Package 'galah'

January 24, 2022

Type Package

Title Atlas of Living Australia (ALA) Data and Resources in R

Version 1.4.0

Description The Atlas of Living Australia ('ALA') provides tools to enable users of biodiversity information to find, access, combine and visualise data on Australian plants and animals. 'galah' enables the R community to directly access data and resources hosted by the ALA and other living atlases.

Depends R (>= 4.1.0)

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

Suggests vcr (>= 0.6.0), covr, dplyr, knitr, magrittr, pkgdown, rmarkdown, taxize, testthat

License MPL-2.0

URL https://github.com/AtlasOfLivingAustralia/galah, https://atlasoflivingaustralia.github.io/galah/

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

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

LazyLoad yes

VignetteBuilder knitr

RoxygenNote 7.1.2

Encoding UTF-8

NeedsCompilation no

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

Repository CRAN

Date/Publication 2022-01-24 10:52:46 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.

`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,
  group_by = NULL,
  limit = 100,
  type = c("record", "species"),
  refresh_cache = FALSE
)
```

Arguments

`request`	optional `data_rquest` 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()`
`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_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

With no arguments, return the total number of records in the ALA

```
atlas_counts()
#> # A tibble: 1 x 1
#>       count
#>       <int>
#> 1 102070026
```

You can group counts by state and territory with `galah_group_by`

```
galah_call() |>
  galah_group_by(stateProvince) |>
  atlas_counts()
#> # A tibble: 100 x 2
#>   stateProvince      count
#>   <chr>              <int>
#> 1 New South Wales 24938082
#> 2 Victoria        21775800
#> 3 Queensland      17550396
#> 4 South Australia  8449336
#> # ... with 96 more rows
```

You can add a filter to narrow your search

```
galah_call() |>
  galah_filter(basisOfRecord == "FossilSpecimen")
#> An object of type `data_request` containing:
#>
#> $filter
#> # A tibble: 1 x 4
#>   variable      logical value          query
#>   <chr>         <chr>   <chr>          <chr>
#> 1 basisOfRecord ==      FossilSpecimen "(basisOfRecord:\"FossilSpecimen\")"
  atlas_counts()
#> # A tibble: 1 x 1
#>       count
#>       <int>
#> 1 102070026
```

Specify `type = species` to count the number of species, and group record counts by kingdom

```
records <- galah_call() |>
  galah_group_by(kingdom) |>
  atlas_counts(type = "species")

records
#> # A tibble: 10 x 2
#>    kingdom   count
#>    <chr>     <dbl>
#>  1 Animalia  90821
#>  2 Plantae   39883
#>  3 Fungi     16752
#>  4 Chromista  1822
#>  5 Protista    635
#>  6 Bacteria    525
#>  7 Protozoa    493
#>  8 Archaea       0
#>  9 Eukaryota     0
#> 10 Virus         0
```

Using `galah_group_by` allows you to cross-tabulate using two different variables, similar to using `dplyr::group_by() %>% dplyr::count()`

```
records <- galah_call() |>
  galah_filter(year > 2015) |>
  galah_group_by(year, basisOfRecord) |>
  atlas_counts()

records
#> # A tibble: 41 x 3
#>   basisOfRecord     year    count
#>   <chr>             <chr>   <int>
#> 1 HUMAN_OBSERVATION 2020  5825030
#> 2 HUMAN_OBSERVATION 2019  5401216
#> 3 HUMAN_OBSERVATION 2018  5267959
#> 4 HUMAN_OBSERVATION 2017  4348547
#> # ... with 37 more rows
```

`atlas_media` *Download 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` allows download of any and all of the media types.

Usage

```
atlas_media(
  request = NULL,
  identify = NULL,
  filter = NULL,
  geolocate = NULL,
  select = galah_select(group = "basic"),
  download_dir,
  refresh_cache = FALSE
)
```

Arguments

request	optional data_rquest 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()
select	data.frame: generated by a call to galah_select()
download_dir	string: path to directory to store the downloaded media in
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

atlas_occurrences() works by first finding all occurrence records matching the filter which contain media, then downloading the metadata for the media and the media files. galah_filter() can take both filter relating to occurrences (e.g. basis of records), and filter relating to media (e.g. type of licence). 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 downloaded media

Examples

Download Regent Honeyeater multimedia

```
media_data <- galah_call() |>
    galah_identify("Regent Honeyeater") |>
    galah_filter(year == 2011) |>
    atlas_media(download_dir = "folder/your-directory")
```

Specify a single media type to download

```
media_data <- galah_call(
     galah_identify("Eolophus Roseicapilla") |>
     galah_filter(multimedia == "Sound") |>
     atlas_media()
```

Filter to only records with a particular licence type (DOESN'T WORK)

