_in("basisOfRecord", c('HUMAN_OBSERVATION', 'OBSERVATION', 'MACHINE_OBSERVATION')),
# pred("country", "US"),
# pred("hasCoordinate", TRUE),
# pred("hasGeospatialIssue", FALSE),
# pred_gte("year", 1999),
# pred_lte("year", 2011),
# pred_gte("month", 3),
# pred_lte("month", 8)
# )

# Using body parameter - pass in your own complete query
## as JSON
query1 <- '{"creator":"sckott",
"notification_address": ["stuff1@gmail.com"],
"predicate": {"type": "and", "predicates": [
{"type": "equals", "key": "TAXON_KEY", "value": "7264332"},
{"type": "equals", "key": "HAS_COORDINATE", "value": "TRUE"}
]}
}'
# res <- occ_download(body = query1, curlopts = list(verb = TRUE))

## as a list
library(jsonlite)
query <- list(
  creator = unbox("sckott"),
  notification_address = "stuff1@gmail.com",
  predicate = list(
    type = unbox("and"),
    predicates = list(
      list(type = unbox("equals"), key = unbox("TAXON_KEY"),
        value = unbox("7264332")),
      list(type = unbox("equals"), key = unbox("HAS_COORDINATE"),
        value = unbox("TRUE"))
    )
  )
)
# res <- occ_download(body = query, curlopts = list(verb = TRUE))

# Prepared query
occ_download_prep(pred("basisOfRecord", "LITERATURE"))
occ_download_prep(pred("basisOfRecord", "LITERATURE"), format = "SIMPLE_CSV")
occ_download_prep(pred("basisOfRecord", "LITERATURE"), format = "SPECIES_LIST")
occ_download_prep(pred_in("taxonKey", c(2977832, 2977901, 2977966, 2977835)))
occ_download_prep(pred_within("POLYGON((-14 42, 9 38, -7 26, -14 42))"))

## a complicated example
occ_download_prep("
occ_download_cached

Check for downloads already in your GBIF account

Description

Check for downloads already in your GBIF account

Usage

occ_download_cached(
    #...
    body = NULL,
    type = "and",
    format = "DWCA",
    user = NULL,
    pwd = NULL,
    email = NULL,
    refresh = FALSE,
    age = 30,
    curlopts = list()
)
Arguments

For `occ_download()` and `occ_download_prep()`, one or more objects of class `occ_predicate` or `occ_predicate_list`, created by `pred*` functions (see `download_predicate_dsl`). If you use this, don’t use body parameter.

body

if you prefer to pass in the payload yourself, use this parameter. If you use this, don’t pass anything to the dots. Accepts either an R list, or JSON. JSON is likely easier, since the JSON library `jsonlite` requires that you unbox strings that shouldn’t be auto-converted to arrays, which is a bit tedious for large queries. optional

type

(character) One of equals (=), and (\&), or (!), lessThan (<), lessThanOrEquals (<=), greaterThan (>), greaterThanOrEquals (>=), in, within, not (!), like, isNot-Null

format

(character) The download format. One of 'DWCA' (default), 'SIMPLE_CSV', or 'SPECIES_LIST'

user

(character) User name within GBIF’s website. Required. See "Authentication" below

pwd

(character) User password within GBIF’s website. Required. See "Authentication" below

email

(character) Email address to receive download notice done email. Required. See "Authentication" below

refresh

(logical) refresh your list of downloads. on the first request of each R session we’ll cache your stored GBIF occurrence downloads locally. you can refresh this list by setting `refresh=TRUE`; if you’re in the same R session, and you’ve done many download requests, then refreshing may be a good idea if you’re using this function

age

(integer) number of days after which you want a new download. default: 30

curlopts

list of named curl options passed on to `HttpClient`. see `curl::curl_options` for curl options

Note

see `downloads` for an overview of GBIF downloads methods

See Also

Other downloads: `download_predicate_dsl`, `occ_download_cancel()`, `occ_download_dataset_activity()`, `occ_download_datasets()`, `occ_download_get()`, `occ_download_import()`, `occ_download_list()`, `occ_download_meta()`, `occ_download_queue()`, `occ_download_wait()`, `occ_download()`

Examples

```r
## Not run:
# these are examples from the package maintainer’s account;
# outcomes will vary by user
occ_download_cached(pred_gte("elevation", 12000L))
occ_download_cached(pred("catalogNumber", 217880))
```
occ_download_cancel

Description

Cancel a download creation process.

Usage

occ_download_cancel(key, user = NULL, pwd = NULL, curlopts = list())

occ_download_cancel_staged(
  user = NULL,
  pwd = NULL,
  limit = 20,
  start = 0,
  curlopts = list()
)

Arguments

key (character) A key generated from a request, like that from occ_download. Required.
user (character) User name within GBIF’s website. Required. See Details.
pwd (character) User password within GBIF’s website. Required. See Details.
curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options
limit Number of records to return. Default: 20
start Record number to start at. Default: 0

Details

Note, these functions only cancel a job in progress. If your download is already prepared for you, this won’t do anything to change that.

occ_download_cancel cancels a specific job by download key - returns success message
occ_download_cancel_staged cancels all jobs with status RUNNING or PREPARING - if none are found, returns a message saying so - if some found, they are cancelled, returning message saying so
Note

see downloads for an overview of GBIF downloads methods

See Also

Other downloads: download_predicate_dsl, occ_download_cached(), occ_download_dataset_activity(), occ_download_datasets(), occ_download_get(), occ_download_import(), occ_download_list(), occ_download_meta(), occ_download_queue(), occ_download_wait(), occ_download()

Examples

```r
## Not run:
# occ_download_cancel(key="0003984-140910143529206")
# occ_download_cancel_staged()

## End(Not run)
```

occ_download_datasets List datasets for a download

Description

List datasets for a download

Usage

```r
occ_download_datasets(key, limit = 20, start = 0, curlopts = list())
```

Arguments

- **key**: A key generated from a request, like that from `occ_download()`
- **limit**: (integer/numeric) Number of records to return. Default: 20, Max: 1000
- **start**: (integer/numeric) Record number to start at. Default: 0
- **curlopts**: list of named curl options passed on to `HttpClient`. see `curl::curl_options` for curl options

Value

A list with two slots:

- **meta**: a single row data.frame with columns: offset, limit, endofrecords, count
- **results**: a tibble with the results, of three columns: downloadKey, datasetKey, numberRecords

Note

see downloads for an overview of GBIF downloads methods
See Also

Other downloads: download_predicate_dsl, occ_download_cached(), occ_download_cancel(),
occ_download_dataset_activity(), occ_download_get(), occ_download_import(), occ_download_list(),
occ_download_meta(), occ_download_queue(), occ_download_wait(), occ_download()

Examples

```r
## Not run:
occ_download_datasets(key="0003983-140910143529206")
occ_download_datasets(key="0003983-140910143529206", limit = 3)
occ_download_datasets(key="0003983-140910143529206", limit = 3, start = 10)

## End(Not run)
```

---

## occ_download_dataset_activity

*Lists the downloads activity of a dataset*

### Description

Lists the downloads activity of a dataset

### Usage

```r
occ_download_dataset_activity(
  dataset,
  limit = 20,
  start = 0,
  curlopts = list()
)
```

### Arguments

- **dataset** (character) A dataset key
- **limit** (integer/numeric) Number of records to return. Default: 20, Max: 1000
- **start** (integer/numeric) Record number to start at. Default: 0
- **curlopts** list of named curl options passed on to `HttpClient`. see `curl::curl_options`

### Value

A list with two slots:

- meta: a single row data.frame with columns: offset, limit, endofrecords, count
- results: a tibble with the nested data flattened, with many columns with the same `download` or `download.request` prefixes
occ_download_get

Get a download from GBIF.

Description

Get a download from GBIF.

Usage

occ_download_get(key, path = ".", overwrite = FALSE, ...)

Arguments

key
path
overwrite
...

A key generated from a request, like that from occ_download
Path to write zip file to. Default: ".", with a .zip appended to the end.
Will only overwrite existing path if TRUE.

named curl options passed on to crul::verb-GET. see curl::curl_options() for curl options

Examples

## Not run:
res <- occ_download_dataset_activity("7f2edc10-f762-11e1-a439-00145eb45e9a")
res
res$meta
res$meta$count

# pagination
occ_download_dataset_activity("7f2edc10-f762-11e1-a439-00145eb45e9a", limit = 3000)
occ_download_dataset_activity("7f2edc10-f762-11e1-a439-00145eb45e9a", limit = 3, start = 10)

## End(Not run)
Details

Downloads the zip file to a directory you specify on your machine. `crul::HttpClient()` is used internally to write the zip file to disk. See `crul::writing-options`. This function only downloads the file. See `occ_download_import` to open a downloaded file in your R session. The speed of this function is of course proportional to the size of the file to download. For example, a 58 MB file on my machine took about 26 seconds.

