Package 'spocc'

October 14, 2022

Title Interface to Species Occurrence Data Sources

Description A programmatic interface to many species occurrence data sources, including Global Biodiversity Information Facility ('GBIF'), 'USGSs' Biodiversity Information Serving Our Nation ('BISON'), 'iNaturalist', 'eBird', Integrated Digitized 'Biocollections' ('iDigBio'), 'VertNet', Ocean 'Biogeographic' Information System ('OBIS'), and Atlas of Living Australia ('ALA'). Includes functionality for retrieving species occurrence data, and combining those data.

Version 1.2.0

License MIT + file LICENSE

URL https://github.com/ropensci/spocc (devel), https://docs.ropensci.org/spocc/ (user manual)

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

LazyData true

Encoding UTF-8

Language en-US

Imports utils, rgbif, rbison, rebird, rvertnet, ridigbio, lubridate, crul, whisker, jsonlite, data.table, tibble, wellknown

Suggests testthat, taxize, vcr

RoxygenNote 7.1.1

X-schema.org-applicationCategory Biodiversity

X-schema.org-keywords specimens, API, web-services, occurrences, species, taxonomy, GBIF, INAT, BISON, Vertnet, eBird, iDigBio, OBIS, ALA

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

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>), Karthik Ram [ctb], Ted Hart [ctb], rOpenSci [fnd] (https://ropensci.org/)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2021-01-05 20:50:03 UTC

R topics documented:

`spocc-package` *Interface to many species occurrence data sources*

Description

A programmatic interface to many species occurrence data sources, including GBIF, USGS's BISON, iNaturalist, Berkeley Ecoinformatics Engine, eBird, iDigBio, VertNet, OBIS, and ALA. Includes functionality for retrieving species occurrence data, and combining that data.

Package API

The main function to use is `occ()` - a single interface to many species occurrence databases (see below for a list).

Other functions include:

- `occ2df()` - Combine results from `occ` into a data.frame
- `wkt_vis()` - Visualize WKT strings (used to define geometry based searches for some data sources) in an interactive map

Currently supported species occurrence data sources

Provider	Web
GBIF	
BISON	
eBird	
iNaturalist	
VertNet	
iDigBio	
OBIS	
ALA	

Duplicates

See `spocc_duplicates()` for more.

Clean data

All data cleaning functionality is in a new package: `scrubr` (). On CRAN: . See also package

Make maps

All mapping functionality is now in a separate package: `mapr`` () (formerly known as spocc-utils‘). On CRAN:

Author(s)

Scott Chamberlain

`as.ala` *Coerce occurrence keys to ALA id objects*

Description

Coerce occurrence keys to ALA id objects

Usage

```
as.ala(x, ...)
```

Arguments

`x`	Various inputs, including the output from a call to `occ()` (class occdat), `occ2df()` (class data.frame), or a list, numeric, alakey, or occkey.
`...`	curl options; named parameters passed on to `crul::HttpClient()`

Value

One or more in a list of both class alakey and occkey

See Also

Other coercion: as.bison(), as.gbif(), as.idigbio(), as.inat(), as.obis(), as.vertnet()

Examples

```
## Not run:
spnames <- c('Barnardius zonarius', 'Grus rubicunda', 'Cracticus tibicen')
out <- occ(query=spnames, from='ala', limit=2)
(res <- occ2df(out))
(tt <- as.ala(out))
as.ala(x = res$key[1])

## End(Not run)
```

---

as.bison *Coerce occurrence keys to bisonkey/occkey objects*

Description

Coerce occurrence keys to bisonkey/occkey objects

Usage

```
as.bison(x, ...)
```

Arguments

x	Various inputs, including the output from a call to occ() (class occdat), occ2df() (class data.frame), or a list, numeric, character, or bisonkey, or occkey.
...	curl options; named parameters passed on to crul::HttpClient()

Details

Internally, we use rbison::bison_solr(), same function we use internally within the occ() function. Although, we query here with the occurrenceID parameter to get the occurrence directly instead of searching for it.

Value

One or more in a list of both class bisonkey and occkey

See Also

Other coercion: as.ala(), as.gbif(), as.idigbio(), as.inat(), as.obis(), as.vertnet()

Examples

```
## Not run:
spnames <- c('Accipiter striatus', 'Setophaga caerulescens',
  'Spinus tristis')
out <- occ(query=spnames, from='bison', limit=2)
res <- occ2df(out)
(tt <- as.bison(out))
(uu <- as.bison(res))
as.bison(as.numeric(res$key[1]))
as.bison(res$key[1])
as.bison(as.list(res$key[1:2]))
as.bison(tt[[1]])
as.bison(uu[[1]])
as.bison(tt[1:2])

## End(Not run)
```

---

as.ecoengine *Coerce occurrence keys to ecoenginekey/occkey objects*

Description

DEFUNCT

Usage

```
as.ecoengine(...)
```

Arguments

...	ignored

---

as.gbif *Coerce occurrence keys to gbifkey/occkey objects*

Description

Coerce occurrence keys to gbifkey/occkey objects

Usage

```
as.gbif(x, ...)
```

Arguments

x	Various inputs, including the output from a call to `occ()` (class occdat), `occ2df()` (class data.frame), or a list, numeric, character, gbifkey, or occkey.
...	curl options; named parameters passed on to `crul::HttpClient()`

Details

Internally, we use `rgbif::occ_get()`, whereas `occ()` uses `rgbif::occ_data()`. We can use `rgbif::occ_get()` here because we have the occurrence key to go directly to the occurrence record.