```
media_data <- galah_call()
      galah_identify("Ornithorhynchus anatinus") |>
      galah_filter(year == 2020,
        license = "http://creativecommons.org/licenses/by-nc/4.0/") |>
       atlas_media()
)
```

You might also want to check how many records have media files before you download them. Do this with `atlas_counts()`

```
galah_call() |>
     galah_filter(multimedia == c("Image","Sound","Video")) |>
     galah_group_by(multimedia) |>
     atlas_counts()
```

See Also

`atlas_counts()` to find the number of records with media- 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).

`atlas_occurrences` *Occurrence records*

Description

The most common form of data stored by ALA 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,
  select = galah_select(group = "basic"),
  mint_doi = FALSE,
  doi = NULL,
  refresh_cache = FALSE
)
```

Arguments

request	optional data_rquest 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()
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	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

Search for occurrences matching a taxon identifier

```
galah_config(email = "your-email@email.com")
galah_call() |>
  galah_identify("Reptilia") |>
  atlas_occurrences()
```

Search for occurrences in a year range

```
galah_call() |>
  galah_filter(year >= 2010, year <= 2020) |>
  atlas_occurrences()
```

Search for occurrences in a WKT-specified area

```
polygon <- "POLYGON((146.24960 -34.05930,146.37045 -34.05930,146.37045 -34.152549,146.24960 -34.15254
galah_call() |>
  galah_geolocate(polygon) |>
  atlas_occurrences()
```

You can also download occurrence records by piping with %>% if you prefer.

```
galah_call() %>%
  galah_identify("Reptilia") %>%
  galah_filter(year >= 2010) %>%
  galah_geolocate(polygon) %>%
  atlas_occurrences()
```

`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,
  refresh_cache = FALSE
)
```

Arguments

`request`	optional `data_rquest` 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()`
`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, look up a genus of interest in the ALA with search_taxa()

```
search_taxa("Heleioporus")
#> # A tibble: 1 x 13
#>   search_term scientific_name scientific_name_~ taxon_concept_id  rank  match_type kingdom phylum cl
#>   <chr>      <chr>           <chr>              <chr>            <chr> <chr>      <chr>   <chr>  <chr> <chr> <chr>
#> 1 Heleioporus Heleioporus    Gray, 1841         urn:lsid:biodiver~ genus exactMatch Animal~ Chord~ Amph~
#> # ... with 1 more variable: issues <chr>
```

It's a good idea to find how many species there are for the taxon you are interested in - in our case, genus *Heleioporus* - with atlas_counts()

```
galah_call() |>
  galah_identify("Heleioporus") |>
  atlas_counts(type = "species")
#> # A tibble: 1 x 1
#>   count
#>   <int>
#> 1     6
```

Now get taxonomic information on all species within this genus with atlas_species()

```
# (every row is a species with associated taxonomic data)
galah_call() |>
  galah_identify("Heleioporus") |>
  atlas_species()
#> # A tibble: 6 x 10
#>   kingdom phylum  class   order family       genus     species    author   species_guid       verna
#>   <chr>   <chr>   <chr>   <chr> <chr>        <chr>     <chr>      <chr>    <chr>              <chr>
#> 1 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Gray, 1~ urn:lsid:biodiv
#> 2 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Shaw & ~ urn:lsid:biodiv
#> 3 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ Gray, 18~ urn:lsid:biodiv
#> 4 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Lee & M~ urn:lsid:biodiv
#> 5 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Lee & M~ urn:lsid:biodiv
#> 6 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ Lee, 1967 urn:lsid:biodiv
```

You can also get taxonomic information on species by piping with %>% or |>. Just begin your query with galah_call()

```
galah_call() |>
  galah_identify("Heleioporus") |>
  atlas_species()
#> # A tibble: 6 x 10
#>  kingdom phylum  class   order family       genus     species    author   species_guid       verna
#>  <chr>   <chr>   <chr>   <chr> <chr>        <chr>     <chr>      <chr>    <chr>              <chr>
#> 1 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Gray, 1~ urn:lsid:biodiv
#> 2 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Shaw & ~ urn:lsid:biodiv
#> 3 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ Gray, 18~ urn:lsid:biodiv
#> 4 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Lee & M~ urn:lsid:biodiv
#> 5 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ (Lee & M~ urn:lsid:biodiv
#> 6 Animalia Chordata Amphibia Anura Limnodynastidae Heleioporus Heleioporu~ Lee, 1967 urn:lsid:biodiv
```

`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_rquest` 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

Examples

Get a taxonomic tree of *Chordata* down to the class level

