Package 'rgbif'

April 19, 2017

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'; <http://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, and getting counts of occurrence records.

Version 0.9.8

License MIT + file LICENSE

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

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

LazyData true

LazyLoad true

VignetteBuilder knitr

Imports xml2, ggplot2, crul (>= 0.3.4), data.table, whisker, magrittr, jsonlite (>= 0.9.16), oai (>= 0.2.2), geoaxe, tibble, wicket (>= 0.2.0)

Suggests roxygen2 (>= 6.0.1), testthat, knitr, reshape2, maps, sp, rgeos

RoxygenNote 6.0.1

NeedsCompilation no

Author Scott Chamberlain [aut, cre],
Vijay Barve [ctb],
Dan Mcglinn [ctb]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2017-04-18 23:54:33 UTC

R topics documented:

`rgbif-package` *Interface to the Global Biodiversity Information Facility API.*

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 `http://www.gbif.org/` via their API.

A note about the old GBIF API

The old GBIF API was at http://data.gbif.org/tutorial/services, but is now defunct - that is, not available anymore. We used to have functions that worked with the old API, but those functions are now not available anymore because GBIF made the old API defunct.

Documentation for the GBIF API

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

Note

See many-values for discussion of how functions vary in how they accept values (single vs. many for the same HTTP request vs. many for different HTTP requests)

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

Karthik Ram <karthik@ropensci.org>

Dan Mcglinn <danmcglinn@gmail.com>

Vijay Barve <vijay.barve@gmail.com>

check_wkt *Check input WKT*

Description

Check input WKT

Usage

```
check_wkt(wkt = NULL)
```

Arguments

wkt	(character) one or more Well Known Text objects

Examples

```
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)')

wkt <- 'MULTIPOLYGON(((30 20, 45 40, 10 40, 30 20)),
((15 5, 40 10, 10 20, 5 10, 15 5)))'
check_wkt(gsub("\n", '', wkt))

# 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
# wkt <- 'POLYGON((30.1 10.1, 10 20, 20 60, 60 60, 30.1 a))'
# check_wkt(wkt)

# this passes this check, but isn't valid for GBIF
wkt <- 'POLYGON((-178.59375 64.83258989321493,-165.9375 59.24622380205539,
-147.3046875 59.065977905449806,-130.78125 51.04484764446178,
-125.859375 36.70806354647625,-112.1484375 23.367471303759686,
-105.1171875 16.093320185359257,-86.8359375 9.23767076398516,
-82.96875 2.9485268155066175,-82.6171875 -14.812060061226388,
-74.8828125 -18.849111862023985,
-77.34375 -47.661687803329166,-84.375 -49.975955187343295,
174.7265625 -50.649460483096114,
179.296875 -42.19189902447192,-176.8359375 -35.634976650677295,
176.8359375 -31.835565983656227,163.4765625 -6.528187613695323,
152.578125 1.894796132058301,135.703125 4.702353722559447,
127.96875 15.077427674847987,127.96875 23.689804541429606,
139.921875 32.06861069132688,149.4140625 42.65416193033991,
159.2578125 48.3160811030533,168.3984375 57.019804336633165,
178.2421875 59.95776046458139,-179.6484375 61.16708631440347,
-178.59375 64.83258989321493))'
```

```
check_wkt(gsub("\n", '', wkt))

# many wkt's, semi-colon separated, for many repeated "geometry" args
wkt <- "POLYGON((-102.2 46.0,-93.9 46.0,-93.9 43.7,-102.2 43.7,-102.2 46.0))
;POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))"
check_wkt(gsub("\n", '', 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.
`by`	(character) One of georeferenced, basisOfRecord, country, or publishingCountry.
`countries`	(numeric) Number of countries to facet on, or a vector of country names
`removezeros`	(logical) Default is 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)

# Pass in country names instead
countries <- isocodes$code[1:10]
count_facet(by='country', countries=countries, removezeros = TRUE)
```

```
# get occurrences by georeferenced state
## across all records
count_facet(by='georeferenced')

## by keys
out <- count_facet(keys, by='georeferenced')
library("reshape2")
dcast(out, .id ~ georeferenced)

# by basisOfRecord
count_facet(by="basisOfRecord")

## 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

`data`	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`
`type`	Type of dataset. Options: include occurrence, checklist, metadata, or sampling_event.
`uuid`	UUID of the data node provider. This must be specified if data is anything other than `all`
`query`	Query term(s). Only used when `data=all`
`id`	A metadata document id.
`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_options` for curl options

Value

A list.

References

http://www.gbif.org/developer/registry#datasets

Examples

```
## Not run:
datasets(limit=5)
datasets(type="occurrence", limit=10)
datasets(uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
datasets(data='contact', uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
datasets(data='metadata', uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
datasets(data='metadata', 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(verbose=TRUE))

## End(Not run)
```

---

dataset_metrics *Get details on a GBIF dataset.*

Description

Get details on a GBIF dataset.

Usage

```
dataset_metrics(uuid, curlopts = list())
```

Arguments

uuid	(character) One or more dataset UUIDs. See examples.
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

References

http://www.gbif.org/developer/registry#datasetMetrics

Examples

```
## Not run:
dataset_metrics(uuid='863e6d6b-f602-4495-ac30-881482b6f799')
dataset_metrics(uuid='66dd0960-2d7d-46ee-a491-87b9adcfe7b1')
dataset_metrics(uuid=c('863e6d6b-f602-4495-ac30-881482b6f799',
   '66dd0960-2d7d-46ee-a491-87b9adcfe7b1'))
dataset_metrics(uuid='66dd0960-2d7d-46ee-a491-87b9adcfe7b1',
  curlopts = list(verbose=TRUE))

## End(Not run)
```

`dataset_search` *Search datasets in GBIF.*

Description

This function does not search occurrence data, only metadata on the datasets that contain occurrence data.

Usage

```
dataset_search(query = NULL, country = NULL, type = NULL,
  keyword = NULL, owningOrg = NULL, publishingOrg = NULL,
  hostingOrg = NULL, publishingCountry = NULL, decade = NULL,
  facet = NULL, facetMincount = NULL, facetMultiselect = NULL,
  limit = 100, start = NULL, pretty = FALSE, return = "all",
  curlopts = list())
```

Arguments

`query`	Query term(s) for full text search. 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*`
`country`	NOT YET IMPLEMENTED. Filters by country as given in isocodes$gbif_name, e.g. `country=CANADA`
`type`	Type of dataset, options include occurrene, metadata, checklist, sampling_event (http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/DatasetType.html)
`keyword`	Keyword to search by. Datasets can be tagged by keywords, which you can search on. The search is done on the merged collection of tags, the dataset keywordCollections and temporalCoverages.
`owningOrg`	Owning organization. DEFUNCT.
`publishingOrg`	Publishing organization. A uuid string. See `organizations`
`hostingOrg`	Hosting organization. A uuid string. See `organizations`
`publishingCountry`	Publishing country. See options at isocodes$gbif_name

decade	Decade, e.g., 1980. 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. Facet by decade to get the break down, e.g. /search?facet=DECADE&facet_only=true (see example below)
facet	A list of facet names used to retrieve the 100 most frequent values for a field. Allowed facets are: datasetKey, highertaxonKey, rank, status, extinct, 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 #, e.g. http://bit.ly/1bMdByP only shows the type value 'ACCEPTED' because the other statuses have counts less than 7,000,000
facetMultiselect	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/19YLXPO still shows all status values even though status is being filtered by `status=ACCEPTED`
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.
pretty	Print informative metadata using `cat`. Not easy to manipulate output though.
return	What to return. One of meta, descriptions, data, facets, or all (Default).
curlopts	list of named curl options passed on to `HttpClient`. see `curl_options` for curl options

Value

A data.frame, list, or message printed to console (using `pretty=TRUE`).

References

http://www.gbif.org/developer/registry#datasetSearch

Examples

```
## Not run:
# Gets all datasets of type "OCCURRENCE".
dataset_search(type="OCCURRENCE", limit = 10)

# Fulltext search for all datasets having the word "amsterdam" somewhere in
# its metadata (title, description, etc).
dataset_search(query="amsterdam", limit = 10)

# Limited search
dataset_search(type="OCCURRENCE", limit=2)
dataset_search(type="OCCURRENCE", limit=2, start=10)

# Return just descriptions
```

```
dataset_search(type="OCCURRENCE", return="descriptions", limit = 10)

# Return metadata in a more human readable way (hard to manipulate though)
dataset_search(type="OCCURRENCE", pretty=TRUE, limit = 10)

# Search by country code. Lookup isocodes first, and use US for United States
isocodes[agrep("UNITED", isocodes$gbif_name),]
dataset_search(country="US", limit = 10)