Value

One or more in a list of both class gbifkey and occkey

See Also

Other coercion: `as.ala()`, `as.bison()`, `as.idigbio()`, `as.inat()`, `as.obis()`, `as.vertnet()`

Examples

```
## Not run:
spnames <- c('Accipiter striatus', 'Setophaga caerulescens',
  'Spinus tristis')
out <- occ(query=spnames, from=c('gbif','ebird'),
  gbifopts=list(hasCoordinate=TRUE), limit=2)
res <- occ2df(out)
(tt <- as.gbif(out))
(uu <- as.gbif(res))
as.gbif(as.numeric(res$key[1]))
as.gbif(res$key[1])
as.gbif(as.list(res$key[1:2]))
as.gbif(tt[[1]])
as.gbif(uu[[1]])
as.gbif(tt[1:2])

## End(Not run)
```

`as.idigbio` *Coerce occurrence keys to idigbio objects*

Description

Coerce occurrence keys to idigbio objects

Usage

```
as.idigbio(x, ...)
```

Arguments

`x`	Various inputs, including the output from a call to `occ()` (class occdat), `occ2df()` (class data.frame), or a list, numeric, character, idigbiokey, or occkey.
`...`	curl options; named parameters passed on to `httr::GET()`

Details

Internally, we use `idig_view_records`, whereas we use `idig_search_records()` in the `occ()` function.

Value

One or more in a list of both class idigbiokey and occkey

See Also

Other coercion: `as.ala()`, `as.bison()`, `as.gbif()`, `as.inat()`, `as.obis()`, `as.vertnet()`

Examples

```
## Not run:
spnames <- c('Accipiter striatus', 'Setophaga caerulescens',
  'Spinus tristis')
out <- occ(query=spnames, from='idigbio', limit=2)
res <- occ2df(out)
(tt <- as.idigbio(out))
(uu <- as.idigbio(res))
as.idigbio(res$key[1])
as.idigbio(as.list(res$key[1:2]))
as.idigbio(tt[[1]])
as.idigbio(uu[[1]])
as.idigbio(tt[1:2])

library("dplyr")
bind_rows(lapply(tt, function(x) data.frame(unclass(x)$data)))

## End(Not run)
```

---

as.inat *Coerce occurrence keys to iNaturalist id objects*

Description

Coerce occurrence keys to iNaturalist id objects

Usage

```
as.inat(x, ...)
```

Arguments

x	Various inputs, including the output from a call to `occ()` (class occdat), `occ2df()` (class data.frame), or a list, numeric, character, inatkey, or occkey.
...	curl options; named parameters passed on to `crul::HttpClient()`

Value

One or more in a list of both class inatkey and occkey

See Also

Other coercion: as.ala(), as.bison(), as.gbif(), as.idigbio(), as.obis(), as.vertnet()

Examples

```
## Not run:
spnames <- c('Accipiter striatus', 'Setophaga caerulescens',
  'Spinus tristis')
out <- occ(query=spnames, from='inat', limit=2)
res <- occ2df(out)
(tt <- as.inat(out))
(uu <- as.inat(res))
as.inat(res$key[1])
as.inat(as.list(res$key[1:2]))
as.inat(tt[[1]])
as.inat(uu[[1]])
as.inat(tt[1:2])

## End(Not run)
```

---

as.obis *Coerce occurrence keys to obis id objects*

Description

Coerce occurrence keys to obis id objects

Usage

```
as.obis(x, ...)
```

Arguments

x	Various inputs, including the output from a call to occ() (class occdat), occ2df() (class data.frame), or a list, numeric, obiskey, or occkey.
...	curl options; named parameters passed on to crul::HttpClient()

Value

One or more in a list of both class obiskey and occkey

See Also

Other coercion: as.ala(), as.bison(), as.gbif(), as.idigbio(), as.inat(), as.vertnet()

Examples

```
## Not run:
spnames <- c('Mola mola', 'Loligo vulgaris', 'Stomias boa')
out <- occ(query=spnames, from='obis', limit=2)
(res <- occ2df(out))
(tt <- as.obis(out))
(uu <- as.obis(res))
as.obis(x = res$key[1])
as.obis(as.list(res$key[1:2]))
as.obis(tt[[1]])
as.obis(uu[[1]])
as.obis(tt[1:2])

library("data.table")
rbindlist(lapply(tt, "[[", "results"),
  use.names = TRUE, fill = TRUE)

## End(Not run)
```

`as.vertnet` *Coerce occurrence keys to vertnetkey/occkey objects*

Description

Coerce occurrence keys to vertnetkey/occkey objects

Usage

```
as.vertnet(x)
```

Arguments

x	Various inputs, including the output from a call to `occ()` (class occdat), `occ2df()` (class data.frame), or a list, numeric, character, vertnetkey, or occkey.

Details

Internally, we use `rvertnet::vert_id()`, whereas `occ()` uses `rvertnet::vertsearch()`.

Value

One or more in a list of both class vertnetkey and occkey

See Also

Other coercion: `as.ala()`, `as.bison()`, `as.gbif()`, `as.idigbio()`, `as.inat()`, `as.obis()`

Examples

```
## Not run:
# spnames <- c('Accipiter striatus', 'Setophaga caerulescens',
#   'Spinus tristis')
# out <- occ(query=spnames, from='vertnet', has_coords=TRUE, limit=2)
# res <- occ2df(out)
# (tt <- as.vertnet(out))
# (uu <- as.vertnet(res))
# keys <- Filter(Negate(is.na), res$key)
# as.vertnet(keys[1])
# as.vertnet(as.list(keys[1:2]))
# as.vertnet(tt[[1]])
# as.vertnet(uu[[1]])
# as.vertnet(tt[1:2])

## End(Not run)
```

---

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

```
bbox2wkt(minx = NA, miny = NA, maxx = NA, maxy = NA, bbox = NULL)

wkt2bbox(wkt)
```

Arguments

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

Value

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))'

wkt2bbox returns a numeric vector of length 4, like c(minx, miny, maxx, maxy).

See Also

Other bbox: wkt_vis()

Other bbox: wkt_vis()

Examples

```
# Convert a bounding box to a WKT

## Pass in a vector of length 4 with all values
bbox2wkt(bbox = c(-125.0,38.4,-121.8,40.9))

## Or pass in each value separately
bbox2wkt(-125.0, 38.4, -121.8, 40.9)

# Convert a WKT object to a bounding box
wkt <- "POLYGON((-125 38.4,-125 40.9,-121.8 40.9,-121.8 38.4,-125 38.4))"
wkt2bbox(wkt)

identical(
 bbox2wkt(-125.0, 38.4, -121.8, 40.9),
 "POLYGON((-125 38.4,-121.8 38.4,-121.8 40.9,-125 40.9,-125 38.4))"
)

identical(
 c(-125.0, 38.4, -121.8, 40.9),
 as.numeric(
   wkt2bbox(
     "POLYGON((-125 38.4,-125 40.9,-121.8 40.9,-121.8 38.4,-125 38.4))"
   )
 )
)
```

---

inspect *Get more data on individual occurrences*

Description

Fetches the complete record, which may or may not be the same as requested through occ(). Some data providers have different ways to retrieve many occurrence records vs. single occurrence records - and sometimes the results are more verbose when retrieving a single occurrence record.