```
galah_call() |>
  galah_identify("chordata") |>
  galah_down_to(class) |>
  atlas_taxonomy()
#>                              levelName
#> 1  Chordata
#> 2   ¦--Cephalochordata
#> 3   ¦   °--Amphioxi
#> 4   ¦--Craniata
#> 5   ¦   °--Agnatha
#> 6   ¦       ¦--Cephalasipidomorphi
#> 7   ¦       °--Myxini
#> 8   ¦--Tunicata
#> 9   ¦   ¦--Appendicularia
#> 10  ¦   ¦--Ascidiacea
#> 11  ¦   °--Thaliacea
#> 12  °--Vertebrata
#> 13      °--Gnathostomata
#> 14          ¦--Amphibia
#> 15          ¦--Aves
#> 16          ¦--Mammalia
#> 17          ¦--Pisces
#> 18          ¦   ¦--Actinopterygii
#> 19          ¦   ¦--Chondrichthyes
#> 20          ¦   ¦--Cephalaspidomorphi
#> 21          ¦   °--Sarcopterygii
#> 22          °--Reptilia
```

Get a taxonomic tree of *Fungi* down to the phylum level

```
galah_call() |>
  galah_identify("fungi") |>
  galah_down_to(class) |>
  atlas_taxonomy()
#>                               levelName
```

```
#> 1  Fungi
#> 2   ¦--Ascomycota
#> 3   ¦   ¦--Ascomycetes
#> 4   ¦   ¦--Discomycetes
#> 5   ¦   ¦--Pezizomycotina
#> 6   ¦   ¦   ¦--Arthoniomycetes
#> 7   ¦   ¦   ¦--Dothideomycetes
#> 8   ¦   ¦   ¦--Eurotiomycetes
#> 9   ¦   ¦   ¦--Geoglossomycetes
#> 10  ¦   ¦   ¦--Laboulbeniomycetes
#> 11  ¦   ¦   ¦--Lecanoromycetes
#> 12  ¦   ¦   ¦--Leotiomycetes
#> 13  ¦   ¦   ¦--Lichinomycetes
#> 14  ¦   ¦   ¦--Orbiliomycetes
#> 15  ¦   ¦   ¦--Pezizomycetes
#> 16  ¦   ¦   ¦--Sordariomycetes
#> 17  ¦   ¦   °--Xylonomycetes
#> 18  ¦   ¦--Saccharomycotina
#> 19  ¦   ¦   °--Saccharomycetes
#> 20  ¦   ¦--Taphrinomycotina
#> 21  ¦   ¦   ¦--Neolectomycetes
#> 22  ¦   ¦   ¦--Pneumocystidomycetes
#> 23  ¦   ¦   ¦--Schizosaccharomycetes
#> 24  ¦   ¦   °--Taphrinomycetes
#> 25  ¦   °--Pyrenomycete
#> 26  ¦--Basidiomycota
#> 27  ¦   ¦--Agaricomycotina
#> 28  ¦   ¦   ¦--Agaricomycetes
#> 29  ¦   ¦   ¦--Dacrymycetes
#> 30  ¦   ¦   °--Tremellomycetes
#> 31  ¦   ¦--Basidiomycetes
#> 32  ¦   ¦--Entorrhizomycetes
#> 33  ¦   ¦--Pucciniomycotina
#> 34  ¦   ¦   ¦--Agaricostilbomycetes
#> 35  ¦   ¦   ¦--Atractiellomycetes
#> 36  ¦   ¦   ¦--Classiculomycetes
#> 37  ¦   ¦   ¦--Cryptomycocolacomycetes
#> 38  ¦   ¦   ¦--Cystobasidiomycetes
#> 39  ¦   ¦   ¦--Microbotryomycetes
#> 40  ¦   ¦   ¦--Pucciniomycetes
#> 41  ¦   ¦   ¦--Spiculogloeomycetes
#> 42  ¦   ¦   ¦--Tritirachiomycetes
#> 43  ¦   ¦   °--Urediniomycetes
#> 44  ¦   ¦--Ustilaginomycotina
#> 45  ¦   ¦   ¦--Exobasidiomycetes
#> 46  ¦   ¦   ¦--Malasseziomycetes
#> 47  ¦   ¦   ¦--Monilielliomycetes
#> 48  ¦   ¦   ¦--Ustilaginomycetes
```

```
#> 49  ¦   ¦   °--Ustomycetes
#> 50  ¦   ¦--Wallemiomycetes
#> 51  ¦   °--Wallemiomycotina
#> 52  ¦       °--Wallemiomycetes
#> 53  ¦--Chytridiomycota
#> 54  ¦   ¦--Blastocladiomycetes
#> 55  ¦   ¦--Chytridiomycetes
#> 56  ¦   ¦--Monoblepharidomycetes
#> 57  ¦   °--Neocallimastigomycetes
#> 58  ¦--Glomeromycota
#> 59  ¦   °--Glomeromycetes
#> 60  ¦--Microspora
#> 61  ¦   °--Microsporea
#> 62  °--Zygomycota
#> 63      ¦--Entomophthoromycotina
#> 64      ¦   ¦--Basidiobolomycetes
#> 65      ¦   ¦--Entomophthoromycetes
#> 66      ¦   °--Neozygitomycetes
#> 67      ¦--Trichomycetes
#> 68      °--Zygomycetes
```

See Also

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

`clear_cached_files` *Clear previously cached files*

Description

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

Usage

```
clear_cached_files()
```

Examples

First, set caching to true with `galah_config()`. Then create a data query. The data you download will be cached in a temporary directory.