# Search by decade
dataset_search(decade=1980, limit = 10)

# Faceting
## just facets
dataset_search(facet="decade", facetMincount="10", limit=0)

## data and facets
dataset_search(facet="decade", facetMincount="10", limit=2)

# Some parameters accept many inputs, treated as OR
dataset_search(type = c("metadata", "checklist"))$data
dataset_search(keyword = c("fern", "algae"))$data
dataset_search(publishingOrg = c("e2e717bf-551a-4917-bdc9-4fa0f342c530",
  "90fd6680-349f-11d8-aa2d-b8a03c50a862"))$data
dataset_search(hostingOrg = c("c5f7ef70-e233-11d9-a4d6-b8a03c50a862",
  "c5e4331-7f2f-4a8d-aa56-81ece7014fc8"))$data
dataset_search(publishingCountry = c("DE", "NZ"))$data
dataset_search(decade = c(1910, 1930))$data

## curl options
dataset_search(facet="decade", facetMincount="10", limit=2,
  curlopts = list(verbose=TRUE))

## End(Not run)
```

`dataset_suggest` *Suggest datasets in GBIF.*

Description

Suggest datasets in GBIF.

Usage

```
dataset_suggest(query = NULL, country = NULL, type = NULL,
  subtype = NULL, keyword = NULL, owningOrg = NULL,
  publishingOrg = NULL, hostingOrg = NULL, publishingCountry = NULL,
  decade = NULL, continent = NULL, limit = 100, start = NULL,
  pretty = FALSE, description = FALSE, curlopts = list())
```

Arguments

query	Query term(s) for full text search. 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*
country	NOT YET IMPLEMENTED. Filters by country as given in isocodes$gbif_name, e.g. country=CANADA
type	Type of dataset, options include occurrene, metadata, checklist, sampling_event (http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/DatasetType.html)
subtype	NOT YET IMPLEMENTED. Will allow filtering of datasets by their dataset subtypes, DC or EML.
keyword	Keyword to search by. Datasets can be tagged by keywords, which you can search on. The search is done on the merged collection of tags, the dataset keywordCollections and temporalCoverages.
owningOrg	Owning organization. DEFUNCT.
publishingOrg	Publishing organization. A uuid string. See organizations
hostingOrg	Hosting organization. A uuid string. See organizations
publishingCountry	Publishing country. See options at isocodes$gbif_name
decade	Decade, e.g., 1980. 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. Facet by decade to get the break down, e.g. /search?facet=DECADE&facet_only=true (see example below)
continent	Not yet implemented, but will eventually allow filtering datasets by their continent(s) as given in our Continent enum.
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.
pretty	Print informative metadata using cat. Not easy to manipulate output though.
description	Return descriptions only (TRUE) or all data (FALSE, default)
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

Value

A data.frame, list, or message printed to console (using pretty=TRUE).

References

http://www.gbif.org/developer/registry#datasetSearch

Examples

```
## Not run:
# Suggest datasets of type "OCCURRENCE".
dataset_suggest(query="Amazon", type="OCCURRENCE")

# Suggest datasets tagged with keyword "france".
dataset_suggest(keyword="france")

# Fulltext search for all datasets having the word "amsterdam" somewhere in
# its metadata (title, description, etc).
dataset_suggest(query="amsterdam")

# Limited search
dataset_suggest(type="OCCURRENCE", limit=2)
dataset_suggest(type="OCCURRENCE", limit=2, start=10)

# Return just descriptions
dataset_suggest(type="OCCURRENCE", limit = 5, description=TRUE)

# Return metadata in a more human readable way (hard to manipulate though)
dataset_suggest(type="OCCURRENCE", limit = 5, pretty=TRUE)

# Search by country code. Lookup isocodes first, and use US for United States
isocodes[agrep("UNITED", isocodes$gbif_name),]
dataset_suggest(country="US", limit = 25)

# Search by decade
dataset_suggest(decade=1980, limit = 30)

# Some parameters accept many inputs, treated as OR
dataset_suggest(type = c("metadata", "checklist"))
dataset_suggest(keyword = c("fern", "algae"))
dataset_suggest(publishingOrg = c("e2e717bf-551a-4917-bdc9-4fa0f342c530",
  "90fd6680-349f-11d8-aa2d-b8a03c50a862"))
dataset_suggest(hostingOrg = c("c5f7ef70-e233-11d9-a4d6-b8a03c50a862",
  "c5e4331-7f2f-4a8d-aa56-81ece7014fc8"))
dataset_suggest(publishingCountry = c("DE", "NZ"))
dataset_suggest(decade = c(1910, 1930))

# curl options
dataset_suggest(type="OCCURRENCE", limit = 2, curlopts = list(verbose=TRUE))

## End(Not run)
```

`downloads` *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 pass these to the function call or set them 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, and you won't need to pass them in to the function call. If yo set them in your `.Rprofile` file, they won't be available until you restart the R session.

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 then 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_meta` - Get metadata progress on a single download
- `occ_download_list` - List your downloads
- `occ_download_cancel` - Cancel a download
- `occ_download_get` - Retrieve a download
- `occ_download_import` - Import a download from local file system

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

Description

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

Usage

```
elevation(input = NULL, latitude = NULL, longitude = NULL,
  latlong = NULL, key, curlopts = list())
```

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.
key	(character) Required. An API key. See Details.
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

Details

To get an API key, see instructions at https://developers.google.com/maps/documentation/elevation/#api_key - It should be an easy process. Once you have the key pass it in to the key parameter. You can store the key in your .Rprofile file and read it in via getOption as in the examples below.

Value

A new column named elevation in the supplied data.frame or a vector with elevation of each location in meters.

References

Uses the Google Elevation API at the following link https://developers.google.com/maps/documentation/elevation/start

Examples

```
## Not run:
apikey <- getOption("g_elevation_api")
key <- name_suggest('Puma concolor')$key[1]
dat <- occ_search(taxonKey=key, return='data', limit=300,
  hasCoordinate=TRUE)
head( elevation(dat, key = apikey) )

# Pass in a vector of lat's and a vector of long's
elevation(latitude=dat$decimalLatitude, longitude=dat$decimalLongitude,
  key = apikey)

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

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

## End(Not run)
```

enumeration *Enumerations.*

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

```
enumeration(x = NULL, curlopts = list())

enumeration_country(curlopts = list())
```

Arguments

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

Value

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

Examples

```
## 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)
```

gbifmap *Make a map to visualize GBIF occurrence data.*

Description

Make a map to visualize GBIF occurrence data.

Usage

```
gbifmap(input = NULL, mapdatabase = "world", region = ".",
  geom = geom_point, jitter = NULL, customize = NULL)
```

Arguments

input	Either a single data.frame or a list of data.frame's (e.g., from different speies). The data.frame has to have, in addition to any other columns, columns named exactly "decimalLatitude" and "decimalLongitude".
mapdatabase	The map database to use in mapping. What you choose here determines what you can choose in the region parameter. One of: county, state, usa, world, world2, france, italy, or nz.
region	The region of the world to map. Run `sort(unique(ggplot2::map_data("world")$region))` to see region names for the world database layer, or `sort(unique(ggplot2::map_data("state")$regi` for the state layer.
geom	The geom to use, one of geom_point or geom_jitter. Don't quote them.
jitter	If you use jitterposition, the amount by which to jitter points in width, height, or both.
customize	Further arguments passed on to ggplot.

Details

gbifmap takes care of cleaning up the data.frame (removing NA's, etc.) returned from rgbif functions, and creating the map. This function gives a simple map of your data. You can look at the code behing the function itself if you want to build on it to make a map according to your specfications.

Note that this function removes values that are impossible on the globe, and those rows that have both lat and long as NA or zeros.

Value

Map (using ggplot2 package) of points on a map or tiles on a map.

Examples

```
## Not run:
# Make a map of Puma concolor occurrences
key <- name_backbone(name='Puma concolor')$speciesKey
dat <- occ_search(taxonKey=key, return='data', limit=100)
gbifmap(dat)

# Plot more Puma concolor occurrences
dat <- occ_search(taxonKey=key, return='data', limit=1200)
nrow(dat)
gbifmap(dat)

# Jitter positions, compare the two
library("ggplot2")
gbifmap(dat)
gbifmap(dat, geom = geom_jitter, jitter = position_jitter(1, 6))

# many species
splist <- c('Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa')
keys <- vapply(splist, function(x) name_suggest(x)$key[1], numeric(1),
  USE.NAMES=FALSE)
dat <- occ_data(keys, limit = 50)
library("data.table")
dd <- rbindlist(lapply(dat, function(z) z$data), fill = TRUE,
  use.names = TRUE)
gbifmap(dd)

## 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 charactere, 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=38.4, miny=-125.0, maxx=40.9, maxy=-121.8)

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

```
## Not run:
res1 <- occ_search(taxonKey=3119195, limit=2)
(xx <- gbif_citation(res1))

res2 <- occ_search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a',
 return='data', limit=20)
(xx <- gbif_citation(res2))

# if no datasetKey field included, we attempt to identify the dataset
## key field included - still works
res3 <- occ_search(taxonKey=3119195, fields=c('name','basisOfRecord','key'),
  limit=20)
(xx <- gbif_citation(res3))
## key field not included - errors
# res3 <- occ_search(taxonKey=3119195, fields=c('name','basisOfRecord','
#    protocol'), limit=20)
# (xx <- gbif_citation(res3))

# character class inputs
## pass in a dataset key
gbif_citation(x='0ec3229f-2b53-484e-817a-de8ceb1fce2b')
## pass in an occurrence key
gbif_citation(x='766766824')

# pass in an occurrence key as a numeric
gbif_citation(x=766766824)

# Downloads
## only works with output from occ_download_get for now
d1 <- occ_download_get("0000066-140928181241064", overwrite = TRUE)
gbif_citation(d1)

## End(Not run)
```

`gbif_issues` *Table of GBIF issues, with codes used in data output, full issue name, and descriptions.*

Description

Table has the following fields:

Usage

```
gbif_issues()
```

Details

- code. Code for issue, making viewing data easier.
- issue. Full name of the issue.
- description. Description of the issue.