Note

see downloads for an overview of GBIF downloads methods

This function used to check for HTTP response content type, but it has changed enough that we no longer check it. If you run into issues with this function, open an issue in the GitHub repository.

See Also

Other downloads: `download_predicate_dsl`, `occ_download_cached()`, `occ_download_cancel()`, `occ_download_dataset_activity()`, `occ_download_datasets()`, `occ_download_import()`, `occ_download_list()`, `occ_download_meta()`, `occ_download_queue()`, `occ_download_wait()`, `occ_download()`

Examples

```r
## Not run:
occ_download_get("0000066-140928181241064")
occ_download_get("0003983-140910143529206", overwrite = TRUE)

## End(Not run)
```

---

**occ_download_import**  
Import a downloaded file from GBIF.

**Description**

Import a downloaded file from GBIF.

**Usage**

```r
occ_download_import(
  x = NULL,
  key = NULL,
  path = ".",
  fill = FALSE,
  encoding = "UTF-8",
  ...
)
```

```r
as.download(path = ".", key = NULL)
```
## S3 method for class 'character'
as.download(path = ".", key = NULL)

## S3 method for class 'download'
as.download(path = ".", key = NULL)

### Arguments

- **x**: The output of a call to `occ_download_get`
- **key**: A key generated from a request, like that from `occ_download`
- **path**: Path to unzip file to. Default: "." Writes to folder matching zip file name
- **fill**: (logical) (default: FALSE). If TRUE then in case the rows have unequal length, blank fields are implicitly filled. passed on to fill parameter in `data.table::fread`
- **encoding**: (character) encoding to read in data; passed to `data.table::fread`. default: "UTF-8". other allowed options: "Latin-1" and "unknown". see ?data.table::fread docs

### Details

You can provide either x as input, or both key and path. We use `data.table::fread()` internally to read data.

### Value

a tibble (data.frame)

### Problems reading data

You may run into errors when using `occ_download_import()`; most often these are due to `data.table::fread()` not being able to parse the occurrence.txt file correctly. The fill parameter passes down to `data.table::fread()` and the ... allows you to pass on any other parameters that `data.table::fread()` accepts. Read the docs for fread for help.

### countryCode result column and Namibia

The country code for Namibia is "NA". Unfortunately in R an "NA" string will be read in to R as an NA/missing. To avoid this, in this function we read in the data, then convert an NA/missing values to the character string "NA". When a country code is truly missing it will be an empty string.

### Note

see downloads for an overview of GBIF downloads methods
See Also

Other downloads: download_predicate_dsl, occ_download_cached(), occ_download_cancel(), occ_download_dataset_activity(), occ_download_datasets(), occ_download_get(), occ_download_list(), occ_download_meta(), occ_download_queue(), occ_download_wait(), occ_download()

Examples

```r
## Not run:
# First, kick off at least 1 download, then wait for the job to be complete
# Then use your download keys
res <- occ_download_get(key="0000066-140928181241064", overwrite=TRUE)
occ_download_import(res)

occ_download_get(key="0000066-140928181241064", overwrite = TRUE) %>%
  occ_download_import

# coerce a file path to the right class to feed to occ_download_import
# as.download("0000066-140928181241064.zip")
# as.download(key = "0000066-140928181241064")
# occ_download_import(as.download("0000066-140928181241064.zip"))

# download a dump that has a CSV file
# res <- occ_download_get(key = "0001369-160509122628363", overwrite=TRUE)
# occ_download_import(res)
# occ_download_import(key = "0001369-160509122628363")

# download and import a species list (in csv format)
# x <- occ_download_get("0000172-190415153152247")
# occ_download_import(x)

## End(Not run)
```

---

### occ_download_list

Lists the downloads created by a user.

**Description**

Lists the downloads created by a user.

**Usage**

```r
occ_download_list(
  user = NULL,
  pwd = NULL,
  limit = 20,
  start = 0,
  curlopts = list()
)
```
Arguments

- **user** *(character)* User name within GBIF’s website. Required. See Details.
- **pwd** *(character)* User password within GBIF’s website. Required. See Details.
- **limit** *(integer/numeric)* Number of records to return. Default: 20, Max: 1000
- **start** *(integer/numeric)* Record number to start at. Default: 0
- **curlopts** list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Value

- a list with two slots:
  - `meta`: a single row data.frame with columns: offset, limit, endofrecords, count
  - `results`: a tibble with the nested data flattened, with many columns with the same request

Note

see downloads for an overview of GBIF downloads methods

See Also

Other downloads: download_predicate_dsl, occ_download_cached(), occ_download_cancel(), occ_download_dataset_activity(), occ_download_datasets(), occ_download_get(), occ_download_import(), occ_download_meta(), occ_download_queue(), occ_download_wait(), occ_download()

Examples

```r
## Not run:
occ_download_list(user="sckott")
occ_download_list(user="sckott", limit = 5)
occ_download_list(user="sckott", start = 21)
## End(Not run)
```

---

**occ_download_meta** Retrieves the occurrence download metadata by its unique key.

Description

Retrieves the occurrence download metadata by its unique key.

Usage

```r
occ_download_meta(key, curlopts = list())
```
Arguments

key
A key generated from a request, like that from `occ_download`
curlopts
list of named curl options passed on to `HttpClient`. see `curl::curl_options`

Value

an object of class `occ_download_meta`, a list with slots for the download key, the DOI assigned to the download, license link, the request details you sent in the `occ_download()` request, and metadata about the size and date/time of the request

Note

see `downloads` for an overview of GBIF downloads methods

See Also

Other downloads: `download_predicate_dsl`, `occ_download_cached()`, `occ_download_cancel()`, `occ_download_dataset_activity()`, `occ_download_datasets()`, `occ_download_get()`, `occ_download_import()`, `occ_download_list()`, `occ_download_queue()`, `occ_download_wait()`, `occ_download()`

Examples

```r
## Not run:
occ_download_meta(key="0003983-140910143529206")
occ_download_meta("0000066-140928181241064")

## End(Not run)
```

---

**occ_download_queue**

Download requests in a queue

Description

Download requests in a queue

Usage

`occ_download_queue(..., .list = list(), status_ping = 10)`

Arguments

... any number of `occ_download()` requests
.list any number of `occ_download_prep()` requests
status_ping (integer) seconds between pings checking status of the download request. generally larger numbers for larger requests. default: 10 (i.e., 10 seconds). must be 10 or greater
Details

This function is a convenience wrapper around \texttt{occ_download()}, allowing the user to kick off any number of requests, while abiding by GBIF rules of 3 concurrent requests per user.

Value

a list of \texttt{occ_download} class objects, see \texttt{occ_download_get()} to fetch data

How it works

It works by using lazy evaluation to collect your requests into a queue (but does not use lazy evaluation if use the \texttt{.list} parameter). Then it kicks of the first 3 requests. Then in a while loop, we check status of those requests, and when any request finishes (see \texttt{When is a job done?} below), we kick off the next, and so on. So in theory, there may not always strictly be 3 running concurrently, but the function will usually provide for 3 running concurrently.

When is a job done?

We mark a job as done by checking the \texttt{/occurrence/download/} API route with our \texttt{occ_download_meta()} function. If the status of the job is any of "succeeded", "killed", or "cancelled", then we mark the job as done and move on to other jobs in the queue.

Beware

This function is still in development. There's a lot of complexity to this problem. We'll be rolling out fixes and improvements in future versions of the package, so expect to have to adjust your code with new versions.

Note

see downloads for an overview of GBIF downloads methods

See Also

Other downloads: \texttt{download_predicate_dsl}, \texttt{occ_download_cached()}, \texttt{occ_download_cancel()}, \texttt{occ_download_dataset_activity()}, \texttt{occ_download_datasets()}, \texttt{occ_download_get()}, \texttt{occ_download_import()}, \texttt{occ_download_list()}, \texttt{occ_download_meta()}, \texttt{occ_download_wait()}, \texttt{occ_download()}

Examples

```r
## Not run:
if (interactive()) { # dont run in automated example runs, too costly
# passing \texttt{occ_download()} requests via ...
out <- occ_download_queue(
  occ_download(pred('taxonKey', 3119195), pred("year", 1976)),
  occ_download(pred('taxonKey', 3119195), pred("year", 2001)),
  occ_download(pred('taxonKey', 3119195), pred("year", 2001),
    pred_lte("month", 8)),
  occ_download(pred('taxonKey', 5229208), pred("year", 2011)),
  occ_download(pred('taxonKey', 2480946), pred("year", 2015)),
```
occ_download_wait

Wait for an occurrence download to be done

Description

Wait for an occurrence download to be done
Usage

occ_download_wait(
x, status_ping = 5, curlopts = list(http_version = 2), quiet = FALSE)
)

Arguments

x and object of class occ_download or downloadkey
status_ping (integer) seconds between each occ_download_meta() request. default is 5, and cannot be < 3
curlopts (list) curl options, as named list, passed on to occ_download_meta()
quiet (logical) suppress messages. default: FALSE

Value

an object of class occ_download_meta, see occ_download_meta() for details

Note

occ_download_queue() is similar, but handles many requests at once; occ_download_wait handles one request at a time

See Also

Other downloads: download_predicate_dsl, occ_download_cached(), occ_download_cancel(), occ_download_dataset_activity(), occ_download_datasets(), occ_download_get(), occ_download_import(), occ_download_list(), occ_download_meta(), occ_download_queue(), occ_download()

Examples

## Not run:
x <- occ_download(
   pred("taxonKey", 9206251),
   pred_in("country", c("US", "MX")),
   pred_gte("year", 1971)
)
res <- occ_download_wait(x)
occ_download_meta(x)

# works also with a downloadkey
occ_download_wait("0000066-140928181241064")

## End(Not run)
occ_facet

Facet GBIF occurrences

Description

Facet GBIF occurrences

Usage

occ_facet(facet, facetMincount = NULL, curlopts = list(), ...)

