Package 'galah'

January 13, 2023

Type Package

Title Biodiversity Data from the Living Atlas Community

Version 1.5.1

Description The living atlas community provides tools to enable users to find, access, combine and visualise data on biodiversity. 'galah' enables the R community to directly access data and resources hosted by the living atlases.

Depends R (>= 4.1.0)

Imports assertthat, crayon, crul, data.tree, digest, dplyr, glue (>= 1.3.2), httr, jsonlite (>= 0.9.8), lifecycle (>= 1.0.0), readr, rlang, sf, stringr (>= 1.0.0), tibble, tidyselect, utils

Suggests vcr (>= 0.6.0), covr, gt, kableExtra, knitr, magrittr, pkgdown, rmarkdown, testthat

License MPL-2.0

URL https://galah.ala.org.au

BugReports https://github.com/AtlasOfLivingAustralia/galah-R/issues

Maintainer Martin Westgate <martin.westgate@csiro.au>

LazyLoad yes

VignetteBuilder knitr

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author Martin Westgate [aut, cre], Matilda Stevenson [aut], Dax Kellie [aut], Peggy Newman [aut]

Repository CRAN

Date/Publication 2023-01-13 09:20:07 UTC

R topics documented:

`atlas_citation` *Generate a citation for occurrence data*

Description

If a `data.frame` was generated using `atlas_occurrences()`, and the `mint_doi` argument was set to `TRUE`, the DOI associated with that dataset is appended to the resulting `data.frame` as an attribute. This function simply formats that DOI as a citation that can be included in a scientific publication. Please also consider citing this package, using the information in `citation("galah")`.

Usage

```
atlas_citation(data)
```

Arguments

data	data.frame: occurrence data generated by atlas_occurrences()

Value

A string containing the citation for that dataset.

Examples

```
## Not run:
atlas_citation(doi)

## End(Not run)
```

---

atlas_counts *Return a count of records*

Description

Prior to downloading data it is often valuable to have some estimate of how many records are available, both for deciding if the query is feasible, and for estimating how long it will take to download. Alternatively, for some kinds of reporting, the count of observations may be all that is required, for example for understanding how observations are growing or shrinking in particular locations, or for particular taxa. To this end, `atlas_counts()` takes arguments in the same format as `atlas_occurrences()`, and provides either a total count of records matching the criteria, or a `data.frame` of counts matching the criteria supplied to the `group_by` argument.

Usage

```
atlas_counts(
  request = NULL,
  identify = NULL,
  filter = NULL,
  geolocate = NULL,
  data_profile = NULL,
  group_by = NULL,
  limit = NULL,
  type = c("record", "species"),
  refresh_cache = FALSE
)
```

Arguments

request	optional data_request object: generated by a call to galah_call().
identify	data.frame: generated by a call to galah_identify().
filter	data.frame: generated by a call to galah_filter()
geolocate	string: generated by a call to galah_geolocate()
data_profile	string: generated by a call to galah_apply_profile()
group_by	data.frame: An object of class galah_group_by, as returned by galah_group_by(). Alternatively a vector of field names (see search_all(fields) and show_all(fields).
limit	numeric: maximum number of categories to return, defaulting to 100. If limit is NULL, all results are returned. For some categories this will take a while.
type	string: one of c("record", "species"). Defaults to "record". If "species", the number of species matching the criteria will be returned, if "record", the number of records matching the criteria will be returned.
refresh_cache	logical: if set to TRUE and galah_config(caching = TRUE) then files cached from a previous query will be replaced by the current query

Value

An object of class tbl_df and data.frame (aka a tibble) returning:

- A single number, if group_by is not specified or,
- A summary of counts grouped by field(s), if group_by is specified

Examples

```
galah_call() |>
  galah_filter(year == 2015) |>
  atlas_counts()
```

---

atlas_media *Get metadata on images, sounds and videos*

Description

In addition to text data describing individual occurrences and their attributes, ALA stores images, sounds and videos associated with a given record. atlas_media displays metadata for any and all of the media types.

Usage

```
atlas_media(
  request = NULL,
  identify = NULL,
  filter = NULL,
  geolocate = NULL,
  data_profile = NULL,
  download_dir = NULL,
  refresh_cache = FALSE
)
```

Arguments

request	optional data_request object: generated by a call to galah_call().
identify	data.frame: generated by a call to galah_identify().
filter	data.frame: generated by a call to galah_filter()
geolocate	string: generated by a call to galah_geolocate()
data_profile	string: generated by a call to galah_apply_profile()
download_dir	string: path to directory where downloaded media will be stored. DEPRECATED; use collect_media() instead.
refresh_cache	logical: if set to TRUE and galah_config(caching = TRUE) then files cached from a previous query will be replaced by the current query. DEPRECATED; use collect_media() instead.

Details

atlas_media() works by first finding all occurrence records matching the filter which contain media, then downloading the metadata for the media. To actually download the files themselves, use collect_media(). It may be beneficial when requesting a large number of records to show a progress bar by setting verbose = TRUE in galah_config().

Value

An object of class tbl_df and data.frame (aka a tibble) of metadata of the requested media.

See Also

atlas_counts() to find the number of records with media; but note this is not necessarily the same as the number of media files, as each record can have more than one media file associated with it (see examples section for how to do this).

Examples

```
## Not run:
# Download Regent Honeyeater records with multimedia attached
galah_call() |>
  galah_identify("Regent Honeyeater") |>
  galah_filter(year == 2011) |>
```

```
  atlas_media()

# Download multimedia
galah_call() |>
  galah_identify("Regent Honeyeater") |>
  galah_filter(year == 2011) |>
  atlas_media() |>
  collect_media(path = "folder/your-directory")

# Specify a single media type to download
galah_call() |>
  galah_identify("Eolophus Roseicapilla") |>
  galah_filter(multimedia == "Sound") |>
  atlas_media()

# It's good to check how many records have media files before downloading
galah_call() |>
  galah_filter(multimedia == c("Image", "Sound", "Video")) |>
  galah_group_by(multimedia) |>
  atlas_counts()

## End(Not run)
```

`atlas_occurrences` *Return occurrence records*

Description

The most common form of data stored by living atlases are observations of individual life forms, known as 'occurrences'. This function allows the user to search for occurrence records that match their specific criteria, and return them as a `data.frame` for analysis. Optionally, the user can also request a DOI for a given download to facilitate citation and re-use of specific data resources.