Source

http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/OccurrenceIssue.html

---

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)
```

gbif_oai *GBIF registry data via OAI-PMH*

Description

GBIF registry data via OAI-PMH

Usage

```
gbif_oai_identify(...)

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

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

gbif_oai_list_metadataformats(id = NULL, ...)

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

gbif_oai_get_records(ids, prefix = "oai_dc", as = "parsed", ...)
```

Arguments

`...`	Curl options passed on to `httr::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

Value

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

Examples

```
## 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-8ec1-8c434fa9e6eb")

gbif_oai_list_sets()
gbif_oai_list_sets(as = "list")

gbif_oai_get_records("9c4e36c1-d3f9-49ce-8ec1-8c434fa9e6eb")
ids <- c("9c4e36c1-d3f9-49ce-8ec1-8c434fa9e6eb",
         "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(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

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

res <- occ_search(scientificName = "Aves", mediaType = 'StillImage',
  return = "media", 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_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

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

Examples

```
## 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', curlopts = list(verbose=TRUE))

## End(Not run)
```

isocodes *Table of country two character ISO codes, and GBIF names*

Description

- code. Two character ISO country code.
- name. Name of country.
- gbif_name. Name of country used by GBIF - this is the name you want to use when searching by country in this package.

many-values *Many value inputs to some parameters*

Description

Many value inputs to some parameters

Details

There are some differences in how functions across **rgbif** behave with respect to many values given to a single parameter (let's call it `foo`).

The following functions originally only iterated over many values passed to `foo` as a vector (e.g., `foo = c(1, 2)`) with completely separate HTTP requests. But now these functions also support passing in many values to the same HTTP request (e.g., `foo = "1;2"`). This is a bit awkward, but means that we don't break existing code.

- `occ_search()`
- `occ_data()`

The following functions, unlike those above, only support passing in many values to the same HTTP request, which is done like `foo = c("1", "2")`.

- `dataset_search()`
- `dataset_suggest()`
- `name_lookup()`
- `name_suggest()`
- `name_usage()`

Last, some parameters in the functions above don't accept more than one, and some functions don't have any parameters that accept more than one value (i.e., none of those listed above).

Each function that has at least some parameters that accept many values also has documentation on this issue.

name_backbone *Lookup names in the GBIF backbone taxonomy.*

Description

Lookup names in the GBIF backbone taxonomy.

Usage

```
name_backbone(name, rank = NULL, kingdom = NULL, phylum = NULL,
  class = NULL, order = NULL, family = NULL, genus = NULL,
  strict = FALSE, verbose = FALSE, start = NULL, limit = 100,
  curlopts = list())
```

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)
genus	(character) If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
strict	(logical) If TRUE it (fuzzy) matches only the given name, but never a taxon in the upper classification (optional)
verbose	(logical) If TRUE show alternative matches considered which had been rejected.
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.
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

Details

If you don't get a match GBIF gives back a list of length 3 with slots synonym, confidence, and matchType='NONE'.

Value

A list for a single taxon with many slots (with verbose=FALSE)

- default), or a list of length two, first element for the suggested taxon match, and a data.frame with alternative name suggestions resulting from fuzzy matching (with verbose=TRUE).

References

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

Examples

```
## 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
name_backbone(name='Helianthus annuus', kingdom='plants', verbose=TRUE)

# Strictness
name_backbone(name='Poa', kingdom='plants', verbose=TRUE, strict=FALSE)
name_backbone(name='Helianthus annuus', kingdom='plants', verbose=TRUE,
  strict=TRUE)

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

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

## End(Not run)
```

---

name_lookup *Lookup names in all taxonomies in GBIF.*

Description

Lookup names in all taxonomies in GBIF.

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

```
name_lookup(query = NULL, rank = NULL, higherTaxonKey = NULL,
  status = NULL, isExtinct = NULL, habitat = NULL, nameType = NULL,
  datasetKey = NULL, nomenclaturalStatus = NULL, limit = 100,
  start = NULL, facet = NULL, facetMincount = NULL,
  facetMultiselect = NULL, type = NULL, hl = NULL, verbose = FALSE,
  return = "all", 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, INFRASUBSPECIFIC_NAME, KINGDOM, ORDER, PHYLUM, SECTION, SERIES, SPECIES, STRAIN, SUBCLASS, SUBFAMILY, SUBFORM, SUBGENUS, SUBKINGDOM, SUBORDER, SUBPHYLUM, SUBSECTION, SUBSERIES, SUBSPECIES, SUBTRIBE, SUBVARIETY, SUPERCLASS, SUPERFAMILY, SUPERORDER, SUPERPHYLUM, SUPRAGENERIC_NAME, TRIBE, UNRANKED, VARIETY
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. • HETEROTYPIC_SYNONYM More specific subclass of SYNONYM. • HOMOTYPIC_SYNONYM More specific subclass of SYNONYM. • INTERMEDIATE_RANK_SYNONYM Used in nub only. • MISAPPLIED 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)

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

`limit` Number of records to return. Maximum: 1000.

`start` Record number to start at.

`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 #, e.g. http://bit.ly/1bMdByP only shows the type value 'ACCEPTED' because the other statuses have counts less than 7,000,000

`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/19YLXPO still shows all status values even though status is being filtered by `status=ACCEPTED`

`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.

`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` One of data, meta, facets, names, or all. If data, a data.frame with the data. facets returns the facets, if `facets=TRUE`, or empy list if `facets=FALSE`. meta returns the metadata for the entire call. names returns the vernacular (common) names for each taxon. all gives all data back in a list. Each element is NULL if there is no contents in that element. hierarchies and names slots are named by the GBIF key, which matches the first column of the data.frame in the data slot. So if you wanted to combine those somehow, you could easily do so using the key.

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

Value

A list of length three. The first element is metadata. The second is either a data.frame (`verbose=FALSE`, default) or a list (`verbose=TRUE`), and the third element is the facet data.

Repeat parmeter inputs

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

- **rank**
- **higherTaxonKey**
- **status**
- **habitat**
- **nameType**
- **datasetKey**

see also many-values

References

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

Examples

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

# Paging
name_lookup(query='mammalia', limit=1)
name_lookup(query='mammalia', limit=1, start=2)

# large requests, use start parameter
first <- name_lookup(query='mammalia', limit=1000)
second <- name_lookup(query='mammalia', limit=1000, start=1000)
tail(first$data)
head(second$data)
first$meta
second$meta

# 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, returning just data
name_lookup(query='Cnaemidophorus', rank="genus", return="data")

# Just metadata
name_lookup(query='Cnaemidophorus', rank="genus", return="meta")

# Just hierarchies
name_lookup(query='Cnaemidophorus', rank="genus", return="hierarchy")
```

```
# Just vernacular (common) names
name_lookup(query='Cnaemidophorus', rank="genus", return="names")

# 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, return='data')
## 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
name_lookup(datasetKey='3f8a1297-3259-4700-91fc-acc4170b27ce')

# 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"))

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

## End(Not run)
```

name_suggest *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.*

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

```
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)
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, subform, subgenus, subkingdom, suborder, subphylum, subsection, subseries, subspecies, subtribe, subvariety, superclass, superfamily, 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.
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

Value

A data.frame with fields selected by fields arg.

Repeat parmeter inputs

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

- **rank**
- **datasetKey**

see also many-values

References

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

Examples

```
## 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-bbc5-522bfb90d847",
  "d7c60346-44b6-400d-ba27-8d3fbeffc8a5"))

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

## End(Not run)
```

---

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

Description

Lookup details for specific names in all taxonomies in GBIF.

Usage