Arguments

- **facet** (character) a character vector of length 1 or greater. Required.
- **facetMincount** (numeric) minimum number of records to be included in the faceting results
- **curlopts** list of named curl options passed on to HttpClient. see curl::curl_options for curl options
- **...** Facet parameters, such as for paging based on each facet variable, e.g., country.facetLimit

Details

All fields can be faceted on except for last "lastInterpreted", "eventDate", and "geometry"

If a faceted variable is not found, it is silently dropped, returning nothing for that query

Value

A list of tibbles (data.frame's) for each facet (each element of the facet parameter).

See Also

occ_search() also has faceting ability, but can include occurrence data in addition to facets.

Examples

```r
## Not run:
occ_facet(facet = "country")

# facetMincount - minimum number of records to be included
# in the faceting results
occ_facet(facet = "country", facetMincount = 3000000L)
oc_facet(facet = c("country", "basisOfRecord"))

# paging with many facets
occ_facet(
  facet = c("country", "basisOfRecord", "hasCoordinate"),
  country.facetLimit = 3,
  basisOfRecord.facetLimit = 6
)
```
occ_get

Get data for GBIF occurrences by occurrence key

Description

Get data for GBIF occurrences by occurrence key

Usage

occ_get(
  key,
  fields = "minimal",
  curlopts = list(),
  return = NULL,
  verbatim = NULL
)

occ_get_verbatim(key, fields = "minimal", curlopts = list())

Arguments

key (numeric/integer) one or more occurrence keys. required
fields (character) Default ("minimal") will return just taxon name, key, latitude, and longitude. 'all' returns all fields. Or specify each field you want returned by name, e.g. fields = c('name', 'decimalLatitude', 'altitude').
curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options
return Defunct. All components are returned now; index to the one(s) you want
verbatim Defunct. verbatim records can now be retrieved using occ_get_verbatim()

Value

For occ_get a list of lists. For occ_get_verbatim a data.frame
References

https://www.gbif.org/developer/occurrence#occurrence

Examples

```r
# Not run:
occ_get(key=855998194)

# many occurrences
occ_get(key=c(101010, 240713150, 855998194))

# Verbatim data
occ_get_verbatim(key=855998194)
occ_get_verbatim(key=855998194, fields='all')
occ_get_verbatim(key=855998194,
  fields=c('scientificName', 'lastCrawled', 'county'))
occ_get_verbatim(key=c(855998194, 620594291), fields='all')
occ_get_verbatim(key=c(855998194, 620594291),
  fields=c('scientificName', 'decimalLatitude', 'basisOfRecord'))

# curl options, pass in a named list
occ_get(key=855998194, curlopts = list(verbos=TRUE))
```

---

`occ_issues`

*Parse and examine further GBIF occurrence issues on a dataset.*

**Description**

Parse and examine further GBIF occurrence issues on a dataset.

**Usage**

```r
occ_issues(.data, ..., mutate = NULL)
```

**Arguments**

- `.data` Output from a call to `occ_search()`, `occ_data()`, or `occ_download_import()`. The data from `occ_download_import` is just a regular data.frame so you can pass in a data.frame to this function, but if it doesn’t have certain columns it will fail.
- `...` Named parameters to only get back (e.g. cdround), or to remove (e.g. -cdround).
- `mutate` (character) One of:
  - `split` Split issues into new columns.
• expand Expand issue abbreviated codes into descriptive names. For downloads datasets, this is not super useful since the issues come to you as expanded already.
• split_expand Split into new columns, and expand issue names.

For split and split_expand, values in cells become y ("yes") or n ("no")

Details
See also the vignette Cleaning data using GBIF issues
Note that you can also query based on issues, e.g., occ_search(taxonKey=1, issue='DEPTH_UNLIKELY'). However, I imagine it’s more likely that you want to search for occurrences based on a taxonomic name, or geographic area, not based on issues, so it makes sense to pull data down, then clean as needed using this function.
This function only affects the data element in the gbif class that is returned from a call to occ_search(). Maybe in a future version we will remove the associated records from the hierarchy and media elements as they are removed from the data element.
You’ll notice that we sort columns to make it easier to glimpse the important parts of your data, namely taxonomic name, taxon key, latitude and longitude, and the issues. The columns are unchanged otherwise.

References

Examples
```r
## Not run:
# what do issues mean, can print whole table
head(gbif_issues())
# or just occurrence related issues
gbif_issues()[which(gbif_issues()$type %in% c("occurrence")),]
# or search for matches
iss <- c('cdround', 'cudc', 'gass84', 'txmathi')
gbif_issues()[gbif_issues()$code %in% iss,]

# compare out data to after occ_issues use
(out <- occ_search(limit=100))
out %>% occ_issues(cdround)

# occ_data
(out <- occ_data(limit=100))
out %>% occ_issues(cdround)

# Parsing output by issue
(res <- occ_data(
  geometry='POLYGON((30.1 10.1,40 40,20 40,10 20,30.1 10.1))',
  limit = 600))

# or parse issues in various ways
### include only rows with cdround issue
```
gg <- res %>% occ_issues(cround)
NROW(res$data)
NROW(gg$data)
head(res$data)[,c(1:5)]
head(gg$data)[,c(1:5)]

### remove data rows with certain issue classes
res %>% occ_issues(-cround, -cudc)

### split issues into separate columns
res %>% occ_issues(mutate = "split")
res %>% occ_issues(-cudc, -mdatunl, mutate = "split")
res %>% occ_issues(gass84, mutate = "split")

### expand issues to more descriptive names
res %>% occ_issues(mutate = "expand")

### split and expand
res %>% occ_issues(mutate = "split_expand")

### split, expand, and remove an issue class
res %>% occ_issues(-cround, mutate = "split_expand")

## Or you can use occ_issues without %>%
occ_issues(res, -cround, mutate = "split_expand")

# from GBIF downloaded data via occ_download_* functions
res <- occ_download_get(key="0000066-140928181241064", overwrite=TRUE)
x <- occ_download_import(res)
occ_issues(x, -txmathi)
occ_issues(x, txmathi)
occ_issues(x, gass84)
occ_issues(x, zerocd)
occ_issues(x, gass84, txmathi)
occ_issues(x, mutate = "split")
occ_issues(x, -gass84, mutate = "split")
occ_issues(x, mutate = "expand")
occ_issues(x, mutate = "split_expand")

# occ_search/occ_data with many inputs - give slightly different output
# format than normal 2482598, 2498387
xyz <- occ_data(taxonKey = c(9362842, 2492483, 2435099), limit = 300)
xyz
length(xyz) # length 3
names(xyz) # matches taxonKey values passed in
occ_issues(xyz, -gass84)
occ_issues(xyz, -cround)
occ_issues(xyz, -cround, -gass84)

## End(Not run)
occ_metadata

Search for catalog numbers, collection codes, collector names, and institution codes.

Description

Search for catalog numbers, collection codes, collector names, and institution codes.