Usage

```
atlas_occurrences(
  request = NULL,
  identify = NULL,
  filter = NULL,
  geolocate = NULL,
  data_profile = NULL,
  select = NULL,
  mint_doi = FALSE,
  doi = NULL,
  refresh_cache = FALSE
)
```

Arguments

request	optional data_request object: generated by a call to galah_call().
identify	data.frame: generated by a call to galah_identify().
filter	data.frame: generated by a call to galah_filter()
geolocate	string: generated by a call to galah_geolocate()
data_profile	string: generated by a call to galah_apply_profile()
select	data.frame: generated by a call to galah_select()
mint_doi	logical: by default no DOI will be generated. Set to TRUE if you intend to use the data in a publication or similar
doi	**[Deprecated]** Use collect_occurrences instead. string: this argument enables retrieval of occurrence records previously downloaded from the ALA, using the DOI generated by the data.
refresh_cache	logical: if set to TRUE and galah_config(caching = TRUE) then files cached from a previous query will be replaced by the current query

Details

Note that unless care is taken, some queries can be particularly large. While most cases this will simply take a long time to process, if the number of requested records is >50 million the call will not return any data. Users can test whether this threshold will be reached by first calling atlas_counts() using the same arguments that they intend to pass to atlas_occurrences(). It may also be beneficial when requesting a large number of records to show a progress bar by setting verbose = TRUE in galah_config().

Value

An object of class tbl_df and data.frame (aka a tibble) of occurrences, containing columns as specified by galah_select(). The data.frame object has the following attributes:

- a listing of the user-supplied arguments of the data_request (i.e., identify, filter, geolocate, select)
- a doi of the data download
- the search_url of the query to ALA API

Examples

```
## Not run:
# Download occurrence records for a specific taxon
galah_config(email = "your_email_here")
galah_call() |>
  galah_identify("Reptilia") |>
  atlas_occurrences()

# Download occurrence records in a year range
galah_call() |>
  galah_identify("Litoria") |>
  galah_filter(year >= 2010 & year <= 2020) |>
```

```
  atlas_occurrences()

# Download occurrences records in a WKT-specified area
polygon <- "POLYGON((146.24960 -34.05930,
                     146.37045 -34.05930,
                     146.37045 -34.152549,
                     146.24960 -34.15254,
                     146.24960 -34.05930))"
galah_call() |>
  galah_identify("Reptilia") |>
  galah_filter(year >= 2010, year <= 2020) |>
  galah_geolocate(polygon) |>
  atlas_occurrences()

## End(Not run)
```

`atlas_species` *Return species lists*

Description

While there are reasons why users may need to check every record meeting their search criteria (i.e. using `atlas_occurrences()`), a common use case is to simply identify which species occur in a specified region, time period, or taxonomic group. This function returns a `data.frame` with one row per species, and columns giving associated taxonomic information.

Usage

```
atlas_species(
  request = NULL,
  identify = NULL,
  filter = NULL,
  geolocate = NULL,
  data_profile = NULL,
  refresh_cache = FALSE
)
```

Arguments

`request`	optional `data_request` object: generated by a call to `galah_call()`.
`identify`	`data.frame`: generated by a call to `galah_identify()`.
`filter`	`data.frame`: generated by a call to `galah_filter()`
`geolocate`	`string`: generated by a call to `galah_geolocate()`
`data_profile`	`string`: generated by a call to `galah_apply_profile()`
`refresh_cache`	`logical`: if set to `TRUE` and `galah_config(caching = TRUE)` then files cached from a previous query will be replaced by the current query

Details

The primary use case of this function is to extract species-level information given a set of criteria defined by `search_taxa()`, `galah_filter()` or `galah_geolocate()`. If the purpose is simply to get taxonomic information that is not restricted by filtering, then `search_taxa()` is more efficient. Similarly, if counts are required that include filter but without returning taxonomic detail, then `atlas_counts()` is more efficient (see examples).

Value

An object of class `tbl_df` and `data.frame` (aka a tibble), returning matching species The `data.frame` object has attributes listing of the user-supplied arguments of the `data_request` (i.e., identify, filter, geolocate, columns)

Examples

```
# First register a valid email address
galah_config(email = "ala4r@ala.org.au")

# Get a list of species within genus "Heleioporus"
# (every row is a species with associated taxonomic data)
galah_call() |>
  galah_identify("Heleioporus") |>
  atlas_species()

# Get a list of species within family "Peramelidae"
galah_call() |>
  galah_identify("peramelidae") |>
  atlas_species()

# It's good idea to find how many species there are before downloading
galah_call() |>
  galah_identify("Heleioporus") |>
  atlas_counts(type = "species")
```

`atlas_taxonomy` *Search taxonomic trees*

Description

The ALA has its' own internal taxonomy that is derived from authoritative sources. `atlas_taxonomy` provides a means to query that taxonomy, returning a tree (class `Node`) showing which lower clades are contained within the specified taxon.

Usage

```
atlas_taxonomy(request = NULL, identify = NULL, down_to = NULL)
```

Arguments

request	optional data_request object: generated by a call to galah_call().
identify	data.frame: generated by a call to galah_identify().
down_to	The identity of the clade at which the downwards search should stop. Should be specified using an object of class character and galah_down_to, as returned from galah_down_to(). Also accepts a string.

Details

The approach used by this function is recursive, meaning that it becomes slow for large queries such as atlas_taxonomy(search_taxa("Plantae"), down_to = galah_down_to(species)). Although the inputs to search_taxa and down_to are case-insensitive, node names are always returned in title case.

Value

A tree consisting of objects of class Node, containing the requested taxonomy. Each node contains the following attributes:

- name: The scientific name of the taxon in question
- rank: The taxonomic rank to which that taxon belongs
- guid: A unique identifier used by the ALA
- authority: The source of the taxonomic name & identifier

See Also

search_taxa() to search for an individual taxon; show_all(ranks) for valid ranks used to specify the down_to argument.