```
name_usage(key = NULL, name = NULL, data = "all", language = NULL,
  datasetKey = NULL, uuid = NULL, sourceId = NULL, rank = NULL,
  shortname = NULL, start = NULL, limit = 100, return = "all",
  curlopts = list())
```

Arguments

key	(numeric) A GBIF key for a taxon
name	(character) Filters by a case insensitive, canonical namestring, e.g. 'Puma concolor'

data	(character) Specify an option to select what data is returned. See Description below.
language	(character) Language, default is english
datasetKey	(character) Filters by the dataset's key (a uuid)
uuid	(character) A uuid for a dataset. Should give exact same results as datasetKey.
sourceId	(numeric) Filters by the source identifier. Not used right now.
rank	(character) Taxonomic rank. Filters by taxonomic rank as one of: CLASS, CULTIVAR, CULTIVAR_GROUP, DOMAIN, FAMILY, FORM, GENUS, INFORMAL, INFRAGENERIC_NAME, INFRAORDER, INFRASPECIFIC_NAME, INFRASUBSPECIFIC_NAME, KINGDOM, ORDER, PHYLUM, SECTION, SERIES, SPECIES, STRAIN, SUBCLASS, SUBFAMILY, SUBFORM, SUBGENUS, SUBKINGDOM, SUBORDER, SUBPHYLUM, SUBSECTION, SUBSERIES, SUBSPECIES, SUBTRIBE, SUBVARIETY, SUPERCLASS, SUPERFAMILY, SUPERORDER, SUPERPHYLUM, SUPRAGENERIC_NAME, TRIBE, UNRANKED, VARIETY
shortname	(character) A short name..need more info on this?
start	Record number to start at. Default: 0. Use in combination with `limit` to page through results.
limit	Number of records to return. Default: 100. Maximum: 1000.
return	One of data, meta, or all. If data, a data.frame with the data. meta returns the metadata for the entire call. all gives all data back in a list.
curlopts	list of named curl options passed on to `HttpClient`. see `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'

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.

Value

A list of length two. The first element is metadata. The second is a data.frame

Repeat parmeter inputs

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

- **rank**
- **datasetKey**
- **uuid**
- **name**
- **langugae**

see also many-values

References

<http://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")

# 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')

# Some parameters accept many inputs, treated as OR
name_usage(rank = c("family", "genus"))
name_usage(datasetKey = c("73605f3a-af85-4ade-bbc5-522bfb90d847",
```

```
  "d7c60346-44b6-400d-ba27-8d3fbeffc8a5"))
name_usage(uuid = c("73605f3a-af85-4ade-bbc5-522bfb90d847",
  "d7c60346-44b6-400d-ba27-8d3fbeffc8a5"))
name_usage(name = c("Puma", "Quercus"))
name_usage(language = c("spanish", "german"))

# Pass on curl options
name_usage(name='Puma concolor', limit=300, curlopts = list(verbose=TRUE))

## End(Not run)
```

`networks` *Networks metadata.*

Description

Networks metadata.

Usage

```
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_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

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

Examples

```
## Not run:
networks(limit=5)
networks(uuid='7ddd1f14-a2b0-4838-95b0-785846f656f3')
uuids <- c('7ddd1f14-a2b0-4838-95b0-785846f656f3',
  '07b013b4-a2da-47a1-a8ef-df685912fbd6')
lapply(uuids, function(x) networks(uuid = x))
networks(data='endpoint', uuid='16ab5405-6c94-4189-ac71-16ca3b753df7')

# curl options
networks(limit=5, curlopts = list(verbose=TRUE))

## End(Not run)
```

---

nodes *Nodes metadata.*

Description

Nodes metadata.

Usage

```
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_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

http://www.gbif.org/developer/registry#nodes

Examples

```
## 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-1cba-4633-90b7-e36e5e97aba8",
  "7a17efec-0a6a-424c-b743-f715852c3c1f",
  "b797ce0f-47e6-4231-b048-6b62ca3b0f55",
  "1193638d-32d1-43f0-a855-8727c94299d8",
  "d3499f89-5bc0-4454-8cdb-60bead228a6d",
  "cdc9736d-5ff7-4ece-9959-3c744360cdb3",
  "a8b16421-d80b-4ef3-8f22-098b01a89255",
  "8df8d012-8e64-4c8a-886e-521a3bdfa623",
  "b35cf8f1-748d-467a-adca-4f9170f20a4e",
  "03e816b3-8f58-49ae-bc12-4e18b358d6d9",
  "073d1223-70b1-4433-bb21-dd70afe3053b",
  "07dfe2f9-5116-4922-9a8a-3e0912276a72",
  "086f5148-c0a8-469b-84cc-cce5342f9242",
  "0909d601-bda2-42df-9e63-a6d51847ebce",
  "0e0181bf-9c78-4676-bdc3-54765e661bb8",
  "109aea14-c252-4a85-96e2-f5f4d5d088f4",
  "169eb292-376b-4cc6-8e31-9c2c432de0ad",
  "1e789bc9-79fc-4e60-a49e-89dfc45a7188",
  "1f94b3ca-9345-4d65-afe2-4bace93aa0fe")

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(verbose=TRUE))

## End(Not run)
```

---

occ_count *Get number of occurrence records.*

Description

Get number of occurrence records.

Usage

```
occ_count(taxonKey = NULL, georeferenced = NULL, basisOfRecord = NULL,
  datasetKey = NULL, date = NULL, typeStatus = NULL,
```

```
catalogNumber = NULL, country = NULL, hostCountry = NULL, year = NULL,
from = 2000, to = 2012, type = "count", publishingCountry = "US",
nubKey = NULL, protocol = NULL, curlopts = list())
```

Arguments

taxonKey	Species key
georeferenced	Return only occurence records with lat/long data (TRUE) or all records (FALSE, default).
basisOfRecord	Basis of record
datasetKey	Dataset key
date	Collection date
typeStatus	A type status. See typestatus() dataset for options
catalogNumber	Catalog number. PARAMETER GONE.
country	Country data was collected in, two letter abbreviation. See http://countrycode.org/ for abbreviations.
hostCountry	Country that hosted the data. PARAMETER GONE.
year	Year data were collected in
from	Year to start at
to	Year to end at
type	One of count (default), schema, basis_of_record, countries, or year.
publishingCountry	Publishing country, two letter ISO country code
nubKey	Species key. PARAMETER NAME CHANGED TO taxonKey.
protocol	Protocol. E.g., 'DWC_ARCHIVE'
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

Details

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

Value

A single numeric value, or a list of numerics.

Supported dimensions

That is, there are only a certain set of supported query parameter combinations that GBIF allows on this API route. They can be found with the call occ_count(type='schema'). They are also presented below:

- basisOfRecord

- basisOfRecord, country
- basisOfRecord, country, isGeoreferenced
- basisOfRecord, country, isGeoreferenced, taxonKey
- basisOfRecord, country, taxonKey
- basisOfRecord, datasetKey
- basisOfRecord, datasetKey, isGeoreferenced
- basisOfRecord, datasetKey, isGeoreferenced, taxonKey
- basisOfRecord, datasetKey, taxonKey
- basisOfRecord, isGeoreferenced, taxonKey
- basisOfRecord, isGeoreferenced, publishingCountry
- basisOfRecord, isGeoreferenced, publishingCountry, taxonKey
- basisOfRecord, publishingCountry
- basisOfRecord, publishingCountry, taxonKey
- basisOfRecord, taxonKey
- country
- country, datasetKey, isGeoreferenced
- country, isGeoreferenced
- country, isGeoreferenced, publishingCountry
- country, isGeoreferenced, taxonKey
- country, publishingCountry
- country, taxonKey
- country, typeStatus
- datasetKey
- datasetKey, isGeoreferenced
- datasetKey, isGeoreferenced, taxonKey
- datasetKey, issue
- datasetKey, taxonKey
- datasetKey, typeStatus
- isGeoreferenced
- isGeoreferenced, publishingCountry
- isGeoreferenced, publishingCountry, taxonKey
- isGeoreferenced, taxonKey
- issue
- publishingCountry
- publishingCountry, taxonKey
- publishingCountry, typeStatus
- taxonKey
- taxonKey, typeStatus
- typeStatus
- protocol
- year

References

http://www.gbif.org/developer/occurrence#metrics

Examples

```
## Not run:
occ_count(basisOfRecord='OBSERVATION')
occ_count(georeferenced=TRUE)
occ_count(country='DE')
occ_count(country='CA', georeferenced=TRUE, basisOfRecord='OBSERVATION')
occ_count(datasetKey='9e7ea106-0bf8-4087-bb61-dfe4f29e0f17')
occ_count(year=2012)
occ_count(taxonKey=2435099)
occ_count(taxonKey=2435099, georeferenced=TRUE)
occ_count(protocol='DWC_ARCHIVE')

# Just schema
occ_count(type='schema')

# Counts by basisOfRecord types
occ_count(type='basisOfRecord')

# Counts by basisOfRecord types
occ_count(typeStatus='ALLOTYPE')
occ_count(typeStatus='HOLOTYPE')

# Counts by countries. publishingCountry must be supplied (default to US)
occ_count(type='countries')

# Counts by year. from and to years have to be supplied, default to 2000
# and 2012
occ_count(type='year', from=2000, to=2012)

# Counts by publishingCountry, must supply a country (default to US)
occ_count(type='publishingCountry')
occ_count(type='publishingCountry', country='BZ')

# Pass on curl options
occ_count(type='year', from=2000, to=2012, curlopts = list(verbose = TRUE))

## End(Not run)
```

---

occ_data *Search for GBIF occurrences - simplified for speed*

Description

Search for GBIF occurrences - simplified for speed

Usage

```
occ_data(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, 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, establishmentMeans = NULL,
  protocol = NULL, license = NULL, organismId = NULL,
  publishingOrg = NULL, stateProvince = NULL, waterBody = NULL,
  locality = NULL, limit = 500, start = 0, spellCheck = NULL,
  curlopts = list())
```