Usage

```
inspect(x, from = "gbif")

## S3 method for class 'data.frame'
inspect(x, from = "gbif")
```

```
## S3 method for class 'occdat'
inspect(x, from = "gbif")

## S3 method for class 'occkey'
inspect(x, from = "gbif")
```

Arguments

x	The output from `occ()` call, output from call to `occ2df()`, or an occurrence ID as a occkey class.
from	(character) The data provider. One of gbif, bison, inat, or vertnet

Value

A list, with each slot named for the data source, and then within data sources is a slot for each taxon, named by it's occurrence ID.

Examples

```
## Not run:
spnames <- c('Accipiter striatus', 'Spinus tristis')
out <- occ(query=spnames, from=c('gbif','bison'),
   gbifopts=list(hasCoordinate=TRUE), limit=2)
res <- occ2df(out)
inspect(res)

out <- occ(query=spnames, from='gbif', gbifopts=list(hasCoordinate=TRUE),
  limit=4)
res <- occ2df(out)
inspect(res)

# from occkeys
key <- as.gbif(res$key[1])
inspect(key)

# idigbio
spnames <- c('Accipiter striatus', 'Spinus tristis')
out <- occ(query=spnames, from='idigbio', limit=20)
inspect(out)

## End(Not run)
```

---

occ *Search for species occurrence data across many data sources.*

Description

Search on a single species name, or many. And search across a single or many data sources.

Usage

```
occ(
  query = NULL,
  from = "gbif",
  limit = 500,
  start = NULL,
  page = NULL,
  geometry = NULL,
  has_coords = NULL,
  ids = NULL,
  date = NULL,
  callopts = list(),
  gbifopts = list(),
  bisonopts = list(),
  inatopts = list(),
  ebirdopts = list(),
  vertnetopts = list(),
  idigbioopts = list(),
  obisopts = list(),
  alaopts = list(),
  throw_warnings = TRUE
)
```

Arguments

`query`	(character) One to many scientific names. See Details for what parameter in each data source we query. Note: ebird now expects species codes instead of scientific names - we pass you name through `rebird::species_code()` internally
`from`	(character) Data source to get data from, any combination of gbif, bison, inat, ebird, and/or vertnet
`limit`	(numeric) Number of records to return. This is passed across all sources. To specify different limits for each source, use the options for each source (gbifopts, bisonopts, inatopts, and ebirdopts). See Details for more. Default: 500 for each source. BEWARE: if you have a lot of species to query for (e.g., n = 10), that's 10 * 500 = 5000, which can take a while to collect. So, when you first query, set the limit to something smallish so that you can get a result quickly, then do more as needed.
`start, page`	(integer) Record to start at or page to start at. See `Paging` in Details for how these parameters are used internally. Optional
`geometry`	(character or nmeric) One of a Well Known Text (WKT) object, a vector of length 4 specifying a bounding box, or an sf object (sfg, sfc, or sf). This parameter searches for occurrences inside a polygon - converted to a polygon from whatever user input is given. A WKT shape written as `POLYGON((30.1 10.1, 20 40, 40 40, 30.1 10.1` would be queried as is, i.e. See Details for more examples of WKT objects. The format of a bounding box is `min-longitude, min-latitude, max-longitude, max-latitude`. Geometry is not possible with vertnet right now, but should be soon. See Details for more info on geometry inputs.

has_coords	(logical) Only return occurrences that have lat/long data. This works for gbif, rinat, idigbio, and vertnet, but is ignored for ebird and bison data sources. You can easily though remove records without lat/long data.
ids	Taxonomic identifiers. This can be a list of length 1 to many. See examples for usage. Currently, identifiers for only 'gbif' and 'bison' for parameter 'from' supported. If this parameter is used, query parameter can not be used - if it is, a warning is thrown.
date	(character/Date) A length 2 vector containing two dates of the form YYY-MM-DD. These can be character of Date class. These are used to do a date range search. Of course there are other types of date searches one may want to do but date range seems like the most common date search use case.
callopts	Options passed on to crul::HttpClient, e.g., for debugging curl calls, setting timeouts, etc.
gbifopts	(list) List of named options to pass on to `rgbif::occ_search()`. See also `occ_options()`
bisonopts	(list) List of named options to pass on to `rbison::bison()`. See also `occ_options()`
inatopts	(list) List of named options to pass on to internal function `get_inat_obs`
ebirdopts	(list) List of named options to pass on to `rebird::ebirdregion()` or `rebird::ebirdgeo()`. See also `occ_options()`
vertnetopts	(list) List of named options to pass on to `rvertnet::searchbyterm()`. See also `occ_options()`.
idigbioopts	(list) List of named options to pass on to `ridigbio::idig_search_records()`. See also `occ_options()`.
obisopts	(list) List of named options to pass on to internal function. See and obis_search for what parameters can be used.
alaopts	(list) List of named options to pass on to internal function. See `Occurrence search` part of the API docs at for possible parameters.
throw_warnings	(logical) `occ()` collects errors returned from each data provider when they occur, and are accessible in the `$meta$errors` slot for each data provider. If you set `throw_warnings=TRUE`, we give these request errors as warnings with `warning()`. if `FALSE`, we don't give warnings, but you can still access them in the output.

Details

The `occ` function is an opinionated wrapper around the rgbif, rbison, rinat, rebird, rvertnet and ridigbio packages (as well as internal custom wrappers around some data sources) to allow data access from a single access point. We take care of making sure you get useful objects out at the cost of flexibility/options - although you can still set options for each of the packages via the gbifopts, bisonopts, inatopts, etc. parameters.