Usage

```r
occ_metadata(
  type = "catalogNumber",
  q = NULL,
  limit = 5,
  pretty = TRUE,
  curlopts = list()
)
```

Arguments

- **type**: Type of data, one of catalogNumber, collectionCode, recordedBy, or institutionCode. Unique partial strings work too, like ’cat’ for catalogNumber
- **q**: Search term
- **limit**: Number of results, default=5
- **pretty**: Pretty as true (Default) uses cat to print data, FALSE gives character strings.
- **curlopts**: list of named curl options passed on to HttpClient. see curl::curl_options for curl options

References

https://www.gbif.org/developer/occurrence#search

Examples

```r
## Not run:
# catalog number
occ_metadata(type = "catalogNumber", q=122)

# collection code
occ_metadata(type = "collectionCode", q=12)

# institution code
occ_metadata(type = "institutionCode", q='GB')

# recorded by
occ_metadata(type = "recordedBy", q='scott')
```
# data as character strings
occ_metadata(type = "catalogNumber", q=122, pretty=FALSE)

# Change number of results returned
occ_metadata(type = "catalogNumber", q=122, limit=10)

# Partial unique type strings work too
occ_metadata(type = "cat", q=122)

# Pass on curl options
occ_metadata(type = "cat", q=122, curlopts = list(verbose = TRUE))

## End(Not run)

---

**occ_search**  
*Search for GBIF occurrences*

**Description**

Search for GBIF occurrences

**Usage**

```r
occ_search(
  taxonKey = NULL,
  scientificName = NULL,
  country = NULL,
  publishingCountry = NULL,
  hasCoordinate = NULL,
  typeStatus = NULL,
  recordNumber = NULL,
  lastInterpreted = NULL,
  continent = NULL,
  geometry = NULL,
  geom_big = "asis",
  geom_size = 40,
  geom_n = 10,
  recordedBy = NULL,
  recordedByID = NULL,
  identifiedByID = NULL,
  basisOfRecord = NULL,
  datasetKey = NULL,
  eventDate = NULL,
  catalogNumber = NULL,
  year = NULL,
  month = NULL,
  decimalLatitude = NULL,
  decimalLongitude = NULL,
```
elevation = NULL,
depth = NULL,
institutionCode = NULL,
collectionCode = NULL,
hasGeospatialIssue = NULL,
issue = NULL,
search = NULL,
mediaType = NULL,
subgenusKey = NULL,
repatriated = NULL,
phylumKey = NULL,
kindomKey = NULL,
classKey = NULL,
orderKey = NULL,
familyKey = NULL,
genusKey = NULL,
speciesKey = NULL,
establishmentMeans = NULL,
degreeOfEstablishment = NULL,
protocol = NULL,
license = NULL,
organismId = NULL,
publishingOrg = NULL,
stateProvince = NULL,
waterBody = NULL,
locality = NULL,
ocurrenceStatus = "PRESENT",
gadmGid = NULL,
coordinateUncertaintyInMeters = NULL,
verbatimScientificName = NULL,
eventId = NULL,
identifiedBy = NULL,
networkKey = NULL,
verbatimTaxonId = NULL,
ocurrenceId = NULL,
organismQuantity = NULL,
organismQuantityType = NULL,
relativeOrganismQuantity = NULL,
iucnRedListCategory = NULL,
lifeStage = NULL,
isInCluster = NULL,
distanceFromCentroidInMeters = NULL,
limit = 500,
start = 0,
fields = "all",
return = NULL,
facet = NULL,
facetMincount = NULL,
facetMultiselect = NULL,
skip_validate = TRUE,
curlopts = list(http_version = 2),
}
)

Arguments

taxonKey (numeric) A taxon key from the GBIF backbone. All included and synonym taxa are included in the search, so a search for aves with taxonKey=212 will match all birds, no matter which species. You can pass many keys to occ_search(taxonKey=c(1,212)).

scientificName A scientific name from the GBIF backbone. All included and synonym taxa are included in the search.

country (character) The 2-letter country code (ISO-3166-1) in which the occurrence was recorded. enumeration_country().

publishingCountry The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded. See enumeration_country().

hasCoordinate (logical) Return only occurrence records with lat/long data (TRUE) or all records (FALSE, default).

typeStatus Type status of the specimen. One of many options.

recordNumber Number recorded by collector of the data, different from GBIF record number.

lastInterpreted Date the record was last modified in GBIF, in ISO 8601 format: yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Supports range queries, 'smaller,larger' (e.g., '1990,1991', whereas '1991,1990' wouldn’t work).

continent The source supplied continent.

- "africa"
- "antarctica"
- "asia"
- "europe"
- "north_america"
- "oceania"
- "south_america"

Continent is not inferred but only populated if provided by the dataset publisher. Applying this filter may exclude many relevant records.

geometry (character) Searches for occurrences inside a polygon in Well Known Text (WKT) format. A WKT shape written as either

- "POINT"
- "LINESTRING"
- "LINEARRING"
- "POLYGON"
• "MULTIPOLYGON"
For Example, "POLYGON((37.08 46.86,38.06 46.86,38.06 47.28,37.08 47.28,
37.0 46.8))". See also the section WKT below.

geom_big
(character) One of "axe", "bbox", or "asis" (default).

geom_size
(integer) An integer indicating size of the cell. Default: 40.

geom_n
(integer) An integer indicating number of cells in each dimension. Default: 10.

recordedBy
(character) The person who recorded the occurrence.

recordedByID
(character) Identifier (e.g. ORCID) for the person who recorded the occurrence.

identifiedByID
(character) Identifier (e.g. ORCID) for the person who provided the taxonomic
identification of the occurrence.

basisOfRecord
(character) The specific nature of the data record. See here.

• "FOSSIL_SPECIMEN"
• "HUMAN_OBSERVATION"
• "MATERIAL_CITATION"
• "MATERIAL_SAMPLE"
• "LIVING_SPECIMEN"
• "MACHINE_OBSERVATION"
• "OBSERVATION"
• "PRESERVED_SPECIMEN"
• "OCCURRENCE"

datasetKey
(character) The occurrence dataset uuid key. That can be found in the dataset
page url. For example, "7e380070-f762-11e1-a439-00145eb45e9a" is the key
for Natural History Museum (London) Collection Specimens.

eventDate
'1991,1990' wouldn't work).

catalogNumber
(character) An identifier of any form assigned by the source within a physical
collection or digital dataset for the record which may not unique, but should be
fairly unique in combination with the institution and collection code.

year
The 4 digit year. A year of 98 will be interpreted as AD 98. Supports range
queries, 'smaller,larger' (e.g., '1990,1991', whereas 1991, 1990' wouldn't work).

month
The month of the year, starting with 1 for January. Supports range queries,
'smaller,larger' (e.g., '1,2', whereas '2,1' wouldn't work).

decimalLatitude
Latitude in decimals between -90 and 90 based on WGS84. Supports range
queries, 'smaller,larger' (e.g., '25.30', whereas '30.25' wouldn't work).

decimalLongitude
Longitude in decimals between -180 and 180 based on WGS84. Supports range
queries (e.g., '-0.4,-0.2', whereas '-0.2,-0.4' wouldn't work).

elevation
Elevation in meters above sea level. Supports range queries, 'smaller,larger'
(e.g., '5,30', whereas '30,5' wouldn't work).
depth

Depth in meters relative to elevation. For example 10 meters below a lake surface with given elevation. Supports range queries, ‘smaller,larger’ (e.g., ‘5,30’, whereas ‘30,5’ wouldn’t work).

institutionCode

An identifier of any form assigned by the source to identify the institution the record belongs to.

collectionCode

(character) An identifier of any form assigned by the source to identify the physical collection or digital dataset uniquely within the text of an institution.

hasGeospatialIssue

(logical) Includes/excludes occurrence records which contain spatial issues (as determined in our record interpretation), i.e. hasGeospatialIssue=TRUE returns only those records with spatial issues while hasGeospatialIssue=FALSE returns only records without spatial issues. The absence of this parameter returns any record with or without spatial issues.

issue

(character) One or more of many possible issues with each occurrence record. Issues passed to this parameter filter results by the issue. One of many options. See here for definitions.

search

(character) Query terms. The value for this parameter can be a simple word or a phrase. For example, search="puma"

mediaType

(character) Media type of "MovingImage", "Sound", or "StillImage".

subgenusKey

(numeric) Subgenus classification key.

repatriated

(character) Searches for records whose publishing country is different to the country where the record was recorded in.

phylumKey

(numeric) Phylum classification key.

kingdomKey

(numeric) Kingdom classification key.

classKey

(numeric) Class classification key.

orderKey

(numeric) Order classification key.

familyKey

(numeric) Family classification key.

genusKey

(numeric) Genus classification key.

speciesKey

(numeric) Species classification key.

establishmentMeans

(character) provides information about whether an organism or organisms have been introduced to a given place and time through the direct or indirect activity of modern humans.