Examples

```
# Get a taxonomic tree of *Chordata* down to the class level
galah_call() |>
  galah_identify("chordata") |>
  galah_down_to(class) |>
  atlas_taxonomy()
```

clear_cached_files *Clear previously cached files*

Description

Deletes cached files within the cached file directory and their query metadata

Usage

```
clear_cached_files()
```

Value

No return value; called for side effect of removing files

Examples

```
## Not run:
# First set caching to true with [galah_config()]
galah_config(caching = TRUE)

# Then create a data query.
# The data you download will be cached in a temporary directory.
dat <- atlas_counts(group_by = galah_group_by(year))

# To clear your cached files directory, use `clear_cached_files()`
clear_cached_files()

## End(Not run)
```

collect_media *Collect media files*

Description

This function downloads full-sized or thumbnail images and media files using information from `atlas_media` to a local directory.

Usage

```
collect_media(
  df,
  type = c("full", "thumbnail"),
  path,
  download_dir,
  refresh_cache
)
```

Arguments

df	tibble returned by atlas_media() or show_all(media)
type	string: either "full" to download original images, or "thumbnail" to download thumbnails
path	string: path to directory where downloaded media will be stored
download_dir	**[Deprecated]** Use path instead.
refresh_cache	logical: if set to TRUE and galah_config(caching = TRUE) then files cached from a previous query will be replaced by the current query. NOTE: Unclear that this works right now.

Value

Available image & media files downloaded to a user local directory.

Examples

```
## Not run:
# Use `atlas_media()` to return a `tibble` of records that contain media
galah_call() |>
  galah_identify("perameles") |>
  galah_filter(year == 2015) |>
  atlas_media()

# To download media files, add `collect_media()` to the end of a query
galah_call() |>
  galah_identify("perameles") |>
  galah_filter(year == 2015) |>
  atlas_media() |>
  collect_media(path = here::here("folder", "subfolder"))

## End(Not run)
```

---

collect_occurrences *Collect occurrence records from a pre-existing DOI or URL*

Description

[Experimental]

Download occurrence records using an existing DOI or URL. Pre-existing DOIs and URLs come from previously generated downloads using atlas_occurrences or online.

Usage

```
collect_occurrences(url, doi)
```

Arguments

url	string: Retrieve occurrence records previously downloaded from the ALA, using the URL provided via email.
doi	string: Retrieve occurrence records previously downloaded from the ALA, using the DOI generated by the data.

Value

An object of class tbl_df and data.frame (aka a tibble) of occurrences

Examples

```
## Not run:
# Download previously retrieved records using an existing DOI or URL
collect_occurrences(doi = "your-doi")

# DOIs can be minted by adding `mint_doi = TRUE` to `atlas_occurrences()`
records <-
  galah_call() |>
  galah_identify("perameles") |>
  galah_filter(year == 2001) |>
  atlas_occurrences(mint_doi = TRUE)

attributes(records)$doi # return minted doi

## End(Not run)
```

---

count.data_request *Count for object of class* data_request

Description

[Experimental]

Usage

```
## S3 method for class 'data_request'
count(x, ..., wt, sort, name, type = c("record", "species"))
```

Arguments

x	An object of class data_request, created using galah_call()
...	currently ignored
wt	currently ignored
sort	currently ignored
name	currently ignored

type	string: one of c("record", "species"). Defaults to "record". If "species", the number of species matching the criteria will be returned, if "record", the number of records matching the criteria will be returned.

See Also

atlas_counts(), with which this function is synonymous.

---

filter.data_request *Narrow a query by specifying filters*

Description

"Filters" are arguments of the form field logical value that are used to narrow down the number of records returned by a specific query. For example, it is common for users to request records from a particular year (year == 2020), or to return all records except for fossils (basisOfRecord != "FossilSpecimen").

[Experimental]

Usage

```
## S3 method for class 'data_request'
filter(.data, ...)
```

Arguments

.data	An object of class data_request, created using galah_call()
...	filters, in the form field logical value

See Also

galah_filter(), with which this function is synonymous.

---

galah_apply_profile *Apply a data quality profile*

Description

A 'profile' is a group of filters that are pre-applied by the ALA. Using a data profile allows a query to be filtered quickly to the most relevant or quality-assured data that is fit-for-purpose. For example, the "ALA" profile is designed to exclude lower quality records, whereas other profiles apply filters specific to species distribution modelling (e.g. CDSM).

Usage

```
galah_apply_profile(...)
```

Arguments

...	a profile name. Should be a `string` - the name or abbreviation of a data quality profile to apply to the query. Valid values can be seen using `show_all(profiles)`

Details

Note that only one profile can be loaded at a time; if multiple profiles are given, the first valid profile is used.

For more bespoke editing of filters within a profile, use `galah_filter()`

Value

A tibble containing a valid data profile value.

See Also

`show_all()` and `search_all()` to look up available data profiles. `galah_filter()` can be used for more bespoke editing of individual data profile filters.

Examples

```
# Apply a data quality profile to a query
galah_call() |>
  galah_identify("reptilia") |>
  galah_filter(year == 2021) |>
  galah_apply_profile(ALA) |>
  atlas_counts()
```

---

galah_call *Start building a data query*

Description

To download data from the ALA (or another atlas), one must construct a data query. This query tells the atlas API what data to download and return, as well as how it should be filtered.

Usage

```
galah_call(
  identify = NULL,
  filter = NULL,
  select = NULL,
  geolocate = NULL,
  data_profile = NULL,
  group_by = NULL,
  down_to = NULL,
```

```
  ...
)

## S3 method for class 'data_request'
print(x, ...)
```

Arguments

identify	data.frame: generated by a call to galah_identify()
filter	data.frame: generated by a call to select_filters()
select	data.frame: generated by a call to galah_select()
geolocate	string: generated by a call to galah_geolocate()
data_profile	string: generated by a call to galah_apply_profile()
group_by	data.frame: generated by a call to galah_group_by()
down_to	data.frame: generated by a call to galah_down_to()
...	other function-specific request parameters
x	an object of class data_request

Details

The galah package enables users to construct their data queries using piping syntax (i.e., %>% from magrittr, or |> from base).

Start a query with galah_call(). Pipe functions like galah_identify(), galah_filter(), [galah_select()], and galah_group_by() to narrow your query and specify filters. Finish a query with an atlas_ function to identify which type of data is downloaded (i.e., atlas_occurrences(), atlas_counts(), atlas_species(), atlas_taxonomy() or atlas_media()).

Using galah_call() with pipes allows you to build & filter a query to download data in the same way that you would wrangle data with dplyr and the tidyverse.