Value

an object of class `occdat`, with a print method to give a brief summary. The print method only shows results for those that have some results (those with no results are not shown). The `occdat` class is just a thin wrapper around a named list, wher the top level names are the data sources:

- gbif
- bison
- inat
- ebird
- vertnet
- idigbio
- obis
- ala

Note that you only get data back for sources that were specified in the `from` parameter. All others are present, but empty.

Then within each data source is an object of class `occdatind` holding another named list that contains:

- meta: metadata
  - source: the data source name (e.g., "gbif")
  - time: time the request was sent
  - found: number of records found (number found across all queries)
  - returned: number of records returned (number of rows in all data.frame's in the `data` slot)
  - type: query type, only "sci" for scientific
  - opts: a named list with the options you sent to the data source
  - errors: a character vector of errors returned, if any occurred
- data: named list of data.frame's, named by the queries sent

Inputs

All inputs to `occ` are one of:

- scientific name
- taxonomic id
- geometry as bounds, WKT, os Spatial classes

To search by common name, first use `occ_names()` to find scientic names or taxonomic IDs, then feed those to this function. Or use the `taxize` package to get names and/or IDs to use here.

Using the query parameter

When you use the `query` parameter, we pass your search terms on to parameters within functions that query data sources you specify. Those parameters are:

- rgbif - `scientificName` in the `rgbif::occ_search()` function - API parameter: same as the `occ` parameter
- rebird - `species` in the `rebird::ebirdregion()` or `rebird::ebirdgeo()` functions, depending on whether you set `method="ebirdregion"` or `method="ebirdgeo"` - API parameters: `sci` for both `rebird::ebirdregion()` and `rebird::ebirdgeo()`

- rbison - `species` or `scientificName` in the `rbison::bison()` or `rbison::bison_solr()` functions, respectively. If you don't pass anything to `geometry` parameter we use `bison_solr`, and if you do we use `bison` - API parameters: same as `occ` parameters
- rvertnet - `taxon` in the `rvertnet::vertsearch()` function - API parameter: `q`
- ridigbio - `scientificname` in the `ridigbio::idig_search_records()` function - API parameter: `scientificname`
- inat - internal function - API parameter: `q`
- obis - internal function - API parameter: `scientificName`
- ala - internal function - API parameter: `q`

If you have questions about how each of those parameters behaves with respect to the terms you pass to it, lookup documentation for those functions, or get in touch at the development repository

iDigBio notes

When searching iDigBio note that by deafult we set `fields = "all"`, so that we return a richer suite of fields than the `ridigbio` R client gives by default. But you can changes this by passing in a `fields` parameter to `idigbioopts` parameter with the specific fields you want.

Maximum of 100,000 results are allowed to be returned. See

BISON notes

We use two different functions when you request data from `bison`. We use `rbison::bison_solr()` by default as it's more flexible. If you pass a value to the `geometry` parameter we use `rbison::bison()`. We'd prefer to just use one function to simplify things, but `rbison::bison_solr()` doesn't support geometry queries.

iNaturalist notes

We're using the iNaturalist API, docs at

API rate limits: max of 100 requests per minute, though they ask that you try to keep it to 60 requests per minute or lower. If they notice usage that has serious impact on their performance they may institute blocks without notification.

There is a hard limit 0f 10,000 observations with the iNaturalist API. We do paging internally so you may not see this aspect, but for example, if you request 12,000 records, you won't be able to get that many. The API will error at anything more than 10,000. We now error if you request more than 10,000 from iNaturalist. There are some alternatives:

- Consider exporting data while logged in to your iNaturalist account, or the iNaturalist research grade observations within GBIF - see - at time of this writing it has 8.5 million observations.
- Search for iNaturalist data within GBIF. e.g., the following searches for iNaturalist data within GBIF and allows more than 10,000 records: "

limit parameter

The `limit` parameter is set to a default of 25. This means that you will get **up to** 25 results back for each data source you ask for data from. If there are no results for a particular source, you'll get zero back; if there are 8 results for a particular source, you'll get 8 back. If there are 26 results for a particular source, you'll get 25 back. You can always ask for more or less back by setting the limit parameter to any number. If you want to request a different number for each source, pass the appropriate parameter to each data source via the respective options parameter for each data source.

WKT

WKT objects are strings of pairs of lat/long coordinates that define a shape. Many classes of shapes are supported, including POLYGON, POINT, and MULTIPOLYGON. Within each defined shape define all vertices of the shape with a coordinate like 30.1 10.1, the first of which is the latitude, the second the longitude.

Examples of valid WKT objects:

- 'POLYGON((30.1 10.1, 10 20, 20 60, 60 60, 30.1 10.1))'
- 'POINT((30.1 10.1))'
- 'LINESTRING(3 4,10 50,20 25)'
- 'MULTIPOINT((3.5 5.6),(4.8 10.5))")'
- 'MULTILINESTRING((3 4,10 50,20 25),(-5 -8,-10 -8,-15 -4))'
- 'MULTIPOLYGON(((1 1,5 1,5 5,1 5,1 1),(2 2,2 3,3 3,3 2,2 2)),((6 3,9 2,9 4,6 3)))'
- 'GEOMETRYCOLLECTION(POINT(4 6),LINESTRING(4 6,7 10))'

Only POLYGON objects are currently supported.

Getting WKT polygons or bounding boxes. We will soon introduce a function to help you select a bounding box but for now, you can use a few sites on the web.

- Bounding box -
- Well known text -

geometry parameter

The behavior of the `occ` function with respect to the `geometry` parameter varies depending on the inputs to the `query` parameter. Here are the options:

- geometry (single), no query - If a single bounding box/WKT string passed in, and no query, a single query is made against each data source.
- geometry (many), no query - If many bounding boxes/WKT strings are passed in, we do a separate query for each bounding box/WKT string against each data source.
- geometry (single), query - If a single bounding box/WKT string passed in, and a single query, we do a single query against each data source.
- geometry (many), query - If many bounding boxes/WKT strings are passed in, and a single query, we do a separate query for each bounding box/WKT string with the same queried name against each data source.