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 taxononKey=212 (i.e. /occurrence/search?taxonKey=212) will match all birds, no matter which species. You can pass many keys by passing occ_search in a call to an lapply-family function (see last example below).
scientificName	A scientific name from the GBIF backbone. All included and synonym taxa are included in the search.
country	The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded. See here <http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2>
publishingCountry	The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded.
hasCoordinate	(logical) Return only occurence records with lat/long data (TRUE) or all records (FALSE, default).
typeStatus	Type status of the specimen. One of many options. See ?typestatus
recordNumber	Number recorded by collector of the data, different from GBIF record number. See <http://rs.tdwg.org/dwc/terms/#recordNumber> for more info
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	Continent. One of africa, antarctica, asia, europe, north_america (North America includes the Caribbean and reachies down and includes Panama), oceania, or south_america

`geometry` Searches for occurrences inside a polygon described in Well Known Text (WKT) format. A WKT shape written as either POINT, LINESTRING, LINEARRING POLYGON, or MULTIPOLYGON. Example of a polygon: POLYGON((30.1 10.1, 20, 20 40, 40 40, 30.1 10.1)) would be queried as <http://bit.ly/1BzNwDq>. See also the section **WKT** below.

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

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

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

`recordedBy` The person who recorded the occurrence.

`basisOfRecord` Basis of record, as defined in our BasisOfRecord enum here <http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/BasisOfRecord.html> Acceptable values are:

- FOSSIL_SPECIMEN An occurrence record describing a fossilized specimen.
- HUMAN_OBSERVATION An occurrence record describing an observation made by one or more people.
- LITERATURE An occurrence record based on literature alone.
- LIVING_SPECIMEN An occurrence record describing a living specimen, e.g.
- MACHINE_OBSERVATION An occurrence record describing an observation made by a machine.
- OBSERVATION An occurrence record describing an observation.
- PRESERVED_SPECIMEN An occurrence record describing a preserved specimen.
- UNKNOWN Unknown basis for the record.

`datasetKey` The occurrence dataset key (a uuid)

`eventDate` Occurrence date 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)

`catalogNumber` 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 WGS 84. 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 WGS 84. 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. Not guaranteed to be que.

collectionCode — 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. See Details. Issues passed to this parameter filter results by the issue.

search — Query terms. The value for this parameter can be a simple word or a phrase.

mediaType — Media type. Default is NULL, so no filtering on mediatype. Options: NULL, 'MovingImage', 'Sound', and '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.

establishmentMeans — (character) EstablishmentMeans, possible values include: INTRODUCED, INVASIVE, MANAGED, NATIVE, NATURALISED, UNCERTAIN

protocol — (character) Protocol or mechanism used to provide the occurrence record. See Details for possible values

license — (character) The type license applied to the dataset or record. Possible values: CC0_1_0, CC_BY_4_0, CC_BY_NC_4_0, UNSPECIFIED, and UNSUPPORTED

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 200,000, which is calculated as the limit+start, so start=199,000 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
spellCheck	(logical) If TRUE ask GBIF to check your spelling of the value passed to the search parameter. IMPORTANT: This only checks the input to the search parameter, and no others. Default: FALSE
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options
...	additional facet parameters

Details

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

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)

References

http://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')$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{scientifcName}
## 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(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)

# Pass in curl options for extra fun
occ_data(taxonKey=2433407, limit=20, curlopts=list(verbose=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)$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, 10 20, 20 40, 40 40, 30.1 10.1))',
  limit=20)
### multipolygon
wkt <- 'MULTIPOLYGON(((-123 38, -123 43, -116 43, -116 38, -123 38)),
   ((-97 41, -97 45, -93 45, -93 41, -97 41)))'
occ_data(geometry = gsub("\n\\s+", "", wkt), limit = 20)
### polygon and taxonkey
key <- name_suggest(q='Aesculus hippocastanum')$key[1]
occ_data(taxonKey=key,
 geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 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 seaerch on many geometry objects
### separate requests: use a vector of strings
wkts <-
c('POLYGON((-102.2 46.0,-93.9 46.0,-93.9 43.7,-102.2 43.7,-102.2 46.0))',
'POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 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
```

```
wkt <- "POLYGON((13.26349675655365 52.53991761181831,18.36115300655365 54.11445544219924,
21.87677800655365 53.80418956368524,24.68927800655365 54.217364774722455,28.20490300655365
54.320018299365124,30.49005925655365 52.85948216284084,34.70880925655365 52.753220564427814,
35.93927800655365 50.46131871049754,39.63068425655365 49.55761261299145,40.86115300655365
46.381388009130845,34.00568425655365 45.279102926537,33.30255925655365 48.636868465271846,
30.13849675655365 49.78513301801265,28.38068425655365 47.2236377039631,29.78693425655365
44.6572866068524,27.67755925655365 42.62220075124676,23.10724675655365 43.77542058000212,
24.51349675655365 47.10412345120368,26.79865300655365 49.55761261299145,23.98615300655365
52.00209943876426,23.63459050655365 49.44345313705238,19.41584050655365 47.580567827212114,
19.59162175655365 44.90682206053508,20.11896550655365 42.36297154876359,22.93146550655365
40.651849782081555,25.56818425655365 39.98171166226459,29.61115300655365 40.78507856230178,
32.95099675655365 40.38459278067577,32.95099675655365 37.37491910393631,26.27130925655365
33.65619609886799,22.05255925655365 36.814081996401605,18.71271550655365 36.1072176729021,
18.53693425655365 39.16878677351903,15.37287175655365 38.346355762190846,15.19709050655365
41.578843777436326,12.56037175655365 41.050735748143424,12.56037175655365 44.02872991212046,
15.19709050655365 45.52594200494078,16.42755925655365 48.05271546733352,17.48224675655365
48.86865641518059,10.62677800655365 47.817178329053135,9.57209050655365 44.154980365192,
8.16584050655365 40.51835445724746,6.05646550655365 36.53210972067291,0.9588092565536499
31.583640057148145,-5.54509699344635 35.68001485298146,-6.77556574344635 40.51835445724746,
-9.41228449344635 38.346355762190846,-12.40056574344635 35.10683619158607,-15.74040949344635
38.07010978950028,-14.68572199344635 41.31532459432774,-11.69744074344635 43.64836179231387,
-8.88494074344635 42.88035509418534,-4.31462824344635 43.52103366008421,-8.35759699344635
47.2236377039631,-8.18181574344635 50.12441989397795,-5.01775324344635 49.55761261299145,
-2.73259699344635 46.25998980446569,-1.67790949344635 44.154980365192,-1.32634699344635
39.30493590580802,2.18927800655365 41.44721797271696,4.47443425655365 43.26556960420879,
2.18927800655365 46.7439668697322,1.83771550655365 50.3492841273576,6.93537175655365
49.671505849335254,5.00177800655365 52.32557322466785,7.81427800655365 51.67627099802223,
7.81427800655365 54.5245591562317,10.97834050655365 51.89375191441792,10.97834050655365
55.43241335888528,13.26349675655365 52.53991761181831))"
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
res <- occ_data(geometry = wkt, geom_big = "bbox")
library("rgeos")
library("sp")
wktsp <- readWKT(wkt)
plot(wktsp)
coordinates(res$data) <- ~decimalLongitude+decimalLatitude
points(res$data)

#### 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)
isocodes[grep("France", isocodes$name),"code"]
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)

# 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)
## spell check - only works with the `search` parameter
### spelled correctly - same result as above call
occ_data(search = "kingfisher", limit=20, spellCheck = TRUE)
### spelled incorrectly - stops with suggested spelling
```

```
# occ_data(search = "kajsdkla", limit=20, spellCheck = TRUE)
### spelled incorrectly - stops with many suggested spellings
###   and number of results for each
# occ_data(search = "helir", limit=20, spellCheck = TRUE)

# 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)

# 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='84c0e1a0-f762-11e1-a439-00145eb45e9a',
   issue=c('TAXON_MATCH_NONE','TAXON_MATCH_HIGHERRANK'), limit = 20)

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

## 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

```
occ_download(..., body = NULL, type = "and",
  user = getOption("gbif_user"), pwd = getOption("gbif_pwd"),
  email = getOption("gbif_email"), curlopts = list())
```

Arguments

...	One or more of query arguments to kick of a download job. If you use this, don't use body parameter. See Details.

body	if you prefer to pass in the payload yourself, use this parameter. if use this, don't ass anythig 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	(charcter) One of equals (=), and (&), or (|), lessThan (<), lessThanOrEquals (<=), greaterThan (>), greaterThanOrEquals (>=), in, within, not (!), like
user	(character) User name within GBIF's website. Required. Set in your `.Rprofile` file with the option `gbif_user`
pwd	(character) User password within GBIF's website. Required. Set in your `.Rprofile` file with the option `gbif_pwd`
email	(character) Email address to recieve download notice done email. Required. Set in your `.Rprofile` file with the option `gbif_email`
curlopts	list of named curl options passed on to `HttpClient`. see `curl_options` for curl options

Details

Argument passed have to be passed as character (e.g., 'country = US'), with a space between key ('country'), operator ('='), and value ('US'). See the `type` parameter for possible options for the operator. This character string is parsed internally.