Value

An object of class data_request.

Examples

```
# Begin your query with `galah_call()`, then pipe using `%>%` or `|>`

# Get number of records of *Aves* from 2001 to 2004 by year
galah_call() |>
  galah_identify("Aves") |>
  galah_filter(year > 2000 & year < 2005) |>
  galah_group_by() |>
  atlas_counts()

# Get information for all species in *Cacatuidae* family
galah_call() |>
  galah_identify("Cacatuidae") |>
```

```
  atlas_species()
## Not run:
# Download records of genus *Eolophus* from 2001 to 2004
galah_config(email = "your-email@email.com")

galah_call() |>
  galah_identify("Eolophus") |>
  galah_filter(year > 2000 & year < 2005) |>
  atlas_occurrences()

## End(Not run)
```

`galah_config` *Get or set configuration options that control galah behaviour*

Description

The `galah` package supports large data downloads, and also interfaces with the ALA which requires that users of some services provide a registered email address and reason for downloading data. The `galah_config` function provides a way to manage these issues as simply as possible.

Usage

```
galah_config(..., profile_path = NULL)

## S3 method for class 'galah_config'
print(x, ...)
```

Arguments

`...` Options can be defined using the form `name = "value"`. Valid arguments are:

- `atlas` string: Living Atlas to point to, Australia by default. Can be an organisation name, acronym, or region (see `show_all_atlases()` for admissible values)
- `caching` logical: if TRUE, results will be cached, and any cached results will be re-used). If FALSE, data will be downloaded.
- `cache_directory` string: the directory to use for the cache. By default this is a temporary directory, which means that results will only be cached within an R session and cleared automatically when the user exits R. The user may wish to set this to a non-temporary directory for caching across sessions. The directory must exist on the file system.
- `download_reason_id` numeric or string: the "download reason" required. by some ALA services, either as a numeric ID (currently 0–13) or a string (see `show_all(reasons)` for a list of valid ID codes and names). By default this is NA. Some ALA services require a valid download_reason_id code, either specified here or directly to the associated R function.

- email string: An email address that has been registered with the chosen atlas. For the ALA, you can register at this address.
- password string: A registerd password (GBIF only)
- run_checks logical: should galah run checks for filters and columns. If making lots of requests sequentially, checks can slow down the process and lead to HTTP 500 errors, so should be turned off. Defaults to TRUE.
- send_email logical: should you receive an email for each query to atlas_occurrences()? Defaults to FALSE; but can be useful in some instances, for example for tracking DOIs assigned to specific downloads for later citation.
- username string: A registerd username (GBIF only)
- verbose logical: should galah give verbose such as progress bars? Defaults to FALSE.

profile_path **[Deprecated]**

Keeping for compatibility with older package versions. It is preferable to not save galah_config options to a .Rprofile file.

x an object of class galah_config

Value

For galah_config(), a list of all options. When galah_config(...) is called with arguments, nothing is returned but the configuration is set.

Examples

```
## Not run:
# To download occurrence records, enter your email in `galah_config()`.
# This email should be registered with the ALA.
# You can register at:
# https://auth.ala.org.au/userdetails/registration/createAccount
galah_config(email = "your-email@email.com")

# Turn on caching in your session
galah_config(caching = TRUE)

# Some ALA services require that you add a reason for downloading data.
# Add your selected reason using the option `download_reason_id`
galah_config(download_reason_id = 0)

# To look up all valid reasons to enter, use `show_all(reasons)`
show_all(reasons)

# Make debugging in your session easier by setting `verbose = TRUE`
galah_config(verbose = TRUE)

## End(Not run)
```

galah_down_to *Specify the lowest taxonomic rank required in a downwards search*

Description

`atlas_taxonomy` generates a downwards search of the taxonomic tree. Use `galah_down_to()` to specify the taxonomic level to search to. `galah_down_to()` uses non-standard evaluation (NSE).

Usage

```
galah_down_to(...)
```

Arguments

`...`	the name of a single taxonomic rank

Value

A string with the named rank

See Also

`galah_select()`, `galah_filter()` and `galah_geolocate()` for related methods.

Examples

```
# Return a taxonomic tree of *Chordata* down to the class level
## Not run:
galah_call() |>
    galah_identify("Vertebrata") |>
    galah_down_to(class) |>
    atlas_taxonomy()

## End(Not run)
```

galah_filter *Narrow a query by specifying filters*

Description

"Filters" are arguments of the form `field logical value` that are used to narrow down the number of records returned by a specific query. For example, it is common for users to request records from a particular year (`year == 2020`), or to return all records except for fossils (`basisOfRecord != "FossilSpecimen"`).

The result of `galah_filter()` can be passed to the `filter` argument in `atlas_occurrences()`, `atlas_species()`, `atlas_counts()` or `atlas_media()`.

Usage

```
galah_filter(..., profile = NULL)
```

Arguments

`...`	filters, in the form `field logical value`
`profile`	**[Soft-deprecated]** Use `galah_apply_profile` instead. If supplied, should be a `string` recording a data quality profile to apply to the query. See `show_all_profiles()` for valid profiles. By default no profile is applied.

Details

`galah_filter` uses non-standard evaluation (NSE), and is designed to be as compatible as possible with `dplyr::filter()` syntax.

All statements passed to `galah_filter()` (except the `profile` argument) take the form of field - logical - value. Permissible examples include:

- = or == (e.g. `year = 2020`)
- `!=`, e.g. `year != 2020`)
- > or >= (e.g. `year >= 2020`)
- < or <= (e.g. `year <= 2020`)
- `OR` statements (e.g. `year == 2018 | year == 2020`)
- `AND` statements (e.g. `year >= 2000 & year <= 2020`)

In some cases `R` will fail to parse inputs with a single equals sign (=), particularly where statements are separated by `&` or `|`. This problem can be avoided by using a double-equals (==) instead.

Notes on behaviour

Separating statements with a comma is equivalent to an `AND` statement; Ergo `galah_filter(year >= 2010 & year < 2020)` is the same as `galah_filter(year >= 2010, year < 2020)`.

All statements must include the field name; so `galah_filter(year == 2010 | year == 2021)` works, as does `galah_filter(year == c(2010, 2021))`, but `galah_filter(year == 2010 | 2021)` fails.