- geometry (single), many query - If a single bounding box/WKT string passed in, and many names to query, we do a separate query for each name, using the same geometry, for each data source.
- geometry (many), many query - If many bounding boxes/WKT strings are passed in, and many names to query, this poses a problem for all data sources, none of which accept many bounding boxes of WKT strings. So, in this scenario, we loop over each name and each geometry query, and then re-combine by queried name, so that you get back a single group of data for each name.

Geometry options by data provider

wkt & bbox allowed, see WKT section above

- gbif
- bison
- obis
- ala

bbox only

- inat
- idigbio

No spatial search allowed

- ebird
- vertnet

Notes on the date parameter

Date searches with the `date` parameter are allowed for all sources except ebird.

Notes on some special cases

- idigbio: We search on the `datecollected` field. Other date fields can be searched on, but we chose `datecollected` as it seemed most appropriate.
- vertnet: If you want more flexible date searches, you can pass various types of date searches to `vertnetopts`. See `rvertnet::searchbyterm()` for more information
- ala: There's some issues with the dates returned from ALA. They are returned as time stamps, and some seem to be malformed. So do beware of using ALA dates for important things.

Get in touch if you have other date search use cases you think are widely useful

Paging

All data sources respond to the `limit` parameter passed to `occ`.

Data sources, however, vary as to whether they respond to an offset. Here's the details on which data sources will respond to `start` and which to the `page` parameter:

- gbif - Responds to `start`. Default: 0

- bison - Responds to `start`. Default: 0
- inat - Responds to `page`. Default: 1
- ebird - No paging, both `start` and `page` ignored.
- vertnet - No paging implemented here, both `start` and `page` ignored. VertNet does have a form of paging, but it uses a cursor, and can't easily be included here via parameters. However, `rvertnet` does paging internally for you. For example, the max records per request for VertNet is 1000; if you request 2000 records, we'll do the first request, and do the second request for you automatically.
- idigbio - Responds to `start`. Default: 0
- obis - Does not respond to `start`. They only allow a starting occurrence UUID up to which to skip. So order of results matters a great deal of course. To paginate with OBIS, do e.g. `obisopts = list(after = "017b7818-5b2c-4c88-9d76-f4471afe5584")`; `after` can be combined with the `limit` value you pass in to the main `occ()` function call. See obis_search for what parameters can be used.
- ala - Responds to `start`. Default: 0

Photographs

The iNaturalist data source provides photographs of the records returned, if available. For example, the following will give photos from inat: `occ(query = 'Danaus plexippus', from = 'inat')$inat$data$Danaus_plexip`

BEWARE

In cases where you request data from multiple providers, especially when including GBIF, there could be duplicate records since many providers' data eventually ends up with GBIF. See `spocc_duplicates()` for more.

See Also

Other queries: `occ_names_options()`, `occ_names()`, `occ_options()`, `spocc_objects`