• "Introduced"
• "Native"
• "NativeReintroduced"
• "Vagrant"
• "Uncertain"
• "IntroducedAssistedColonisation"

degreeOfEstablishment

(character) Provides information about degree to which an Organism survives, reproduces, and expands its range at the given place and time. One of many options.
protocol (character) Protocol or mechanism used to provide the occurrence record. One of many options.

license (character) The type license applied to the dataset or record.
- "CC0_1_0"
- "CC_BY_4_0"
- "CC_BY_NC_4_0"

organismId (numeric) An identifier for the Organism instance (as opposed to a particular digital record of the Organism). May be a globally unique identifier or an identifier specific to the data set.

publishingOrg (character) The publishing organization key (a UUID).

stateProvince (character) The name of the next smaller administrative region than country (state, province, canton, department, region, etc.) in which the Location occurs.

waterBody (character) The name of the water body in which the locations occur.

locality (character) The specific description of the place.

occurrenceStatus (character) Default is "PRESENT". Specify whether search should return "PRESENT" or "ABSENT" data.

gadmGid (character) The gadm id of the area occurrences are desired from. https://gadm.org/.

coordinateUncertaintyInMeters A number or range between 0-1,000,000 which specifies the desired coordinate uncertainty. A coordinateUncertaintyInMeters=1000 will be interpreted all records with exactly 1000m. Supports range queries, 'smaller,larger' (e.g., '1000,10000', whereas '10000,1000' wouldn't work).

verbatimScientificName (character) Scientific name as provided by the source.

eventId (character) identifier(s) for a sampling event.

identifiedBy (character) names of people, groups, or organizations.

networkKey (character) The occurrence network key (a uuid) who assigned the Taxon to the subject.

verbatimTaxonId (character) The taxon identifier provided to GBIF by the data publisher.

occurrenceId (character) occurrence id from source.

organismQuantity A number or range which specifies the desired organism quantity. An organismQuantity=5 will be interpreted all records with exactly 5. Supports range queries, smaller,larger (e.g., '5,20', whereas '20,5' wouldn’t work).

organismQuantityType (character) The type of quantification system used for the quantity of organisms. For example, "individuals" or "biomass".

relativeOrganismQuantity (numeric) A relativeOrganismQuantity=0.1 will be interpreted all records with exactly 0.1 The relative measurement of the quantity of the organism (a number between 0-1). Supports range queries, "smaller,larger" (e.g., '0.1,0.5', whereas '0.5,0.1' wouldn’t work).
iucnRedListCategory
(character) The IUCN threat status category.
- "NE" (Not Evaluated)
- "DD" (Data Deficient)
- "LC" (Least Concern)
- "NT" (Near Threatened)
- "VU" (Vulnerable)
- "EN" (Endangered)
- "CR" (Critically Endangered)
- "EX" (Extinct)
- "EW" (Extinct in the Wild)

lifeStage (character) the life stage of the occurrence. One of many options.

isInCluster (logical) identify potentially related records on GBIF.

distanceFromCentroidInMeters
A number or range. A value of "2000,*" means at least 2km from known centroids. A value of "0" would mean occurrences exactly on known centroids. A value of "0,2000" would mean within 2km of centroids. Max value is 5000.

limit
Number of records to return. Default: 500. Note that the per request maximum is 300, but since we set it at 500 for the function, we do two requests to get you the 500 records (if there are that many). Note that there is a hard maximum of 100,000, which is calculated as the limit+start, so start=99,000 and limit=2000 won’t work

start
Record number to start at. Use in combination with limit to page through results. Note that we do the paging internally for you, but you can manually set the start parameter

fields (character) Default ("all") returns all fields. 'minimal' returns just taxon name, key, datasetKey, latitude, and longitude. Or specify each field you want returned by name, e.g. fields = c('name','latitude','elevation').

return Defunct. All components (meta, hierarchy, data, media, facets) are returned now; index to the one(s) you want. See occ_data() if you just want the data component

facet (character) a character vector of length 1 or greater. Required.

facetMincount (numeric) minimum number of records to be included in the faceting results

facetMultiselect (logical) Set to TRUE to still return counts for values that are not currently filtered. See examples. Default: FALSE

Faceting: All fields can be faceted on except for last "lastInterpreted", "eventDate", and "geometry"
You can do facet searches alongside searching occurrence data, and return both, or only return facets, or only occurrence data, etc.

skip_validate (logical) whether to skip wellknown::validate_wkt call or not. passed down to check_wkt(). Default: TRUE

curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options

... additional facet parameters
**Value**

An object of class gbif, which is a S3 class list, with slots for metadata (meta), the occurrence data itself (data), the taxonomic hierarchy data (hier), and media metadata (media). In addition, the object has attributes listing the user supplied arguments and whether it was a ‘single’ or ‘many’ search; that is, if you supply two values of the datasetKey parameter to searches are done, and it’s a ‘many’. meta is a list of length four with offset, limit, endOfRecords and count fields. data is a tibble (aka data.frame). hier is a list of data.frames of the unique set of taxa found, where each data.frame is its taxonomic classification. media is a list of media objects, where each element holds a set of metadata about the media object.

**Multiple values passed to a parameter**

There are some parameters you can pass multiple values to in a vector, each value of which produces a different request (multiple different requests = c("a","b")). Some parameters allow multiple values to be passed in the same request (multiple same request = "a;b") in a semicolon separated string (e.g., 'a;b'); if given we’ll do a single request with that parameter repeated for each value given (e.g., foo=a&foo=b if the parameter is foo).

See article [Multiple Values](#).

**Hierarchies**

Hierarchies are returned with each occurrence object. There is no option to return them from the API. However, within the occ_search function you can select whether to return just hierarchies, just data, all of data and hierarchies and metadata, or just metadata. If all hierarchies are the same we just return one for you.

**curl debugging**

You can pass parameters not defined in this function into the call to the GBIF API to control things about the call itself using curlopts. See an example below that passes in the verbose function to get details on the http call.

**WKT**

Examples of valid WKT objects:

- ‘POLYGON(( -19.5 34.1, 27.8 34.1, 35.9 68.1, -25.3 68.1, -19.5 34.1 ))’
- ‘MULTIPOLYGON((( -123 38, -116 38, -116 43, -123 43, -123 38 ), ( -97 41, -93 41, -93 45, -97 45, -97 41 )))’
- ‘POINT( -120 40 )’
- ‘LINESTRING( 3 4, 10 50, 20 25 )’

Note that GBIF expects counter-clockwise winding order for WKT. You can supply clockwise WKT, but GBIF treats it as an exclusion, so you get all data not inside the WKT area. occ_download() behaves differently in that you should simply get no data back at all with clockwise WKT.
Long WKT

Options for handling long WKT strings: Note that long WKT strings are specially handled when using `occ_search` or `occ_data`. Here are the three options for long WKT strings (> 1500 characters), set one of these three via the parameter `geom_big`:

- **asis** - the default setting. This means we don’t do anything internally. That is, we just pass on your WKT string just as we’ve done before in this package.

- **axe** - this option uses the `sf` package to chop up your WKT string into many polygons, which then leads to a separate data request for each polygon piece, then we combine all dat back together to give to you. Note that if your WKT string is not of type polygon, we drop back to `asis` as there’s no way to chop up linestrings, etc. This option will in most cases be slower than the other two options. However, this polygon splitting approach won’t have the problem of the disconnect between how many records you want and what you actually get back as with the `bbox` option.

  This method uses `sf::st_make_grid` and `sf::st_intersection`, which has two parameters `cellsize` and `n`. You can tweak those parameters here by tweaking `geom_size` and `geom_n`. `geom_size` seems to be more useful in toggling the number of WKT strings you get back.

  See `wkt_parse` to manually break make WKT bounding box from a larger WKT string, or break a larger WKT string into many smaller ones.

- **bbox** - this option checks whether your WKT string is longer than 1500 characters, and if it is we create a bounding box from the WKT, do the GBIF search with that bounding box, then prune the resulting data to only those occurrences in your original WKT string. There is a big caveat however. Because we create a bounding box from the WKT, and the `limit` parameter determines some subset of records to get, then when we prune the resulting data to the WKT, the number of records you get could be less than what you set with your `limit` parameter. However, you could set the limit to be high enough so that you get all records back found in that bounding box, then you’ll get all the records available within the WKT.

Counts

There is a slight difference in the way records are counted here vs. results from `occ_count`. For equivalent outcomes, in this function use `hasCoordinate=TRUE`, and `hasGeospatialIssue=FALSE` to have the same outcome using `occ_count` with `isGeoreferenced=TRUE`.