It is possible to use an object to specify required values, e.g. `year_value <- 2010; galah_filter(year > year_value)`

`solr` supports range queries on text as well as numbers; so this is valid: `galah_filter(cl22 >= "Tasmania")`

Value

A tibble containing filter values.

See Also

`search_taxa()` and `galah_geolocate()` for other ways to restrict the information returned by `atlas_occurrences()` and related functions. Use `search_all(fields)` to find fields that you can filter by, and `show_values()` to find what values of those filters are available.

Examples

```
# Filter query results to return records of interest
galah_call() |>
  galah_filter(year >= 2019,
               basisOfRecord == "HumanObservation") |>
  atlas_counts()
```

`galah_geolocate` *Narrow a query to within a specified area*

Description

Restrict results to those from a specified area. Areas can be specified as either polygons or bounding boxes, depending on `type`.

Usage

```
galah_geolocate(..., type = c("polygon", "bbox"))
```

Arguments

`...`	a single `sf` object, WKT string or shapefile. Bounding boxes can be supplied as a `tibble`/`data.frame` or a `bbox`
`type`	`string`: one of `c("polygon", "bbox")`. Defaults to `"polygon"`. If `type = "polygon"`, a multipolygon will be built via `galah_polygon()`. If `type = "bbox"`, a multipolygon will be built via `galah_bbox()`. The multipolygon is used to narrow a query to the ALA.

Details

By default, `type` is set to `"polygon"` which narrows queries to within an area supplied as a `POLYGON`. Polygons must be specified as either an `sf` object, a 'well-known text' (WKT) string, or a shapefile. Shapefiles must be simple to be accepted by the ALA.

Alternatively, set `type = "bbox"` to narrow queries to within a bounding box. Bounding boxes can be extracted from a supplied `sf` object or a shapefile. A bounding box can also be supplied as a `bbox` object (via `sf::st_bbox()`) or a `tibble`/`data.frame`.

If `type = "polygon"`, WKT strings longer than 10000 characters and `sf` objects with more than 500 vertices will not be accepted by the ALA. Some polygons may need to be simplified. If `type = "bbox"`, sf objects and shapefiles will be converted to a bounding box to query the ALA.

Value

length-1 object of class `character` and `galah_geolocate`, containing a multipolygon WKT string representing the area provided.

See Also

galah_polygon() and galah_bbox() for specific functions to narrow queries by a specified area. search_taxa(), galah_filter() and galah_select() for other ways to restrict the information returned by atlas_occurrences() and related functions.

Examples

```
## Not run:
# Search for records within a polygon using a shapefile
location <- sf::st_read("path/to/shapefile.shp")
galah_call() |>
  galah_identify("vulpes") |>
  galah_geolocate(location) |>
  atlas_counts()

# Search for records within the bounding box of a shapefile
location <- sf::st_read("path/to/shapefile.shp")
galah_call() |>
  galah_identify("vulpes") |>
  galah_geolocate(location, type = "bbox") |>
  atlas_counts()

## End(Not run)

# Search for records within a polygon using an `sf` object
location <-
"POLYGON((143.32 -18.78,145.30 -20.52,141.52 -21.50,143.32 -18.78))" |>
 sf::st_as_sfc()
galah_call() |>
  galah_identify("reptilia") |>
  galah_polygon(location) |>
  atlas_counts()

# Search for records using a Well-known Text string (WKT)
wkt <- "POLYGON((142.36228 -29.00703,
                 142.74131 -29.00703,
                 142.74131 -29.39064,
                 142.36228 -29.39064,
                 142.36228 -29.00703))"
galah_call() |>
  galah_identify("vulpes") |>
  galah_geolocate(wkt) |>
  atlas_counts()

# Search for records within the bounding box extracted from an `sf` object
location <-
"POLYGON((143.32 -18.78,145.30 -20.52,141.52 -21.50,143.32 -18.78))" |>
 sf::st_as_sfc()
galah_call() |>
  galah_identify("vulpes") |>
  galah_geolocate(location, type = "bbox") |>
  atlas_counts()
```

```
# Search for records using a bounding box of coordinates
b_box <- sf::st_bbox(c(xmin = 143, xmax = 148, ymin = -29, ymax = -28),
                     crs = sf::st_crs("WGS84"))
galah_call() |>
  galah_identify("reptilia") |>
  galah_geolocate(b_box, type = "bbox") |>
  atlas_counts()

# Search for records using a bounding box in a `tibble` or `data.frame`
b_box <- tibble::tibble(xmin = 148, ymin = -29, xmax = 143, ymax = -21)
galah_call() |>
  galah_identify("vulpes") |>
  galah_geolocate(b_box, type = "bbox") |>
  atlas_counts()
```

`galah_group_by` *Specify fields to group when downloading record counts*

Description

`atlas_counts` supports server-side grouping of data. Grouping can be used to return record counts grouped by multiple, valid fields (found by `search_all(fields)`. Use `galah_group_by` when using the `group_by` argument of `atlas_counts` to return record counts summed by one or more valid fields.

Usage

```
galah_group_by(..., expand = TRUE)
```

Arguments

`...`	zero or more individual column names to include
`expand`	`logical`: When passed to `group_by` argument of `atlas_counts`, should factor levels be expanded? Defaults to `TRUE`.

Value

If any arguments are provided, returns a `data.frame` with columns `name` and `type`, as per `galah_select()`; if no arguments are provided, returns `NULL`.

See Also

`galah_select()`, `galah_filter()` and `galah_geolocate()` for related methods.

Examples

```
galah_call() |>
  galah_group_by(basisOfRecord) |>
  atlas_counts()
```

`galah_identify` *Narrow a query by passing taxonomic identifiers*

Description

When conducting a search or creating a data query, it is common to identify a known taxon or group of taxa to narrow down the records or results returned.

Usage

```
galah_identify(..., search = TRUE)
```

Arguments

`...`	one or more scientific names (if `search = TRUE`) or taxonomic identifiers (if `search = FALSE`); or an object of class `ala_id` (from `search_taxa`).
`search`	(logical); should the results in question be passed to `search_taxa`?

Details

`galah_identify()` is used to identify taxa you want returned in a search or a data query. Users to pass scientific names or taxonomic identifiers with pipes to provide data only for the biological group of interest.

It is good to use `search_taxa()` and `search_identifiers()` first to check that the taxa you provide to `galah_identify()` return the correct results.