Examples

```
## Not run:
# Single data sources
(res <- occ(query = 'Accipiter striatus', from = 'gbif', limit = 5))
res$gbif
(res <- occ(query = 'Accipiter striatus', from = 'ebird', limit = 50))
res$ebird
(res <- occ(query = 'Danaus plexippus', from = 'inat', limit = 50,
  has_coords = TRUE))
res$inat
res$inat$data
data.table::rbindlist(res$inat$data$Danaus_plexippus$photos)
(res <- occ(query = 'Bison bison', from = 'bison', limit = 50))
res$bison
(res <- occ(query = 'Bison bison', from = 'vertnet', limit = 5))
res$vertnet
res$vertnet$data$Bison_bison
```

```
occ2df(res)

# Paging
one <- occ(query = 'Accipiter striatus', from = 'gbif', limit = 5)
two <- occ(query = 'Accipiter striatus', from = 'gbif', limit = 5, start = 5)
one$gbif
two$gbif

# iNaturalist limits: they allow at most 10,000; query through GBIF to get
# more than 10,000
# See
# x <- occ(query = 'Danaus plexippus', from = 'gbif', limit = 10100,
#   gbifopts = list(datasetKey = "50c9509d-22c7-4a22-a47d-8c48425ef4a7"))
# x$gbif

# Date range searches across data sources
## Not possible for ebird
## bison
occ(query = 'Acer', date = c('2010-08-08', '2010-08-21'), from = 'bison', limit=5)
## ala
occ(date = c('2018-01-01T00:00:00Z', '2018-03-28T00:00:00Z'), from = 'ala', limit = 5)
## gbif
occ(query = 'Accipiter striatus', date = c('2010-08-01', '2010-08-31'), from = 'gbif', limit=5)
## vertnet
occ(query = 'Mustela nigripes', date = c('1990-01-01', '2015-12-31'), from = 'vertnet', limit=5)
## idigbio
occ(query = 'Acer', date = c('2010-01-01', '2015-12-31'), from = 'idigbio', limit=5)
## obis
occ(query = 'Mola mola', date = c('2015-01-01', '2015-12-31'), from = 'obis', limit=5)
## inat
occ(query = 'Danaus plexippus', date = c('2015-01-01', '2015-12-31'), from = 'inat', limit=5)


# Restrict to records with coordinates
occ(query = "Acer", from = "idigbio", limit = 5, has_coords = TRUE)

occ(query = 'Setophaga caerulescens', from = 'ebird', ebirdopts = list(loc='US'))
occ(query = 'Spinus tristis', from = 'ebird', ebirdopts =
   list(method = 'ebirdgeo', lat = 42, lng = -76, dist = 50))

# idigbio data
## scientific name search
occ(query = "Acer", from = "idigbio", limit = 5)
occ(query = "Acer", from = "idigbio", idigbioopts = list(offset = 5, limit  = 3))
## geo search
bounds <- c(-120, 40, -100, 45)
occ(from = "idigbio", geometry = bounds, limit = 10)
## just class arachnida, spiders
occ(idigbioopts = list(rq = list(class = 'arachnida')), from = "idigbio", limit = 10)
## search certain recordsets
sets <- c("1ffce054-8e3e-4209-9ff4-c26fa6c24c2f",
    "8dc14464-57b3-423e-8cb0-950ab8f36b6f",
    "26f7cbde-fbcb-4500-80a9-a99daa0ead9d")
```

```
occ(idigbioopts = list(rq = list(recordset = sets)), from = "idigbio", limit = 10)

# Many data sources
(out <- occ(query = 'Pinus contorta', from=c('gbif','bison','vertnet'), limit=10))

## Select individual elements
out$gbif
out$gbif$data
out$vertnet

## Coerce to combined data.frame, selects minimal set of
## columns (name, lat, long, provider, date, occurrence key)
occ2df(out)

# Pass in limit parameter to all sources. This limits the number of occurrences
# returned to 10, in this example, for all sources, in this case gbif and inat.
occ(query='Pinus contorta', from=c('gbif','inat'), limit=10)

# Geometry
## Pass in geometry parameter to all sources. This constraints the search to the
## specified polygon for all sources, gbif and bison in this example.
## Check out to get a WKT string
occ(query='Accipiter', from='gbif',
   geometry='POLYGON((30.1 10.1, 10 20, 20 60, 60 60, 30.1 10.1))')
occ(query='Helianthus annuus', from='bison', limit=50,
   geometry='POLYGON((-111.06 38.84, -110.80 39.37, -110.20 39.17, -110.20 38.90,
                      -110.63 38.67, -111.06 38.84))')

## Or pass in a bounding box, which is automatically converted to WKT (required by GBIF)
## via the bbox2wkt function. The format of a bounding box is
## [min-longitude, min-latitude, max-longitude, max-latitude].
occ(query='Accipiter striatus', from='gbif', geometry=c(-125.0,38.4,-121.8,40.9))

## lots of results, can see how many by indexing to meta
res <- occ(query='Accipiter striatus', from='gbif',
   geometry='POLYGON((-69.9 49.2,-69.9 29.0,-123.3 29.0,-123.3 49.2,-69.9 49.2))')
res$gbif

## You can pass in geometry to each source separately via their opts parameter, at
## least those that support it. Note that if you use rinat, you reverse the order, with
## latitude first, and longitude second, but here it's the reverse for consistency across
## the spocc package
bounds <- c(-125.0,38.4,-121.8,40.9)
occ(query = 'Danaus plexippus', from="inat", geometry=bounds)

## Passing geometry with multiple sources
occ(query = 'Danaus plexippus', from=c("inat","gbif"), geometry=bounds)

## Using geometry only for the query
### A single bounding box
occ(geometry = bounds, from = "gbif", limit=50)
### Many bounding boxes
occ(geometry = list(c(-125.0,38.4,-121.8,40.9), c(-115.0,22.4,-111.8,30.9)), from = "gbif")
```

```
## Many geometry and many names
res <- occ(query = c('Danaus plexippus', 'Accipiter striatus'),
  geometry = list(c(-125.0,38.4,-121.8,40.9), c(-115.0,22.4,-111.8,30.9)), from = "bison")
res

## Geometry only with WKT
wkt <- 'POLYGON((-98.9 44.2,-89.1 36.6,-116.7 37.5,-102.5 39.6,-98.9 44.2))'
occ(from = "gbif", geometry = wkt, limit = 10)

# Specify many data sources, another example
ebirdopts = list(loc = 'US'); gbifopts  =  list(country = 'US')
out <- occ(query = 'Setophaga caerulescens', from = c('gbif','inat','bison','ebird'),
    gbifopts = gbifopts, ebirdopts = ebirdopts, limit=20)
occ2df(out)

# Pass in many species names, combine just data to a single data.frame, and
# first six rows
spnames <- c('Accipiter striatus', 'Setophaga caerulescens', 'Spinus tristis')
(out <- occ(query = spnames, from = 'gbif', gbifopts = list(hasCoordinate = TRUE), limit=25))
df <- occ2df(out)
head(df)

# no query, geometry, or ids passed
## many dataset keys to gbif
dsets <- c("14f3151a-e95d-493c-a40d-d9938ef62954", "f934f8e2-32ca-46a7-b2f8-b032a4740454")
occ(limit = 20, from = "gbif", gbifopts = list(datasetKey = dsets))
## class name to idigbio
occ(limit = 20, from = "idigbio", idigbioopts = list(rq = list(class = 'arachnida')))

# taxize integration
## You can pass in taxonomic identifiers
library("taxize")
(ids <- get_ids(c("Chironomus riparius","Pinus contorta"), db = c('itis','gbif')))
occ(ids = ids[[1]], from='bison', limit=20)
occ(ids = ids, from=c('bison','gbif'), limit=20)

(ids <- get_ids("Chironomus riparius", db = 'gbif'))
occ(ids = ids, from='gbif', limit=20)

(ids <- get_gbifid("Chironomus riparius"))
occ(ids = ids, from='gbif', limit=20)

(ids <- get_tsn('Accipiter striatus'))
occ(ids = ids, from='bison', limit=20)

## sf classes
library("sp")
library("sf")
one <- Polygon(cbind(c(91,90,90,91), c(30,30,32,30)))
spone = Polygons(list(one), "s1")
sppoly = SpatialPolygons(list(spone), as.integer(1))
```

```
## single polygon in a sf class
x <- st_as_sf(sppoly)
out <- occ(geometry = x, limit=50)
out$gbif$data
mapr::map_leaflet(out)

## single polygon in a sfc class
x <- st_as_sf(sppoly)
out <- occ(geometry = x[[1]], limit=50)
out$gbif$data

## single polygon in a sf POLYGON class
x <- st_as_sf(sppoly)
x <- unclass(x[[1]])[[1]]
class(x)
out <- occ(geometry = x, limit=50)
out$gbif$data

## two polygons in an sf class
one <- Polygon(cbind(c(-121.0,-117.9,-121.0,-121.0), c(39.4, 37.1, 35.1, 39.4)))
two <- Polygon(cbind(c(-123.0,-121.2,-122.3,-124.5,-123.5,-124.1,-123.0),
                     c(44.8,42.9,41.9,42.6,43.3,44.3,44.8)))
spone = Polygons(list(one), "s1")
sptwo = Polygons(list(two), "s2")
sppoly = SpatialPolygons(list(spone, sptwo), 1:2)
sppoly_df <- SpatialPolygonsDataFrame(sppoly,
   data.frame(a=c(1,2), b=c("a","b"), c=c(TRUE,FALSE),
   row.names=row.names(sppoly)))
x <- st_as_sf(sppoly_df)
out <- occ(geometry = x, limit=50)
out$gbif$data


# curl debugging
occ(query = 'Accipiter striatus', from = 'gbif', limit=10,
 callopts=list(verbose = TRUE))
occ(query = 'Accipiter striatus', from = 'bison', limit=10,
 callopts=list(verbose = TRUE))
occ(query = 'Accipiter striatus', from = 'inat',
 callopts=list(verbose = TRUE))
occ(query = 'Mola mola', from = 'obis', limit = 200,
 callopts = list(verbose = TRUE))

########## More thorough data source specific examples
# idigbio
## scientific name search
res <- occ(query = "Acer", from = "idigbio", limit = 5)
res$idigbio

## geo search
### bounding box
bounds <- c(-120, 40, -100, 45)
occ(from = "idigbio", geometry = bounds, limit = 10)
```

```
### wkt
# wkt <- 'POLYGON((-69.9 49.2,-69.9 29.0,-123.3 29.0,-123.3 49.2,-69.9 49.2))'
wkt <- 'POLYGON((-98.9 44.2,-89.1 36.6,-116.7 37.5,-102.5 39.6,-98.9 44.2))'
occ(from = "idigbio", geometry = wkt, limit = 10)

## limit fields returned
occ(query = "Acer", from = "idigbio", limit = 5,
   idigbioopts = list(fields = "scientificname"))

## offset and max_items
occ(query = "Acer", from = "idigbio", limit = 5,
   idigbioopts = list(offset = 10))

## sort
occ(query = "Acer", from = "idigbio", limit = 5,
   idigbioopts = list(sort = TRUE))$idigbio
occ(query = "Acer", from = "idigbio", limit = 5,
   idigbioopts = list(sort = FALSE))$idigbio

## more complex queries
### parameters passed to "rq", get combined with the name queried
occ(query = "Acer", from = "idigbio", limit = 5,
   idigbioopts = list(rq = list(basisofrecord="fossilspecimen")))$idigbio

#### NOTE: no support for multipolygons yet
## WKT's are more flexible than bounding box's. You can pass in a WKT with multiple
## polygons like so (you can use POLYGON or MULTIPOLYGON) when specifying more than one
## polygon. Note how each polygon is in it's own set of parentheses.
# occ(query='Accipiter striatus', from='gbif',
#    geometry='MULTIPOLYGON((30 10, 10 20, 20 60, 60 60, 30 10),
#                           (30 10, 10 20, 20 60, 60 60, 30 10))')

# OBIS examples
## basic query
(res <- occ(query = 'Mola mola', from = 'obis', limit = 200))
## get to obis data
res$obis
## get obis + gbif data
(res <- occ(query = 'Mola mola', from = c('obis', 'gbif'), limit = 200))
res$gbif
res$obis
## no match found
(res <- occ(query = 'Linguimaera thomsonia', from = 'obis'))
## geometry query
geometry <- "POLYGON((8.98 48.05,15.66 48.05,15.66 45.40,8.98 45.40,8.98 48.05))"
(res <- occ(from = 'obis', geometry = geometry, limit = 50))
res$obis

## Pass in spatial classes
## sp classes no longer supported

## Paging
(res1 <- occ(query = 'Mola mola', from = 'obis', limit = 10))
```

```
occ_ids <- res1$obis$data$Mola_mola$id
(res2 <- occ(query = 'Mola mola', from = 'obis',
  limit = 10, obisopts = list(after = occ_ids[length(occ_ids)])))
res1$obis
res2$obis
## Pass in any parameters to obisopts as a list
(res <- occ(query = 'Mola mola', from = 'obis',
   obisopts = list(startdepth = 40, enddepth = 50)))
min(res$obis$data$Mola_mola$minimumDepthInMeters, na.rm=TRUE)
max(res$obis$data$Mola_mola$maximumDepthInMeters, na.rm=TRUE)


# ALA examples
## basic query
(res <- occ(query = 'Alaba vibex', from = 'ala', limit = 200))
## get to ala data
res$ala
occ2df(res)

# geometry search
(x <- occ(query = "Macropus", from = 'ala',
  geometry = "POLYGON((145 -37,150 -37,150 -30,145 -30,145 -37))"))
x$ala
occ2df(x)

## End(Not run)
```