Note

Maximum number of records you can get with this function is 100,000. See https://www.gbif.org/developer/occurrence

References

https://www.gbif.org/developer/occurrence#search

See Also

downloads(), `occ_data()`
Examples

## Not run:
# Search by species name, using \code{\link[name_backbone]} first to get key
(key <- name_suggest(q='Helianthus annuus', rank='species')$data$key[1])
occ_search(taxonKey=key, limit=2)

# Return 20 results, this is the default by the way
occ_search(taxonKey=key, limit=20)

# Get just metadata
occ_search(taxonKey=key, limit=0)$meta

# Instead of getting a taxon key first, you can search for a name directly
# However, note that using this approach (with \code{scientificName="..."})
# you are getting synonyms too. The results for using \code{scientificName} and
# \code{taxonKey} parameters are the same in this case, but I wouldn't be surprised if for some
# names they return different results
occ_search(scientificName = 'Ursus americanus')
key <- name_backbone(name = 'Ursus americanus', rank='species')$usageKey
occ_search(taxonKey = key)

# Search by dataset key
occ_search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', limit=20)$data

# Search by catalog number
occ_search(catalogNumber="49366", limit=20)
# separate requests: use a vector of strings
occ_search(catalogNumber=c("49366","Bird.27847588"), limit=10)
# one request, many instances of same parameter: use semi-colon sep. string
occ_search(catalogNumber="49366;Bird.27847588", limit=10)

# Get all data, not just lat/long and name
occ_search(taxonKey=key, fields='all', limit=20)

# Or get specific fields. Note that this isn't done on GBIF's side of things. This
# is done in R, but before you get the return object, so other fields are garbage
# collected
occ_search(taxonKey=key, fields=c('name','basisOfRecord','protocol'), limit=20)

# Use paging parameters (limit and start) to page. Note the different results
# for the two queries below.
occ_search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a',start=10,limit=5)$data
occ_search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a',start=20,limit=5)$data

# Many dataset keys
# separate requests: use a vector of strings
occ_search(datasetKey=c("50c9509d-22c7-4a22-a47d-8c48425ef4a7",
    "7b5d6a48-f762-11e1-a439-00145eb45e9a"), limit=20)
# one request, many instances of same parameter: use semi-colon sep. string
v="50c9509d-22c7-4a22-a47d-8c48425ef4a7;7b5d6a48-f762-11e1-a439-00145eb45e9a"
occ_search(datasetKey = v, limit=20)
# Occurrence data: lat/long data, and associated metadata with occurrences
## The `data` slot has a data.frame of all data together
## for easy manipulation
occ_search(taxonKey=key, limit=20)$data

# Taxonomic hierarchy data
## In the `hier` slot
occ_search(taxonKey=key, limit=10)$hier

# Search by recorder
occ_search(recordedBy="smith", limit=20)

# Many collector names
occ_search(recordedBy=c("smith","BJ Stacey"), limit=20)

# recordedByID
occ_search(recordedByID="https://orcid.org/0000-0003-1691-239X", limit=20)

# identifiedByID
occ_search(identifiedByID="https://orcid.org/0000-0003-4710-2648", limit=20)

# Pass in curl options for extra fun
occ_search(taxonKey=2433407, limit=20, curlopts=list(verbose=TRUE))$hier
occ_search(taxonKey=2433407, limit=20, curlopts = list(noprogress = FALSE,
    progressfunction = function(down, up) {
        cat(sprintf("up: %d | down %d\n", up, down))
        return(TRUE)
    }
))$hier

# Search for many species
splist <- c('Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa')
keys <- sapply(splist, function(x) name_suggest(x)$data$key[1], USE.NAMES=FALSE)
## separate requests: use a vector of strings
occ_search(taxonKey = keys, limit=5)
## one request, many instances of same parameter: use semi-colon sep. string
occ_search(taxonKey = paste0(keys, collapse = ";"), limit=5)

# Search using a synonym name
# Note that you'll see a message printing out that the accepted name will be used
occ_search(scientificName = 'Pulsatilla patens', fields = c('name','scientificName'), limit=5)

# Search on latitude and longitude
occ_search(decimalLatitude=48, decimalLongitude=10)

# Search on a bounding box
## in well known text format
### polygon
occ_search

occ_search(geom='POLYGON((30.1 10.1,40 40,40,10 20,30.1 10.1))', limit=20)
### multipolygon
((30.1 -97,40 -97,40,40,10 -97,30.1 -97)))'
occ_search(geom = gsub("\n\s+", "", wkt), limit = 20)

## taxonKey + WKT
key <- name_suggest(q='Aesculus hippocastanum')$data$key[1]
occ_search(taxonKey=key, geom='POLYGON((30.1 10.1,40 40,40,10 20,30.1 10.1))', limit=20)

## or using bounding box, converted to WKT internally
occ_search(geom=(-125.0,38.4,-121.8,40.9), limit=20)

# Search on a long WKT string - too long for a GBIF search API request
## We internally convert your WKT string to a bounding box
## then do the query
## then clip the results down to just those in the original polygon
## - Alternatively, you can set the parameter geom_big="bbox"

wkt <- gsub("\n\s+", "", wkt)
#### Default option with large WKT string fails

```r
res <- occ_search(geometry = wkt)
```

#### if WKT too long, with 'geom_big=bbox': makes into bounding box

```r
res <- occ_search(geometry = wkt, geom_big = "bbox")$data
```

#### Or, use 'geom_big=axe'

```r
(res <- occ_search(geometry = wkt, geom_big = "axe"))
```

##### manipulate essentially number of polygons that result, so number of requests

- default geom_size is 40
- fewer calls

```r
(res <- occ_search(geometry = wkt, geom_big = "axe", geom_size=50))
```

- more calls

```r
(res <- occ_search(geometry = wkt, geom_big = "axe", geom_size=30))
```

# Search on country

```r
occ_search(country='US', fields=c('name','country'), limit=20)
occ_search(country='FR', fields=c('name','country'), limit=20)
occ_search(country='DE', fields=c('name','country'), limit=20)
```

### separate requests: use a vector of strings

```r
occ_search(country=c('US','DE'), limit=20)
```

### one request, many instances of same parameter: use semi-colon sep. string

```r
occ_search(country = 'US;DE', limit=20)
```

# Get only occurrences with lat/long data

```r
countryKey=key, hasCoordinate=TRUE, limit=20)
```

# Get only occurrences that were recorded as living specimens

```r
taxonKey=key, basisOfRecord="LIVING_SPECIMEN", hasCoordinate=TRUE, limit=20)
```

## multiple values in a vector = a separate request for each value

```r
taxonKey=key,
    basisOfRecord=c("LIVING_SPECIMEN", "HUMAN_OBSERVATION"), limit=20)
```

## multiple values in a single string, ";" separated = one request including all values

```r
taxonKey=key,
    basisOfRecord="LIVING_SPECIMEN;HUMAN_OBSERVATION", limit=20)
```

# Get occurrences for a particular eventDate

```r
taxonKey=key, eventDate="2013", limit=20)
taxonKey=key, year="2013", limit=20)
taxonKey=key, month="6", limit=20)
```

# Get occurrences based on depth

```r
key <- name_backbone(name='Salmo salar', kingdom='animals')$speciesKey
occ_search(taxonKey=key, depth="5", limit=20)
```

# Get occurrences based on elevation

```r
key <- name_backbone(name='Puma concolor', kingdom='animals')$speciesKey
occ_search(taxonKey=key, elevation=50, hasCoordinate=TRUE, limit=20)
```

# Get occurrences based on institutionCode

```r
occ_search(institutionCode="TLMF", limit=20)
```

### separate requests: use a vector of strings
# Search using a query string
occ_search(search = "kingfisher", limit = 20)

# search on repatriated - doesn't work right now
# occ_search(repatriated = "")

# Search using a query string
occ_search(phylumKey = 7707728, limit = 5)

# Search using a query string
occ_search(kingdomKey = 1, limit = 5)

# Search using a query string
occ_search(classKey = 216, limit = 5)

# Search using a query string
occ_search(orderKey = 7192402, limit = 5)

# Search using a query string
occ_search(familyKey = 3925, limit = 5)

# Search using a query string
occ_search(genusKey = 1935496, limit = 5)

# Search using a query string
occ_search(establishmentMeans = "INVASIVE", limit = 5)
occ_search(establishmentMeans = "NATIVE", limit = 5)
occ_search(establishmentMeans = "UNCERTAIN", limit = 5)

# Search using a query string
occ_search(protocol = "DIGIR", limit = 5)

# Search using a query string
occ_search(license = "CC_BY_4_0", limit = 5)

# Search using a query string
occ_search(organismId = "100", limit = 5)

# Search using a query string
occ_search(publishingOrg = "28eb1a3f-1c15-4a95-931a-4af90ecb574d", limit = 5)

# Search using a query string
occ_search(stateProvince = "")
occ_search(stateProvince = "California", limit = 5)

# search on waterBody
occ_search(waterBody = "AMAZONAS BASIN, RIO JURUA", limit = 5)

# search on locality
res <- occ_search(locality = c("Trondheim", "Hovekilen"), limit = 5)
res$Trondheim$data
res$Hovekilen$data

# Range queries
## See Detail for parameters that support range queries
occ_search(depth="50,100") # this is a range depth, with lower/upper limits in character string
occ_search(depth=c(50,100)) # this is not a range search, but does two searches for each depth