Value

A tibble containing identified taxa.

See Also

`search_taxa()` to find identifiers from scientific names; `search_identifiers()` for how to get names if taxonomic identifiers are already known.

Examples

```
# Specify a taxon. A valid taxon will return an identifier.
galah_identify("reptilia")

# Specify more than one taxon at a time.
galah_identify("reptilia", "mammalia", "aves", "pisces")

# Use `galah_identify()` to narrow your queries
galah_call() |>
  galah_identify("Eolophus") |>
  atlas_counts()

# If you know a valid taxon identifier, add it and set `search = FALSE`.
galah_call() |>
 galah_identify("https://biodiversity.org.au/afd/taxa/009169a9-a916-40ee-866c-669ae0a21c5c",
                search = FALSE) |>
  atlas_counts()
```

---

galah_select *Specify fields for occurrence download*

Description

The living atlases store content in hundreds of different fields, and users often require thousands or millions of records at a time. To reduce time taken to download data, and limit complexity of the resulting `data.frame`, it is sensible to restrict the fields returned by `atlas_occurrences()`. This function allows easy selection of fields, or commonly-requested groups of columns, following syntax shared with `dplyr::select()`.

Usage

```
galah_select(..., group = c("basic", "event", "media", "assertions"))
```

Arguments

`...`	zero or more individual column names to include
`group`	`string`: (optional) name of one or more column groups to include. Valid options are `"basic"`, `"event"` and `"assertions"`

Details

The full list of available fields can be viewed with `show_all(fields)`.

Calling the argument `group = "basic"` returns the following columns:

- `decimalLatitude`
- `decimalLongitude`

- eventDate
- scientificName
- taxonConceptID
- recordID
- dataResourceName
- occurrenceStatus

Using group = "event" returns the following columns:

- eventRemarks
- eventTime
- eventID
- eventDate
- samplingEffort
- samplingProtocol

Using group = "media" returns the following columns:

- multimedia
- multimediaLicence
- images
- videos
- sounds

Using group = "assertions" returns all quality assertion-related columns. The list of assertions is shown by show_all_assertions().

Value

A tibble specifying the name and type of each column to include in the call to atlas_counts() or atlas_occurrences().

See Also

search_taxa(), galah_filter() and galah_geolocate() for other ways to restrict the information returned by atlas_occurrences() and related functions; atlas_counts() for how to get counts by levels of variables returned by galah_select; show_all(fields) to list available fields.

Examples

```
## Not run:
# Download occurrence records of *Perameles*,
# Only return scientificName and eventDate columns
galah_config(email = "your-email@email.com")
galah_call() |>
  galah_identify("perameles")|>
  galah_select(scientificName, eventDate) |>
```

```
  atlas_occurrences()

# Only return the "basic" group of columns and the basisOfRecord column
galah_call() |>
  galah_identify("perameles") |>
  galah_select(basisOfRecord, group = "basic") |>
  atlas_occurrences()

## End(Not run)
```

`group_by.data_request` *Specify fields to group when downloading record counts*

Description

`atlas_counts` supports server-side grouping of data. Grouping can be used to return record counts grouped by multiple, valid fields (found by `search_all(fields)`. Use `galah_group_by` when using the `group_by` argument of `atlas_counts` to return record counts summed by one or more valid fields. **[Experimental]**

Usage

```
## S3 method for class 'data_request'
group_by(.data, ...)
```

Arguments

`.data`	An object of class `data_request`
`...`	zero or more individual column names to include

Value

If any arguments are provided, returns a `data.frame` with columns `name` and `type`, as per `galah_select()`; if no arguments are provided, returns `NULL`.

See Also

`galah_group_by()`, with which this function is synonymous.

`identify.data_request` *Narrow a query by passing taxonomic identifiers*

Description

When conducting a search or creating a data query, it is common to identify a known taxon or group of taxa to narrow down the records or results returned.

Usage

```
## S3 method for class 'data_request'
identify(x, ..., search = TRUE)
```

Arguments

`x`	An object of class `data_request`, created using `galah_call()`
`...`	one or more scientific names (if `search = TRUE`) or taxonomic identifiers (if `search = FALSE`); or an object of class `ala_id` (from `search_taxa`).
`search`	(logical); should the results in question be passed to `search_taxa`?

Details

`galah_identify()` is used to identify taxa you want returned in a search or a data query. Users to pass scientific names or taxonomic identifiers with pipes to provide data only for the biological group of interest.

It is good to use `search_taxa()` and `search_identifiers()` first to check that the taxa you provide to `galah_identify()` return the correct results.

Value

A tibble containing identified taxa.

See Also

`galah_identify()`, with which this function is synonymous; `search_taxa()` to find identifiers from scientific names; `search_identifiers()` for how to get names if taxonomic identifiers are already known.

search_all *Search for record information*

Description

The living atlases store a huge amount of information, above and beyond the occurrence records that are their main output. In galah, one way that users can investigate this information is by searching for a specific option or category for the type of information they are interested in. Functions prefixed with search_ do this, displaying any matches to a search term within the valid options for the information specified by the suffix.

[Experimental] search_all() is a helper function that can do searches within multiple types of information from search_ sub-functions. See Details (below) for accepted values.

Usage

```
search_all(type, query)

search_apis(query)

search_assertions(query)

search_atlases(query)

search_collections(query)

search_datasets(query)

search_providers(query)

search_fields(query)

search_licences(query)

search_reasons(query)

search_ranks(query)

search_profiles(query)

search_lists(query)
```

Arguments

type	A string to specify what type of parameters should be searched.
query	A string specifying a search term. Searches are not case-sensitive.

Details

There are five categories of information, each with their own specific sub-functions to look-up each type of information. The available types of information for `search_all()` are:

Category	Type	Description	Sub-fun
configuration	`atlases`	Search for what atlases are available	`search_`
	`apis`	Search for what APIs & functions are available for each atlas	`search_`
	`reasons`	Search for what values are acceptable as 'download reasons' for a specified atlas	`search_`
taxonomy	`taxa`	Search for one or more taxonomic names	`search_`
	`identifiers`	Take a universal identifier and return taxonomic information	`search_`
	`ranks`	Search for valid taxonomic ranks (e.g. Kingdom, Class, Order, etc.)	`search_`
filters	`fields`	Search for fields that are stored in an atlas	`search_`
	`assertions`	Search for results of data quality checks run by each atlas	`search_`
	`licenses`	Search for copyright licences applied to media	`search_`
group filters	`profiles`	Search for what data profiles are available	`search_`
	`lists`	Search for what species lists are available	`search_`
data providers	`providers`	Search for which institutions have provided data	`search_`
	`collections`	Search for the specific collections within those institutions	`search_`
	`datasets`	Search for the data groupings within those collections	`search_`

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) containing all data that match the search query.