`occ2df` *Combine results from occ calls to a single data.frame*

Description

Combine results from occ calls to a single data.frame

Usage

```
occ2df(obj, what = "data")
```

Arguments

`obj`	Input from occ, an object of class `occdat`, or an object of class `occdatind`, the individual objects from each source within the `occdat` class.
`what`	(character) One of data (default) or all (with metadata)

Details

This function combines a subset of data from each data provider to a single data.frame, or metadata plus data if you request `what="all"`. The single data.frame contains the following columns:

- name - scientific (or common) name

- longitude - decimal degree longitude
- latitude - decimal degree latitude
- prov - data provider
- date - occurrence record date
- key - occurrence record key

Examples

```
## Not run:
# combine results from output of an occ() call
spnames <- c('Accipiter striatus', 'Setophaga caerulescens',
  'Spinus tristis')
out <- occ(query=spnames, from='gbif', gbifopts=list(hasCoordinate=TRUE),
  limit=10)
occ2df(out)
occ2df(out$gbif)

out <- occ(
  query='Accipiter striatus',
  from=c('gbif','bison','ebird','inat'),
  gbifopts=list(hasCoordinate=TRUE), limit=2)
occ2df(out)
occ2df(out$bison)

# or combine many results from a single data source
spnames <- c('Accipiter striatus', 'Spinus tristis')
out <- occ(query=spnames, from='gbif', limit=2)
occ2df(out$gbif)

## End(Not run)
```

---

occ_names *Search for species names across many data sources.*

Description

Search for species names across many data sources.