The value can be comma separated, in which case we'll turn that into a predicate combined with the OR operator, for example, `"taxonKey = 2480946,5229208"` will turn into

```
'{
   "type": "or",
   "predicates": [
     {
      "type": "equals",
      "key": "TAXON_KEY",
      "value": "2480946"
     },
     {
      "type": "equals",
      "key": "TAXON_KEY",
      "value": "5229208"
     }
   ]
}'
```

Acceptable arguments to ... are:

- taxonKey = 'TAXON_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'
- continent = 'CONTINENT'
- geometry = 'GEOMETRY'
- basisOfRecord = 'BASIS_OF_RECORD'
- datasetKey = 'DATASET_KEY'
- eventDate = 'EVENT_DATE'
- catalogNumber = 'CATALOG_NUMBER'
- 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'

References

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

Examples

```
## Not run:
# occ_download("basisOfRecord = LITERATURE")
# occ_download('taxonKey = 3119195')
# occ_download('decimalLatitude > 50')
# occ_download('elevation >= 9000')
# occ_download('decimalLatitude >= 65')
# occ_download("country = US")
# occ_download("institutionCode = TLMF")
# occ_download("catalogNumber = Bird.27847588")

# res <- occ_download('taxonKey = 7264332', 'hasCoordinate = TRUE')

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

```
# Multiple queries
# occ_download('decimalLatitude >= 65', 'decimalLatitude <= -65', type="or")
# gg <- occ_download('depth = 80', 'taxonKey = 2343454', type="or")

# complex example with many predicates
# shows example of how to do date ranges for both year and month
# res <- occ_download(
#  "taxonKey = 2480946,5229208",
#  "basisOfRecord = HUMAN_OBSERVATION,OBSERVATION,MACHINE_OBSERVATION",
#  "country = US",
#  "hasCoordinate = true",
#  "hasGeospatialIssue = false",
#  "year >= 1999",
#  "year <= 2011",
#  "month >= 3",
#  "month <= 8"
# )

# Using body parameter - pass in your own complete query
## as JSON
query1 <- '{"creator":"sckott",
  "notification_address":["myrmecocystus@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(verbose=TRUE))

## as a list
library(jsonlite)
query <- list(
  creator = unbox("sckott"),
  notification_address = "myrmecocystus@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(verbose = TRUE))

## End(Not run)
```

`occ_download_cancel` *Cancel a download cretion process.*

Description

Cancel a download cretion process.

Usage

```
occ_download_cancel(key, user = getOption("gbif_user"),
  pwd = getOption("gbif_pwd"), curlopts = list())

occ_download_cancel_staged()
```

Arguments

key	A key generated from a request, like that from occ_download. Required.
user	(character) User name within GBIF's website. Required.
pwd	(character) User password within GBIF's website. Required.
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

Details

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

Examples

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

## End(Not run)
```

---

occ_download_get *Get a download from GBIF.*

Description

Get a download from GBIF.

Usage

```
occ_download_get(key, path = ".", overwrite = FALSE, curlopts = list())
```

Arguments

key	A key generated from a request, like that from occ_download
path	Path to write zip file to. Default: ".", with a .zip appended to the end.
overwrite	Will only overwrite existing path if TRUE.
curlopts	list of named curl options passed on to HttpClient. see curl_options for curl options

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.

Examples

```
## 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

```
occ_download_import(x = NULL, key = NULL, path = ".", ...)

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
`...`	parameters passed on to `data.table::fread()`

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)

Examples

```
## 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")

## End(Not run)
```

`occ_download_list` *Lists the downloads created by a user.*

Description

Lists the downloads created by a user.

Usage

```
occ_download_list(user = getOption("gbif_user"),
  pwd = getOption("gbif_pwd"), limit = 20, start = 0, curlopts = list())
```

Arguments

`user`	A user name, look at option "gbif_user" first
`pwd`	Your password, look at option "gbif_pwd" first
`limit`	Number of records to return. Default: 20
`start`	Record number to start at. Default: 0
`curlopts`	list of named curl options passed on to `HttpClient`. see `curl_options` for curl options

Examples

```
## 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

```
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_options` for curl options

Examples

```
## Not run:
occ_download_meta(key="0003983-140910143529206")
occ_download_meta("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_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

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

# facetMincount - minimum number of records to be included
#   in the faceting results
occ_facet(facet = "country", facetMincount = 30000000L)
occ_facet(facet = c("country", "basisOfRecord"))

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

# paging
## limit
occ_facet(facet = "country", country.facetLimit = 3)
## offset
occ_facet(facet = "country", country.facetLimit = 3,
  country.facetOffset = 3)

# Pass on curl options
occ_facet(facet = "country", country.facetLimit = 3,
  curlopts = list(verbose = TRUE))

## End(Not run)
```

occ_fields *Vector of fields in the output for the function* occ_search()

Description

These fields can be specified in the fields parameer in the occ_search() function.

occ_get *Get data for specific GBIF occurrences.*

Description

Get data for specific GBIF occurrences.

Usage

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

Arguments

key	Occurrence key
return	One of data, hier, meta, or all. If 'data', a data.frame with the data. 'hier' returns the classifications in a list for each record. meta returns the metadata for the entire call. 'all' gives all data back in a list. Ignored if verbatim=TRUE.
verbatim	Return verbatim object (TRUE) or cleaned up object (FALSE, default).
fields	(character) Default ('minimal') will return just taxon name, key, latitude, and longitute. '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_options for curl options

Value

A data.frame or list of data.frame's.

References

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

Examples

```
## Not run:
occ_get(key=766766824, return='data')
occ_get(key=766766824, 'hier')
occ_get(key=766766824, 'all')

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

# Verbatim data
occ_get(key=766766824, verbatim=TRUE)
occ_get(key=766766824, fields='all', verbatim=TRUE)
occ_get(key=766766824, fields=c('scientificName', 'lastCrawled', 'county'),
  verbatim=TRUE)
occ_get(key=c(766766824, 620594291, 766420684), verbatim=TRUE)
occ_get(key=c(766766824, 620594291, 766420684), fields='all', verbatim=TRUE)
occ_get(key=c(766766824, 620594291, 766420684),
   fields=c('scientificName', 'decimalLatitude', 'basisOfRecord'),
   verbatim=TRUE)

# Pass in curl options
occ_get(key=766766824, curlopts = list(verbose=TRUE))

## End(Not run)
```

`occ_issues` *Parse and examine further GBIF issues on a dataset.*

Description

Parse and examine further GBIF issues on a dataset.

Usage

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

Arguments

`.data`	Output from a call to `occ_search()`, but only if `return="all"`, or `return="data"`, otherwise function stops with error
`...`	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. • split_expand Split into new columns, and expand issue names. • expand Expand issue abbreviated codes into descriptive 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 remove from the `data` element.

References

http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/OccurrenceIssue.html

Examples

```
## Not run:
## 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'), ]

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

# Parsing output by issue
(res <- occ_search(geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))', limit = 50))

## or parse issues in various ways
### inlude only rows with gass84 issue
gg <- res %>% occ_issues(gass84)
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(-cdround, -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(-cudc, mutate = "split_expand")

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

## End(Not run)
```

`occ_issues_lookup` *Lookup occurrence issue definitions and short codes*

Description

Lookup occurrence issue definitions and short codes

Usage

```
occ_issues_lookup(issue = NULL, code = NULL)
```

Arguments

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

Examples

```
occ_issues_lookup(issue = 'CONTINENT_COUNTRY_MISMATCH')
occ_issues_lookup(issue = 'MULTIMEDIA_DATE_INVALID')
occ_issues_lookup(issue = 'ZERO_COORDINATE')
occ_issues_lookup(code = 'cdiv')
```

`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

```
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_options for curl options

References

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

Examples

```
## 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

```
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, 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, establishmentMeans = NULL,
  protocol = NULL, license = NULL, organismId = NULL,
  publishingOrg = NULL, stateProvince = NULL, waterBody = NULL,
  locality = NULL, limit = 500, start = 0, fields = "all",
  return = "all", spellCheck = NULL, facet = NULL, facetMincount = NULL,
  facetMultiselect = NULL, curlopts = list(), ...)
```

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 taxononKey=212 (i.e. /occurrence/search?taxonKey=212) will match all birds, no matter which species. You can pass many keys by passing occ_search in a call to an lapply-family function (see last example below).
scientificName	A scientific name from the GBIF backbone. All included and synonym taxa are included in the search.
country	The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded. See here <http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2>
publishingCountry	The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded.
hasCoordinate	(logical) Return only occurence records with lat/long data (TRUE) or all records (FALSE, default).
typeStatus	Type status of the specimen. One of many options. See ?typestatus
recordNumber	Number recorded by collector of the data, different from GBIF record number. See <http://rs.tdwg.org/dwc/terms/#recordNumber> for more info
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	Continent. One of africa, antarctica, asia, europe, north_america (North America includes the Caribbean and reachies down and includes Panama), oceania, or south_america

`geometry` Searches for occurrences inside a polygon described in Well Known Text (WKT) format. A WKT shape written as either POINT, LINESTRING, LINEARRING POLYGON, or MULTIPOLYGON. Example of a polygon: POLYGON((30.1 10.1, 20, 20 40, 40 40, 30.1 10.1)) would be queried as <http://bit.ly/1BzNwDq>. See also the section **WKT** below.