References

- Darwin Core terms https://dwc.tdwg.org/terms/
- ALA fields https://api.ala.org.au/#ws72
- ALA assertions fields https://api.ala.org.au/#ws81

See Also

See `search_taxa()` and `search_identifiers()` for more information on taxonomic searches. Use the `show_all()` function and `show_all_()` sub-functions to show available options of information. These functions are used to pass valid arguments to `galah_select()`, `galah_filter()`, and related functions.

Examples

```
# Search for fields that include the word "date"
search_all(fields, "date")

# Search for fields that include the word "marine"
search_all(fields, "marine")

# Search using a single taxonomic term
# (see `?search_taxa()` for more information)
```

```
search_all(taxa, "Reptilia") # equivalent

# Look up a unique taxon identifier
# (see `?search_identifiers()` for more information)
search_all(identifiers,
           "https://id.biodiversity.org.au/node/apni/2914510")

# Search for species lists that match "endangered"
search_all(lists, "endangered") # equivalent

# Search for a valid taxonomic rank, "subphylum"
search_all(ranks, "subphylum")
```

`search_identifiers` *Search for taxa with taxonomic identifiers*

Description

In the ALA, all records are associated with an identifier that uniquely identifies the taxon to which that record belongs. Once those identifiers are known, this function allows you to use them to look up further information on the taxon in question. Effectively this is the inverse function to `search_taxa()`, which takes names and provides identifiers. The resulting `tibble` of taxonomic information can also be passed to `galah_identify()` to filter queries to the specified taxon or taxa.

Usage

```
search_identifiers(query)
```

Arguments

`query`	`string`: A vector containing one or more taxonomic identifiers, given as strings.

Value

An object of class `tbl_df`, `data.frame` (aka a tibble) and `ala_id` containing taxonomic information.

See Also

`search_taxa()` for how to find species by (scientific) names. `galah_identify()`, `galah_select()`, `galah_filter()` and `galah_geolocate()` for other ways to restrict the information returned by `atlas_occurrences()` and related functions.

Examples

```
# Look up a unique taxon identifier
search_identifiers(query = "https://id.biodiversity.org.au/node/apni/2914510")
```

search_media *Search for associated media of occurrence records*

Description

Search for media files for a set of occurrence records downloaded using `atlas_occurrences()`. `search_media()` also accepts a set of media IDs (parsed or unparsed).

Usage

```
search_media(df)
```

Arguments

`df`	A `tibble` of species occurrence records or media IDs.

Value

a `tibble` of matching media files of occurrence records or media ids

Examples

```
## Not run:
# Search for media files for a set of species occurrence records
occs <- galah_call() |>
  galah_identify("perameles") |>
  galah_filter(year == 2001) |>
  atlas_occurrences()

search_media(occs)

## End(Not run)
```

search_taxa *Look up taxon information*

Description

Look up taxonomic names before downloading data from the ALA, using `atlas_occurrences()`, `atlas_species()` or `atlas_counts()`. Taxon information returned by `search_taxa()` may be passed to `galah_identify()` to provide the `identify` argument of `atlas_` functions. `search_taxa()` allows users to disambiguate homonyms (i.e. where the same name refers to taxa in different clades) prior to downloading data.

Usage

```
search_taxa(...)
```

Arguments

...	: A string of one or more scientific names, separated by commas, or a data frame specifying taxonomic levels. Note that searches are not case-sensitive.

Details

Users can also specify taxonomic levels in a search using a data frame (tibble). Taxa may be specified using either the `specificEpithet` argument to designate the second element of a Latin binomial, or the `scientificName` argument to specify the scientific name (which may include the subspecific epithet if required).

Value

An object of class `tbl_df`, `data.frame` (aka a tibble) and `ala_id` containing taxonomic information.

See Also

`search_identifiers()` for how to get names if taxonomic identifiers are already known. `galah_identify()`, `galah_select()`, `galah_filter()`, and `galah_geolocate()` for ways to restrict the information returned by `atlas_occurrences()` and related functions. `atlas_taxonomy()` to look up taxonomic trees.

Examples

```
# Search using a single string.
# Note that `search_taxa()` isn't case sensitive
search_taxa("Reptilia")

# Search using multiple strings.
# `search_taxa()` will return one row per taxon
search_taxa("reptilia", "mammalia")

# Specify taxonomic levels in a tibble using "specificEpiphet"
search_taxa(tibble::tibble(
  class = "aves",
  family = "pardalotidae",
  genus = "pardalotus",
  specificEpithet = "punctatus"))

# Specify taxonomic levels in a tibble using "scientificName"
search_taxa(tibble::tibble(
  family = c("pardalotidae", "maluridae"),
  scientificName = c("Pardalotus striatus striatus", "malurus cyaneus")))

# `galah_identify()` uses `search_taxa()` to narrow data queries
taxa <- search_taxa("reptilia", "mammalia")
```

```
galah_call() |>
  galah_identify(taxa) |>
  atlas_counts()
```

`select.data_request` *Specify fields for occurrence download*

Description

The living atlases store content in hundreds of different fields, and users often require thousands or millions of records at a time. To reduce time taken to download data, and limit complexity of the resulting `data.frame`, it is sensible to restrict the fields returned by `atlas_occurrences()`. This function allows easy selection of fields, or commonly-requested groups of columns, following syntax shared with `dplyr::select()`.