Usage

```
occ_names(
  query = NULL,
  from = "gbif",
  limit = 100,
  rank = "species",
  callopts = list(),
  gbifopts = list(),
  bisonopts = list()
)
```

Arguments

query	(character) One to many names. Either a scientific name or a common name. Only scientific names supported right now.
from	(character) Data source to get data from, any combination of gbif or bison
limit	(numeric) Number of records to return. This is passed across all sources. To specify different limits for each source, use the options for each source (gbifopts, bisonopts). See Details for more.
rank	(character) Taxonomic rank to limit search space. Used in GBIF, but not used in BISON.
callopts	Options passed on to `crul::HttpClient()`, e.g., for debugging curl calls, setting timeouts, etc.
gbifopts	(list) List of named options to pass on to `rgbif::name_lookup()`. See also `occ_names_options()`
bisonopts	(list) List of named options to pass on to `rbison::bison_tax()`. See also `occ_names_options()`

Details

Not all 7 data sources available from the `occ()` function are available here, as not all of those sources have functionality to search for names.

We strongly encourage you to use the `taxize` package if you want to search for taxonomic or common names, convert common to scientific names, etc. That package was built exactly for that purpose, and we only provide a bit of name searching here in this function.

See Also

Other queries: `occ_names_options()`, `occ_options()`, `occ()`, `spocc_objects`

Examples

```
## Not run:
# Single data sources
## gbif
(res <- occ_names(query = 'Accipiter striatus', from = 'gbif'))
head(res$gbif$data[[1]])

## bison
(res <- occ_names(query = '*bear', from = 'bison'))
res$bison$data

## End(Not run)
```

occ_names_options *Look up options for parameters passed to each source for occ_names function*

Description

Look up options for parameters passed to each source for occ_names function

Usage

```
occ_names_options(from = "gbif", where = "console")
```

Arguments

from	(character) Data source to get data from, any combination of gbif or bison. Case doesn't matter.
where	(character) One of console (print to console) or html (opens help page, if in non-interactive R session, prints help to console).

Details

Any of the parameters passed to e.g. `rgbif::name_lookup()` from the `rgbif` package can be passed in the associated gbifopts list in `occ()`.

Note that the from parameter is lowercased within the function and is called through `match.arg` first, so you can match on unique partial strings too (e.g., 'rb' for 'rbison').

Value

Opens up the documentation for the function that is used internally within the occ function for each source.

See Also

Other queries: `occ_names()`, `occ_options()`, `occ()`, `spocc_objects`

Examples

```
## Not run:
# opens up documentation for this function
occ_names_options()

# Open up documentation for the appropriate search function for each source
occ_names_options('gbif')
occ_names_options('bison')

# Or open in html version
occ_names_options('bison', 'html')

## End(Not run)
```

occ_options *Look up options for parameters passed to each source*

Description

Look up options for parameters passed to each source

Usage

```
occ_options(from = "gbif", where = "console")
```

Arguments

from	(character) Data source to get data from, any combination of gbif, bison, ebird, idigibio and/or vertnet. Case doesn't matter. inat is not included here, see that package's help docs.
where	(character) One of console (print to console) or html (opens help page, if in non-interactive R session, prints help to console).

Details

Any of the parameters passed to e.g. `rgbif::occ_data()` from the `rgbif` package can be passed in the associated gbifopts list in `occ()`

Note that the from parameter is lowercased within the function and is called through match.arg first, so you can match on unique partial strings too (e.g., 'rv' for 'rvertnet').

Value

Opens up the documentation for the function that is used internally within the occ function for each source.

See Also

Other queries: `occ_names_options()`, `occ_names()`, `occ()`, `spocc_objects`

Examples

```
## Not run:
# opens up documentation for this function
occ_options()

# Open up documentation for the appropriate search function for each source
occ_options('gbif')
occ_options('ebird')
occ_options('bison')
occ_options('idigbio')
occ_options('vertnet')
```

```
# Or open in html version
occ_options('bison', 'html')

## End(Not run)
```

`spocc_duplicates` *A note about duplicate occurrence records*

Description

BEWARE: spocc provides you a nice interface to many data providers for species occurrence data. However, in cases where you request data from GBIF *in addition* to other data sources, there could be duplicate records. This is because GBIF is, to use an ecology analogy, a top predator, and pulls in data from lower nodes in the food chain. For example, iNaturalist provides data to GBIF, so if you search for occurrence records for *Pinus contorta* from iNaturalist and GBIF, you could get, for example, 20 of the same records.

We think a single R interface to many occurrence record providers will provide a consistent way to work with occurrence data, making analyses and vizualizations more repeatable across providers.

For cleaning data, see packages `scrubr` () and `CoordinateCleaner` ()

Do get in touch with us if you have concerns, have ideas for eliminating duplicates

---

`wkt_vis` *Visualize well-known text area's on a map.*

Description

This can be helpful in visualizing the area in which you are searching for occurrences with the `occ()` function.

Usage

```
wkt_vis(x, zoom = 6, maptype = "terrain", browse = TRUE)
```

Arguments

`x`	Input well-known text area (character)
`zoom`	Zoom level, defaults to 6 (numeric)
`maptype`	Map type, default is terrain (character)
`browse`	Open in browser or not. If not, gives back path to html file. Default: `TRUE` (logical)

Details

Uses Mapbox's map layers, openes in your default browser

See Also

Other bbox: bbox2wkt()

Examples

```
## Not run:
poly <- 'POLYGON((-111.06 38.84, -110.80 39.37, -110.20 39.17, -110.20 38.90,
     -110.63 38.67, -111.06 38.84))'
wkt_vis(poly)

poly2 <- 'POLYGON((-125 38.4,-125 40.9,-121.8 40.9,-121.8 38.4,-125 38.4))'
wkt_vis(poly2)

# Multiple polygons
x <- "POLYGON((-125 38.4, -121.8 38.4, -121.8 40.9, -125 40.9, -125 38.4),
(-115 22.4, -111.8 22.4, -111.8 30.9, -115 30.9, -115 22.4))"
wkt_vis(x)

# don't open in browser
poly2 <- 'POLYGON((-125 38.4,-125 40.9,-121.8 40.9,-121.8 38.4,-125 38.4))'
wkt_vis(poly2, browse = FALSE)

## End(Not run)
```

Index