## Range search with year
occ_search(year='1999,2000', limit=20)

## Range search with latitude
occ_search(decimalLatitude='29.59,29.6')

## Range search with distanceFromCentroidInMeters
occ_search(distanceFromCentroidInMeters = "2000,*") # at least 2km from centroids
occ_search(distanceFromCentroidInMeters = "0,2000") # close to centroids within 2km
occ_search(distanceFromCentroidInMeters = 0) # exactly on centroids

# Search by specimen type status
## Look for possible values of the typeStatus parameter looking at the typestatus dataset
occ_search(typeStatus = 'allotype', fields = c('name','typeStatus'))

# Search by specimen record number
## This is the record number of the person/group that submitted the data, not GBIF's numbers
## You can see that many different groups have record number 1, so not super helpful
occ_search(recordNumber = 1, fields = c('name','recordNumber','recordedBy'))

# Search by last time interpreted: Date the record was last modified in GBIF
## The lastInterpreted parameter accepts ISO 8601 format dates, including
## yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Range queries are accepted for lastInterpreted
occ_search(lastInterpreted = '2014-04-02', fields = c('name','lastInterpreted'))

# Search by continent
## One of africa, antarctica, asia, europe, north_america, oceania, or south_america
occ_search(continent = 'south_america')$meta
occ_search(continent = 'africa')$meta
occ_search(continent = 'oceania')$meta
occ_search(continent = 'antarctica')$meta

# Search for occurrences with images
occ_search(mediaType = 'StillImage')$media
occ_search(mediaType = 'MovingImage')$media
occ_search(mediaType = 'Sound')$media
# Query based on issues - see Details for options
## one issue
occ_search(taxonKey=1, issue='DEPTH_UNLIKELY', fields =
c('name', 'key', 'decimalLatitude', 'decimalLongitude', 'depth'))
## two issues
occ_search(taxonKey=1, issue=c('DEPTH_UNLIKELY', 'COORDINATE_ROUNDED'))

# Show all records in the Arizona State Lichen Collection that can't be matched to the GBIF backbone properly:
occ_search(datasetKey='84c0e1a0-f762-11e1-a439-00145eb45e9a',
  issue=c('TAXON_MATCH_NONE', 'TAXON_MATCH_HIGHER_RANK'))

# Parsing output by issue
(res <-occ_search(geometry='POLYGON((30.1 10.1, 40 40, 20 40, 10 20, 30.1 10.1))', limit = 50))

## what do issues mean, can print whole table, or search for matches
head(gbif_issues())
gbif_issues()[ gbif_issues()$code %in% c('cdround', 'cudc', 'gass84', 'txmathi'), ]

## or parse issues in various ways
### remove data rows with certain issue classes
library('magrittr')
res %>% occ_issues(gass84)
### split issues into separate columns
res %>% occ_issues(mutate = "split")
### expand issues to more descriptive names
res %>% occ_issues(mutate = "expand")
### split and expand
res %>% occ_issues(mutate = "split_expand")
### split, expand, and remove an issue class
res %>% occ_issues(-cudc, mutate = "split_expand")

# If you try multiple values for two different parameters you are wacked on the hand
# occ_search(taxonKey=c(2482598,2492010), recordedBy=c("smith","BJ Stacey"))

# Get a lot of data, here 1500 records for Helianthus annuus
# out <- occ_search(taxonKey=key, limit=1500)
# nrow(out$data)

### the WKT string is fine, but GBIF says bad polygon

# occ_search(geometry = gsub("\n", '', wkt))
### unable to parse due to last number pair needing two numbers, not one
# wkt <- 'POLYGON((-178.5 64.8,-165.9 59.2,-147.3 59.0,-130.7 51.0,-125.8))'
# occ_search(geometry = wkt)

### unable to parse due to unclosed string
# wkt <- 'POLYGON((-178.5 64.8,-165.9 59.2,-147.3 59.0,-130.7 51.0))'
# occ_search(geometry = wkt)

### another of the same
# wkt <- 'POLYGON((-178.5 64.8,-165.9 59.2,-147.3 59.0,-130.7 51.0,-125.8 36.7))'
# occ_search(geometry = wkt)

### returns no results
# wkt <- 'LINESTRING(3 4,10 50,20 25)'
# occ_search(geometry = wkt)

### Apparently a point is allowed, but errors
# wkt <- 'POINT(45 -122)'
# occ_search(geometry = wkt)

### Faceting
x <- occ_search(facet = "country", limit = 0)
x$facets
x <- occ_search(facet = "establishmentMeans", limit = 10)
x$facets
x$data
x <- occ_search(facet = c("country", "basisOfRecord"), limit = 10)
x$data
x$facets
x$facets$country
x$facets$basisOfRecord
x$facets$basisOfRecord$count
x <- occ_search(facet = "country", facetMincount = 30000000L, limit = 10)
x$facets
x$data

# paging per each faceted variable
(x <- occ_search(
  facet = c("country", "basisOfRecord", "hasCoordinate"),
  country.facetLimit = 3,
  basisOfRecord.facetLimit = 6,
  limit = 0
))
x$facets

# You can set limit=0 to get number of results found
occ_search(datasetKey = '7b5d6a48-f762-11e1-a439-00145eb45e9a', limit = 0)$meta
occ_search(scientificName = 'Ursus americanus', limit = 0)$meta
occ_search(scientificName = 'Ursus americanus', limit = 0)$meta

## End(Not run)
organizations

Description

Organizations metadata.

Usage

organizations(
  data = "all",
  country = NULL,
  uuid = NULL,
  query = NULL,
  limit = 100,
  start = NULL,
  curlopts = list()
)

Arguments

data (character) The type of data to get. One or more of: 'organization', 'contact', 'endpoint', 'identifier', 'tag', 'machineTag', 'comment', 'hostedDataset', 'ownedDataset', 'deleted', 'pending', 'nonPublishing', or the special 'all'. Default: 'all'
country (character) Filters by country.
uuid (character) UUID of the data node provider. This must be specified if data is anything other than 'all', 'deleted', 'pending', or 'nonPublishing'.
query (character) Query nodes. Only used when data='all'
limit Number of records to return. Default: 100. Maximum: 1000.
start Record number to start at. Default: 0. Use in combination with limit to page through results.
curlopts list of named curl options passed on to HttpClient. see curl::curl_options for curl options

Value

A list of length of two, consisting of a data.frame meta when uuid is NULL, and data which can either be a list or a data.frame depending on the requested type of data.

References

https://www.gbif.org/developer/registry#organizations
## Examples

```r
## Not run:
organizations(limit=5)
organizations(query="france", limit=5)
organizations(country = "SPAIN")
organizations(uuid="4b4b2111-ee51-45f5-bf5e-f535f4a1c9dc")
organizations(data=\'contact\', uuid="4b4b2111-ee51-45f5-bf5e-f535f4a1c9dc")
organizations(data=\'pending\')
organizations(data=c('contact', 'endpoint'),
              uuid="4b4b2111-ee51-45f5-bf5e-f535f4a1c9dc")

# Pass on curl options
organizations(query="spain", curlopts = list(verbose=TRUE))

## End(Not run)
```

### parsenames

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

#### Description

Parse taxon names using the GBIF name parser.

#### Usage

```r
parsenames(scientificname, curlopts = list())
```

#### Arguments

- `scientificname`  
  A character vector of scientific names.

- `curlopts`  
  list of named curl options passed on to `HttpClient`. see `curl::curl_options` for curl options

#### Value

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

#### Author(s)

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

#### References

[https://www.gbif.org/developer/species#parser](https://www.gbif.org/developer/species#parser)
Examples

```r
## Not run:
parsenames(scientificname='x Agropogon littoralis')
parsenames(c('Arrhenatherum elatius var. elatius',
              'Secale cereale subsp. cereale', 'Secale cereale ssp. cereale',
              'Vanessa atalanta (Linnaeus, 1758)'))
parsenames("Ajuga pyramidata")
parsenames("Ajuga pyramidata x reptans")

# Pass on curl options
# res <- parsenames(c('Arrhenatherum elatius var. elatius',
#                      'Secale cereale subsp. cereale', 'Secale cereale ssp. cereale',
#                      'Vanessa atalanta (Linnaeus, 1758)'), curlopts=list(verbose=TRUE))

## End(Not run)
```

---

**Description**

- `density_spplist()`: service no longer provided
- `densitylist()`: service no longer provided
- `gbifdata()`: service no longer provided
- `gbifmap_dens()`: service no longer provided
- `gbifmap_list()`: service no longer provided
- `occurrencedensity()`: service no longer provided
- `providers()`: service no longer provided
- `resources()`: service no longer provided
- `taxoncount()`: service no longer provided
- `taxonget()`: service no longer provided
- `taxonsearch()`: service no longer provided
- `stylegeojson()`: moving this functionality to spocc package, will be removed soon
- `togojson()`: moving this functionality to spocc package, will be removed soon
- `gist()`: moving this functionality to spocc package, will be removed soon
- `occ_spellcheck()`: GBIF has removed the spellCheck parameter from their API

**Details**

The above functions have been removed. See [https://github.com/ropensci/rgbif](https://github.com/ropensci/rgbif) and poke around the code if you want to find the old functions in previous versions of the package.
rgb_country_codes

Look up 2 character ISO country codes

Description

Look up 2 character ISO country codes

Usage

rgb_country_codes(country_name, fuzzy = FALSE, ...)