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

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

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

`recordedBy` The person who recorded the occurrence.

`basisOfRecord` Basis of record, as defined in our BasisOfRecord enum here <http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/BasisOfRecord.html> Acceptable values are:

- FOSSIL_SPECIMEN An occurrence record describing a fossilized specimen.
- HUMAN_OBSERVATION An occurrence record describing an observation made by one or more people.
- LITERATURE An occurrence record based on literature alone.
- LIVING_SPECIMEN An occurrence record describing a living specimen, e.g.
- MACHINE_OBSERVATION An occurrence record describing an observation made by a machine.
- OBSERVATION An occurrence record describing an observation.
- PRESERVED_SPECIMEN An occurrence record describing a preserved specimen.
- UNKNOWN Unknown basis for the record.

`datasetKey` The occurrence dataset key (a uuid)

`eventDate` Occurrence date 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)

`catalogNumber` 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 WGS 84. 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 WGS 84. 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. Not guaranteed to be que.

collectionCode — 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. See Details. Issues passed to this parameter filter results by the issue.

search — Query terms. The value for this parameter can be a simple word or a phrase.

mediaType — Media type. Default is NULL, so no filtering on mediatype. Options: NULL, 'MovingImage', 'Sound', and '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.

establishmentMeans — (character) EstablishmentMeans, possible values include: INTRODUCED, INVASIVE, MANAGED, NATIVE, NATURALISED, UNCERTAIN

protocol — (character) Protocol or mechanism used to provide the occurrence record. See Details for possible values

license — (character) The type license applied to the dataset or record. Possible values: CC0_1_0, CC_BY_4_0, CC_BY_NC_4_0, UNSPECIFIED, and UNSUPPORTED

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 200,000, which is calculated as the limit+start, so start=199,000 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, latitude, and longitute. Or specify each field you want returned by name, e.g. fields = c('name','latitude','elevation').
return	One of data, hier, meta, or all. If data, a data.frame with the data. hier returns the classifications in a list for each record. meta returns the metadata for the entire call. all gives all data back in a list.
spellCheck	(logical) If TRUE ask GBIF to check your spelling of the value passed to the search parameter. IMPORTANT: This only checks the input to the search parameter, and no others. Default: FALSE
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.
curlopts	list of named curl options passed on to HttpClient. see 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.frame's 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. If the return parameter is set to something other than default you get back just the meta, data, hier, or media.

References

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

See Also

downloads(), occ_data(), occ_facet()

Examples

```
## Not run:
# Search by species name, using \code{\link{name_backbone}} first to get key
(key <- name_suggest(q='Helianthus annuus', rank='species')$key[1])
occ_search(taxonKey=key, limit=2)

# Return 20 results, this is the default by the way
occ_search(taxonKey=key, limit=20)

# Return just metadata for the search
occ_search(taxonKey=key, limit=0, return='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{scientifcName} 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', return='data', limit=20)

# 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,
   return="data")
occ_search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a',start=20,limit=5,
```

```
   return="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
## If return='data' the output is a data.frame of all data together
## for easy manipulation
occ_search(taxonKey=key, return='data', limit=20)

# Taxonomic hierarchy data
## If return='meta' the output is a list of the hierarch for each record
occ_search(taxonKey=key, return='hier', limit=10)

# Search by recorder
occ_search(recordedBy="smith", limit=20)

# Many collector names
occ_search(recordedBy=c("smith","BJ Stacey"), limit=20)

# Pass in curl options for extra fun
occ_search(taxonKey=2433407, limit=20, return='hier',
  curlopts=list(verbose=TRUE))
occ_search(taxonKey=2433407, limit=20, return='hier',
  curlopts = list(
    noprogress = FALSE,
    progressfunction = function(down, up) {
      cat(sprintf("up: %d | down %d\n", up, down))
      return(TRUE)
    }
  )
)
# occ_search(taxonKey=2433407, limit=20, return='hier',
#   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)$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 latitidue and longitude
```

```
occ_search(search="kingfisher", decimalLatitude=50, decimalLongitude=-10)

# Search on a bounding box
## in well known text format
### polygon
occ_search(geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))', limit=20)
### multipolygon
wkt <- 'MULTIPOLYGON(((-123 38, -123 43, -116 43, -116 38, -123 38)),
   ((-97 41, -97 45, -93 45, -93 41, -97 41)))'
occ_search(geometry = gsub("\n\\s+", "", wkt), limit = 20)

## taxonKey + WKT
key <- name_suggest(q='Aesculus hippocastanum')$key[1]
occ_search(taxonKey=key, geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))',
   limit=20)
## or using bounding box, converted to WKT internally
occ_search(geometry=c(-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"`
##  - An additional alternative is to use the GBIF download API, see ?downloads
wkt <- "POLYGON((13.26349675655365 52.53991761181831,18.36115300655365 54.11445544219924,
21.87677800655365 53.80418956368524,24.68927800655365 54.217364774722455,28.20490300655365
54.320018299365124,30.49005925655365 52.85948216284084,34.70880925655365 52.753220564427814,
35.93927800655365 50.46131871049754,39.63068425655365 49.55761261299145,40.86115300655365
46.381388009130845,34.00568425655365 45.279102926537,33.30255925655365 48.636868465271846,
30.13849675655365 49.78513301801265,28.38068425655365 47.2236377039631,29.78693425655365
44.6572866068524,27.67755925655365 42.62220075124676,23.10724675655365 43.77542058000212,
24.51349675655365 47.10412345120368,26.79865300655365 49.55761261299145,23.98615300655365
52.00209943876426,23.63459050655365 49.44345313705238,19.41584050655365 47.580567827212114,
19.59162175655365 44.90682206053508,20.11896550655365 42.36297154876359,22.93146550655365
40.651849782081555,25.56818425655365 39.98171166226459,29.61115300655365 40.78507856230178,
32.95099675655365 40.38459278067577,32.95099675655365 37.37491910393631,26.27130925655365
33.65619609886799,22.05255925655365 36.814081996401605,18.71271550655365 36.1072176729021,
18.53693425655365 39.16878677351903,15.37287175655365 38.346355762190846,15.19709050655365
41.578843777436326,12.56037175655365 41.050735748143424,12.56037175655365 44.02872991212046,
15.19709050655365 45.52594200494078,16.42755925655365 48.05271546733352,17.48224675655365
48.86865641518059,10.62677800655365 47.817178329053135,9.57209050655365 44.154980365192,
8.16584050655365 40.51835445724746,6.05646550655365 36.53210972067291,0.9588092565536499
31.583640057148145,-5.54509699344635 35.68001485298146,-6.77556574344635 40.51835445724746,
-9.41228449344635 38.346355762190846,-12.40056574344635 35.10683619158607,-15.74040949344635
38.07010978950028,-14.68572199344635 41.31532459432774,-11.69744074344635 43.64836179231387,
-8.88494074344635 42.88035509418534,-4.31462824344635 43.52103366008421,-8.35759699344635
47.2236377039631,-8.18181574344635 50.12441989397795,-5.01775324344635 49.55761261299145,
-2.73259699344635 46.25998980446569,-1.67790949344635 44.154980365192,-1.32634699344635
39.30493590580802,2.18927800655365 41.44721797271696,4.47443425655365 43.26556960420879,
2.18927800655365 46.7439668697322,1.83771550655365 50.3492841273576,6.93537175655365
49.671505849335254,5.00177800655365 52.32557322466785,7.81427800655365 51.67627099802223,
7.81427800655365 54.5245591562317,10.97834050655365 51.89375191441792,10.97834050655365
55.43241335888528,13.26349675655365 52.53991761181831))"
```

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

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

#### if WKT too long, with 'geom_big=bbox': makes into bounding box
res <- occ_search(geometry = wkt, geom_big = "bbox")$data
library("rgeos")
library("sp")
wktsp <- readWKT(wkt)
plot(wktsp)
coordinates(res) <- ~decimalLongitude+decimalLatitude
points(res)

#### Or, use 'geom_big=axe'
(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
(res <- occ_search(geometry = wkt, geom_big = "axe", geom_size=50))
###### more calls
(res <- occ_search(geometry = wkt, geom_big = "axe", geom_size=30))


# Search on country
occ_search(country='US', fields=c('name','country'), limit=20)
isocodes[grep("France", isocodes$name),"code"]
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
occ_search(country=c('US','DE'), limit=20)
### one request, many instances of same parameter: use semi-colon sep. string
occ_search(country = 'US;DE', limit=20)