Usage

```
## S3 method for class 'data_request'
select(.data, ..., group = c("basic", "event", "media", "assertions"))
```

Arguments

`.data`	An object of class `data_request`, created using `galah_call()`
`...`	zero or more individual column names to include
`group`	`string`: (optional) name of one or more column groups to include. Valid options are `"basic"`, `"event"` and `"assertions"`

Details

The full list of available fields can be viewed with `show_all(fields)`. **[Experimental]**

Calling the argument `group = "basic"` returns the following columns:

- `decimalLatitude`
- `decimalLongitude`
- `eventDate`
- `scientificName`
- `taxonConceptID`
- `recordID`
- `dataResourceName`
- `occurrenceStatus`

Using `group = "event"` returns the following columns:

- `eventRemarks`

- eventTime
- eventID
- eventDate
- samplingEffort
- samplingProtocol

Using group = "media" returns the following columns:

- multimedia
- multimediaLicence
- images
- videos
- sounds

Using group = "assertions" returns all quality assertion-related columns. The list of assertions is shown by show_all_assertions().

Value

A tibble specifying the name and type of each column to include in the call to atlas_counts() or atlas_occurrences().

See Also

galah_select(), with which this function is synonymous.

---

show_all *Show valid record information*

Description

The living atlases store a huge amount of information, above and beyond the occurrence records that are their main output. In galah, one way that users can investigate this information is by showing all the available options or categories for the type of information they are interested in. Functions prefixed with show_all_ do this, displaying all valid options for the information specified by the suffix.

[Experimental] show_all() is a helper function that can display multiple types of information from show_all_ sub-functions. See Details (below) for accepted values.

Usage

```
show_all(type, limit = NULL)

show_all_assertions(limit = NULL)

show_all_atlases(limit = NULL)

show_all_cached_files(limit = NULL)

show_all_apis(limit = NULL)

show_all_collections(limit = NULL)

show_all_datasets(limit = NULL)

show_all_providers(limit = NULL)

show_all_fields(limit = NULL)

show_all_licences(limit = NULL)

show_all_reasons(limit = NULL)

show_all_ranks(limit = NULL)

show_all_profiles(limit = NULL)

show_all_lists(limit = NULL)
```

Arguments

type	A string to specify what type of parameters should be shown.
limit	Optional number of values to return. Defaults to NULL, i.e. all records

Details

There are five categories of information, each with their own specific sub-functions to look-up each type of information. The available types of information for show_all_ are:

Category	Type	Description	Sub-function
configuration	atlases	Show what atlases are available	show_all_at
	apis	Show what APIs & functions are available for each atlas	show_all_ap
	reasons	Show what values are acceptable as 'download reasons' for a specified atlas	show_all_re
taxonomy	ranks	Show valid taxonomic ranks (e.g. Kingdom, Class, Order, etc.)	show_all_ra
filters	fields	Show fields that are stored in an atlas	show_all_fi
	assertions	Show results of data quality checks run by each atlas	show_all_as
	licenses	Show what copyright licenses are applied to media	show_all_li
group filters	profiles	Show what data profiles are available	show_all_pr

	lists	Show what species lists are available	show_all_li
data providers	providers	Show which institutions have provided data	show_all_pr
	collections	Show the specific collections within those institutions	show_all_co
	datasets	Shows all the data groupings within those collections	show_all_da

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) containing all data of interest.

References

- Darwin Core terms https://dwc.tdwg.org/terms/
- ALA fields https://api.ala.org.au/#ws72
- ALA assertions fields https://api.ala.org.au/#ws81

See Also

Use the `search_all()` function and `search_()` sub-functions to search for information. These functions are used to pass valid arguments to `galah_select()`, `galah_filter()`, and related functions.

Examples

```
# See all supported atlases
show_all(atlases)

# Show a list of all available data quality profiles
show_all(profiles)

# Show a listing of all accepted reasons for downloading occurrence data
show_all(reasons)

# Show a listing of all taxonomic ranks
show_all(ranks)
```

---

show_values *Show or search for values within a specified field*

Description

Users may wish to see the specific values *within* a chosen field, profile or list to narrow queries or understand more about the information of interest. `show_values()` provides users with these values. `search_values()` allows users for search for specific values within a specified field.

Usage

```
show_values(df)

search_values(df, query)
```

Arguments

df	A search result from search_fields(), search_profiles() or search_lists().
query	A string specifying a search term. Not case sensitive.

Details

Each **Field** contains categorical or numeric values. For example:

- The field "year" contains values 2021, 2020, 2019, etc.
- The field "stateProvince" contains values New South Wales, Victoria, Queensland, etc. These are used to narrow queries with galah_filter().

Each **Profile** consists of many individual quality filters. For example, the "ALA" profile consists of values:

- Exclude all records where spatial validity is FALSE
- Exclude all records with a latitude value of zero
- Exclude all records with a longitude value of zero

Each **List** contains a list of species, usually by taxonomic name. For example, the Endangered Plant species list contains values:

- Acacia curranii (Curly-bark Wattle)
- Brachyscome papillosa (Mossgiel Daisy)
- Solanum karsense (Menindee Nightshade)

Value

A tibble of values for a specified field, profile or list.

Examples

```
# Show values in field 'cl22'
search_fields("cl22") |>
  show_values()

# Search for any values in field 'cl22' that match 'tas'
search_fields("cl22") |>
  search_values("tas")

# See items within species list "dr19257"
search_lists("dr19257") |>
  show_values()
```

slice_head.data_request *Slice for object of class* data_request

Description

[Experimental] This is a simple function to set the 'limit' argument in `atlas_counts()` using `dplyr` syntax.

Usage

```
## S3 method for class 'data_request'
slice_head(.data, ..., n, prop)
```

Arguments

`.data`	An object of class `data_request`, created using `galah_call()`
`...`	currently ignored
`n`	The number of rows to be returned. If data are grouped (using group_by), this operation will be performed on each group.
`prop`	currently ignored

st_crop.data_request *Narrow a query to within a specified polygon*

Description

Restrict results to those from a specified area. Areas must be polygons. Polygons must be supplied as an `sf` object, a 'well-known text' (WKT) string, or a shapefile. Polygons and shapefiles must not be overly complex (i.e. have too many characters or too many vertices) or they will not be accepted in a query to the ALA.

`st_crop` is masked from `sf`, but when piped after `galah_call()`, is functionally synonymous with `galah_polygon()`

Usage

```
## S3 method for class 'data_request'
st_crop(x, y, ...)
```

Arguments

x	An object of class `data_request`, created using `galah_call()`
y	A single `sf` object, WKT string or shapefile
...	currently ignored

Details

[Experimental]

See Also

`galah_polygon()`, with which this function is synonymous.

Index