Arguments

country_name   Name of country to look up
fuzzy           If TRUE, uses agrep to do fuzzy search on names.
...             Further arguments passed on to agrep or grep

Examples

## Not run:
rgb_country_codes(country_name="United")

## End(Not run)

taxrank

Get the possible values to be used for (taxonomic) rank arguments in GBIF API methods.

description

Get the possible values to be used for (taxonomic) rank arguments in GBIF API methods.

Usage

taxrank()

Examples

## Not run:
taxrank()

## End(Not run)
**Description**

parse wkt into smaller bits

**Usage**

```r
wkt_parse(wkt, geom_big, geom_size = 40, geom_n = 10)
```

**Arguments**

- `wkt` (character) A WKT string. Required.
- `geom_big` (character) One of "axe" or "bbox". Required.
- `geom_size` (integer) An integer indicating size of the cell. Default: 40.
- `geom_n` (integer) An integer indicating number of cells in each dimension. Default: 10.

**Examples**

```r
wkt <- "POLYGON((13.26349675655365 52.53991761181831,18.36115300655365 54.11445544271845, 21.87677800655365 ... 52.753205655365 54.722367709361, 55.3293425655365 54.324133588528, 13.26349675655365))"
```
wkt <- gsub("\n", " ", wkt)

if (requireNamespace("sf", quietly=TRUE)) {
  # to a bounding box in wkt format
  wkt_parse(wkt, geom_big = "bbox")

  # to many wkt strings, chopped up from input
  wkt_parse(wkt, geom_big = "axe")
  wkt_parse(wkt, geom_big = "axe", 60)
  wkt_parse(wkt, geom_big = "axe", 30)
  wkt_parse(wkt, geom_big = "axe", 20)
  wkt_parse(wkt, geom_big = "axe", 10)
  wkt_parse(wkt, geom_big = "axe", 5)
}

Index

* downloads
  download_predicate_dsl, 22
  occ_download, 92
  occ_download_cached, 96
  occ_download_cancel, 98
  occ_download_dataset_activity, 100
  occ_download_datasets, 99
  occ_download_get, 101
  occ_download_import, 102
  occ_download_list, 104
  occ_download_meta, 105
  occ_download_queue, 106
  occ_download_wait, 108

as.download(occ_download_import), 102

crul::HttpClient, 7, 9, 11, 14, 16, 45, 48

crul::verb-GET, 28, 101
crul::writing-options, 102

data.table::fread, 103
data.table::fread(), 103
dataset, 6
dataset_comment (dataset_uuid_funs), 16
dataset_consortium
  (dataset_uuid_funs), 16
dataset_contact (dataset_uuid_funs), 16
dataset_doi, 9
dataset_duplicate (dataset_list_funs), 11
dataset_endpoint (dataset_uuid_funs), 16
dataset_export (dataset_search), 12
dataset_get (dataset_uuid_funs), 16
dataset_gridded, 10
dataset_identifier (dataset_uuid_funs), 16
dataset_list_funs, 11
dataset_machinetag (dataset_uuid_funs), 16
dataset_metrics (dataset_uuid_funs), 16
dataset_networks (dataset_uuid_funs), 16
dataset_noendpoint (dataset_list_funs), 11
dataset_process (dataset_uuid_funs), 16
dataset_search, 12
dataset_suggest (dataset_search), 12
dataset_tag (dataset_uuid_funs), 16
dataset_uuid_funs, 16
datasets, 8
density_spplist(), 135
densitylist(), 135
derived_dataset, 17
derived_dataset_prep (derived_dataset), 17
downloads(), 85, 124

elevation, 27
enumeration, 30
enumeration_country (enumeration), 30
gbif_bbox2wkt, 31
gbif_citation, 32
gbif_geocode, 33
gbif_issues, 34
gbif_issues_lookup, 34
gbif_names, 35
gbif_oai, 36
gbif_oai_get_records (gbif_oai), 36
gbif_oai_identify (gbif_oai), 36
gbif_oai_list_identifiers (gbif_oai), 36
gbif_oai_list_metadataformats
  (gbif_oai), 36
gbif_oai_list_records (gbif_oai), 36
gbif_oai_list_sets (gbif_oai), 36
gbif_photos, 38
gbif_wkt2bbox (gbif_bbox2wkt), 31
gbifdata(), 135
gbifmap_dens(), 135
gbifmap_list(), 135
gist(). 135

HttpClient, 8, 30, 39, 51, 52, 59, 61, 63, 65, 67, 69, 71, 82, 93, 97–100, 105, 106, 110, 111, 115, 122, 133, 134

installations, 39

lit_count (lit_search), 40
lit_search, 40

map_fetch, 43
map_fetch(), 49
mvt_fetch, 47
mvt_fetch(), 46

name_backbone, 49
name_backbone_checklist, 52
name_backbone_verbose (name_backbone), 49
name_issues, 55
name_issues(), 65
name_lookup, 56
name_lookup(), 65
name_parse, 61
name_suggest, 62
name_usage, 64
name_usage(), 55

network, 67
network_constituents (network), 67
networks, 69
nodes, 70

occ_count, 72, 84, 124
occ_count(), 76
occ_count_, 75
occ_count_basis_of_record (occ_count_), 75
occ_count_basis_of_record(), 73
occ_count_country (occ_count_), 75
occ_count_country(), 73
occ_count_pub_country (occ_count_), 75
occ_count_pub_country(), 73
occ_count_year (occ_count_), 75
occ_count_year(), 73
occ_data, 76, 83, 124
occ_data(), 32, 93, 112, 122, 124
occ_download, 27, 92, 97, 99–102, 104–107, 109
occ_download(), 20, 21, 83, 99, 106, 107, 123
occ_downloadCached, 27, 94, 96, 99–102, 104–107, 109
occ_downloadCached(), 21
occ_download_cancel, 27, 94, 97, 98, 100–102, 104–107, 109
occ_download_cancel(), 21
occ_download_cancel_staged
(occ_download_cancel), 98
occ_download_cancel_staged(), 21
occ_download_dataset_activity, 27, 94, 97, 99, 100, 102, 104–107, 109
occ_download_dataset_activity(), 21
occ_download_datasets, 27, 94, 97, 99, 99, 101, 102, 104–107, 109
occ_download_datasets(), 21
occ_download_get, 27, 94, 97, 99–101, 101, 104–107, 109
occ_download_get(), 21, 97, 107
occ_download_import, 27, 94, 97, 99–102, 102, 105–107, 109
occ_download_import(), 21, 112
occ_download_list, 27, 94, 97, 99–102, 104, 106, 107, 109
occ_download_list(), 21
occ_download_meta, 27, 94, 97, 99–102, 104, 105, 107, 109
occ_download_meta(), 21, 32, 107
occ_download_prep (occ_download), 92
occ_download_prep(), 21, 106
occ_download_queue, 27, 94, 97, 99–102, 104–106, 106, 109
occ_download_queue(), 21, 93, 109
occ_download_wait, 27, 94, 97, 99–102, 104–107, 108
occ_download_wait(), 21
occ_facet, 110
occ_get, 111
occ_get_verbatim (occ_get), 111
occ_issues, 112
occ_metadata, 115
occ_search, 83, 116, 124
occ_search(), 20, 32, 84, 85, 93, 110, 112,
INDEX

113
occ_spellcheck(), 135
occurrencedensity(), 135
options(), 20
organizations, 133
parsenames, 134
pred (download_predicate_dsl), 22
pred_and (download_predicate_dsl), 22
pred_default (download_predicate_dsl), 22
pred_gt (download_predicate_dsl), 22
pred_gte (download_predicate_dsl), 22
pred_in (download_predicate_dsl), 22
pred_isnull (download_predicate_dsl), 22
pred_like (download_predicate_dsl), 22
pred_lt (download_predicate_dsl), 22
pred_lte (download_predicate_dsl), 22
pred_not (download_predicate_dsl), 22
pred_notnull (download_predicate_dsl), 22
pred_or (download_predicate_dsl), 22
pred_within (download_predicate_dsl), 22
providers(), 135
resources(), 135
rgb_country_codes, 136
rgbif (rgbif-package), 3
rgbif-defunct, 135
rgbif-package, 3
stylegeojson(), 135
Sys.setenv(), 20
taxoncount(), 135
taxonget(), 135
taxonsearch(), 135
taxrank, 136
togeojson(), 135
wkt_parse, 84, 124, 137