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

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

# Get occurrences for a particular eventDate
occ_search(taxonKey=key, eventDate="2013", limit=20)
occ_search(taxonKey=key, year="2013", limit=20)
occ_search(taxonKey=key, month="6", limit=20)

# Get occurrences based on depth
key <- name_backbone(name='Salmo salar', kingdom='animals')$speciesKey
occ_search(taxonKey=key, depth="5", limit=20)

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

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

# Get occurrences based on collectionCode
occ_search(collectionCode="Floristic Databases MV - Higher Plants", limit=20)
occ_search(collectionCode=c("Floristic Databases MV - Higher Plants","Artport"))

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

# Search using a query string
# occ_search(search = "kingfisher", limit=20)
## spell check - only works with the `search` parameter
### spelled correctly - same result as above call
# occ_search(search = "kingfisher", limit=20, spellCheck = TRUE)
### spelled incorrectly - stops with suggested spelling
# occ_search(search = "kajsdkla", limit=20, spellCheck = TRUE)
### spelled incorrectly - stops with many suggested spellings
###   and number of results for each
# occ_search(search = "helir", limit=20, spellCheck = TRUE)



# search on repatriated - doesn't work right now
# occ_search(repatriated = "")

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

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

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

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

# search on familyKey
occ_search(familyKey = 3925, limit = 5)

# search on genusKey
occ_search(genusKey = 1935496, limit = 5)

# search on establishmentMeans
occ_search(establishmentMeans = "INVASIVE", limit = 5)
occ_search(establishmentMeans = "NATIVE", limit = 5)
occ_search(establishmentMeans = "UNCERTAIN", limit = 5)
```

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

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

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

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

# search on 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')

# 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', return = 'meta')
```

```
occ_search(continent = 'africa', return = 'meta')
occ_search(continent = 'oceania', return = 'meta')
occ_search(continent = 'antarctica', return = 'meta')

# Search for occurrences with images
occ_search(mediaType = 'StillImage', return='media')
occ_search(mediaType = 'MovingImage', return='media')
occ_search(mediaType = 'Sound', return='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 cant be matched to the GBIF
# backbone properly:
occ_search(datasetKey='84c0e1a0-f762-11e1-a439-00145eb45e9a',
   issue=c('TAXON_MATCH_NONE','TAXON_MATCH_HIGHERRANK'))

# Parsing output by issue
(res <- occ_search(geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 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, return="data")
# nrow(out)

# If you pass in an invalid polygon you get hopefully informative errors

### the WKT string is fine, but GBIF says bad polygon
wkt <- 'POLYGON((-178.59375 64.83258989321493,-165.9375 59.24622380205539,
-147.3046875 59.065977905449806,-130.78125 51.04484764446178,-125.859375 36.70806354647625,
-112.1484375 23.367471303759686,-105.1171875 16.093320185359257,-86.8359375 9.23767076398516,
-82.96875 2.9485268155066175,-82.6171875 -14.812060061226388,-74.8828125 -18.849111862023985,
-77.34375 -47.661687803329166,-84.375 -49.975955187343295,174.7265625 -50.649460483096114,
```

```
179.296875 -42.19189902447192,-176.8359375 -35.634976650677295,176.8359375 -31.835565983656227,
163.4765625 -6.528187613695323,152.578125 1.894796132058301,135.703125 4.702353722559447,
127.96875 15.077427674847987,127.96875 23.689804541429606,139.921875 32.06861069132688,
149.4140625 42.65416193033991,159.2578125 48.3160811030533,168.3984375 57.019804336633165,
178.2421875 59.95776046458139,-179.6484375 61.16708631440347,-178.59375 64.83258989321493))'

# 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, return = "meta")

## End(Not run)
```

`occ_spellcheck` *Spell check search term for occurrence searches*

Description

Spell check search term for occurrence searches

Usage

```
occ_spellcheck(search, curlopts = list())
```

Arguments

`search`	(character) query term
`curlopts`	list of named curl options passed on to `HttpClient`. see `curl_options` for curl options

Value

A boolean if search term spelled correctly, or if not spelled correctly with no suggested alternatives. If spelled incorrectly and suggested alternatives given, we give back a list with slots "correctlySpelled" (boolean) and "suggestions" (list)

Examples

```
## Not run:
# incorrectly spelled, with suggested alternative
occ_spellcheck(search = "kajsdkla")

# incorrectly spelled, without > 1 suggested alternative
occ_spellcheck(search = "helir")

# incorrectly spelled, without no alternatives
occ_spellcheck(search = "asdfadfasdf")

# correctly spelled,  alternatives
occ_spellcheck(search = "helianthus")

## End(Not run)
```

`organizations` *Organizations metadata.*

Description

Organizations metadata.

Usage

```
organizations(data = "all", 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'`
`uuid`	(character) UUID of the data node provider. This must be specified if data is anything other than 'all'.
`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_options` for curl options

Value

A list of length one or two. If `uuid` is NULL, then a data.frame with call metadata, and a data.frame, but if `uuid` given, then a list.

References

http://www.gbif.org/developer/registry#organizations

Examples

```
## Not run:
organizations(limit=5)
organizations(query="france", limit=5)
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

```
parsenames(scientificname, curlopts = list())
```

Arguments

`scientificname`	A character vector of scientific names.
`curlopts`	list of named curl options passed on to `HttpClient`. see `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

http://www.gbif.org/developer/species#parser

Examples

```
## 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)
```

`rgbif-defunct` *Defunct functions in rgbif*

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
- `togeojson`: moving this functionality to spocc package, will be removed soon
- `gist`: moving this functionality to spocc package, will be removed soon

Details

The above functions have been removed. See https://github.com/ropensci/rgbif and poke around the code if you want to find the old functions in previous versions of the package, or email Scott at <myrmecocystus@gmail.com>

`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

```
rgb_country_codes(country_name="United")
```

---

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)
```

---

typestatus *Type status options for GBIF searching*

Description

- name. Name of type.
- description. Description of the type.

wkt_parse *parse wkt into smaller bits*

Description

parse wkt into smaller bits

Usage

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

Examples

```
wkt <- "POLYGON((13.26349675655365 52.53991761181831,18.36115300655365 54.11445544219924,
21.87677800655365 53.80418956368524,24.68927800655365 54.217364774722455,28.20490300655365
54.320018299365124,30.49005925655365 52.85948216284084,34.70880925655365 52.753220564427814,
35.93927800655365 50.46131871049754,39.63068425655365 49.55761261299145,40.86115300655365
46.381388009130845,34.00568425655365 45.279102926537,33.30255925655365 48.636868465271846,
30.13849675655365 49.78513301801265,28.38068425655365 47.2236377039631,29.78693425655365
44.6572866068524,27.67755925655365 42.62220075124676,23.10724675655365 43.77542058000212,
24.51349675655365 47.10412345120368,26.79865300655365 49.55761261299145,23.98615300655365
52.00209943876426,23.63459050655365 49.44345313705238,19.41584050655365 47.580567827212114,
19.59162175655365 44.90682206053508,20.11896550655365 42.36297154876359,22.93146550655365
40.651849782081555,25.56818425655365 39.98171166226459,29.61115300655365 40.78507856230178,
32.95099675655365 40.38459278067577,32.95099675655365 37.37491910393631,26.27130925655365
33.65619609886799,22.05255925655365 36.814081996401605,18.71271550655365 36.1072176729021,
18.53693425655365 39.16878677351903,15.37287175655365 38.346355762190846,15.19709050655365
41.578843777436326,12.56037175655365 41.050735748143424,12.56037175655365 44.02872991212046,
15.19709050655365 45.52594200494078,16.42755925655365 48.05271546733352,17.48224675655365
48.86865641518059,10.62677800655365 47.817178329053135,9.57209050655365 44.154980365192,
8.16584050655365 40.51835445724746,6.05646550655365 36.53210972067291,0.9588092565536499
31.583640057148145,-5.54509699344635 35.68001485298146,-6.77556574344635 40.51835445724746,
-9.41228449344635 38.346355762190846,-12.40056574344635 35.10683619158607,-15.74040949344635
38.07010978950028,-14.68572199344635 41.31532459432774,-11.69744074344635 43.64836179231387,
-8.88494074344635 42.88035509418534,-4.31462824344635 43.52103366008421,-8.35759699344635
47.2236377039631,-8.18181574344635 50.12441989397795,-5.01775324344635 49.55761261299145,
-2.73259699344635 46.25998980446569,-1.67790949344635 44.154980365192,-1.32634699344635
39.30493590580802,2.18927800655365 41.44721797271696,4.47443425655365 43.26556960420879,
2.18927800655365 46.7439668697322,1.83771550655365 50.3492841273576,6.93537175655365
49.671505849335254,5.00177800655365 52.32557322466785,7.81427800655365 51.67627099802223,
```

```
7.81427800655365 54.5245591562317,10.97834050655365 51.89375191441792,10.97834050655365
55.43241335888528,13.26349675655365 52.53991761181831))"
wkt <- gsub("\n", " ", wkt)

# 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