```
galah_config(caching = TRUE)
dat <- atlas_counts(group_by = galah_group_by(year))
```

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

```
clear_cached_files()
```

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)
```

Arguments

...	Options can be defined using the form name = value. Valid arguments are: • atlas string: Living Atlas to point to, Australia by default • 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 ALA at this address. A registered email is required for some functions in galah. • 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. • verbose logical: should galah give verbose output to assist debugging? Defaults to FALSE. • 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.
profile_path	string: (optional), path to a directory to store config values in. If provided, config values will be written to a new or existing .Rprofile file for future sessions. NULL by default.

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

To configure your session to allow you to download occurrence records, enter your email in `galah_config()`. This email should be registered with the ALA, which you can do here

```
galah_config(email = "your-email@email.com")
```

Turn on caching in your session

```
galah_config(caching = FALSE)
```

It is required by some ALA services that you add a reason for downloading data. To look up all valid reasons to enter, use `show_all_reasons()`

```
show_all_reasons()
#> # A tibble: 13 x 2
#>       id name
#>    <int> <chr>
#>  1     0 conservation management/planning
#>  2     1 biosecurity management/planning
#>  3     2 environmental assessment
#>  4     3 education
#>  5     4 scientific research
#>  6     5 collection management
#>  7     6 other
#>  8     7 ecological research
#>  9     8 systematic research/taxonomy
#> 10    10 testing
#> 11    11 citizen science
#> 12    12 restoration/remediation
#> 13    13 species modelling
```

Add your selected reason using the option `download_reason_id`

```
galah_config(download_reason_id = 0)
```

You can also make debugging in your session easier by setting `verbose = TRUE`

```
galah_config(download_reason_id = 0,
             verbose = TRUE)
```

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

Description

`atlas_taxonomy` generates a downwards search of the taxonomic tree. This function can be used to specify the name of a valid taxonomic rank using non-standard evaluation (NSE), for consistency with other `galah_` functions.

Usage

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

Arguments

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

Value

A string with the named rank

Examples

An example of using `galah_down_to()` with `atlas_taxonomy()`. Return a taxonomic tree of *Chordata* down to the class level

```
galah_call() |>
    galah_identify("Chordata") |>
    galah_down_to(class) |>
    atlas_taxonomy()
#>                              levelName
#> 1  Chordata
#> 2   ¦--Cephalochordata
#> 3   ¦   °--Amphioxi
#> 4   ¦--Craniata
#> 5   ¦   °--Agnatha
#> 6   ¦       ¦--Cephalasipidomorphi
#> 7   ¦       °--Myxini
#> 8   ¦--Tunicata
#> 9   ¦   ¦--Appendicularia
#> 10  ¦   ¦--Ascidiacea
#> 11  ¦   °--Thaliacea
#> 12  °--Vertebrata
#> 13      °--Gnathostomata
#> 14          ¦--Amphibia
#> 15          ¦--Aves
#> 16          ¦--Mammalia
```

```
#> 17          ¦--Pisces
#> 18          ¦   ¦--Actinopterygii
#> 19          ¦   ¦--Chondrichthyes
#> 20          ¦   ¦--Cephalaspidomorphi
#> 21          ¦   °--Sarcopterygii
#> 22          °--Reptilia
```

Another example: return a taxonomic tree of *Cacatuidae* down to the genus level

```
galah_call() |>
  galah_identify("Cacatuidae") |>
  galah_down_to(genus) |>
  atlas_taxonomy()
#>                    levelName
#> 1  Cacatuidae
#> 2   ¦--Cacatuinae
#> 3   ¦   ¦--Cacatuini
#> 4   ¦   ¦   ¦--Cacatua
#> 5   ¦   ¦   ¦--Callocephalon
#> 6   ¦   ¦   ¦--Eolophus
#> 7   ¦   ¦   °--Lophochroa
#> 8   ¦   °--Microglossini
#> 9   ¦       °--Probosciger
#> 10  ¦--Calyptorhynchinae
#> 11  ¦   °--Calyptorhynchus
#> 12  °--Nymphicinae
#> 13      °--Nymphicus
```

See Also

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

`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 `filters` argument in `atlas_occurrences()`, `atlas_species()` or `atlas_counts()`. `galah_filter` uses non-standard evaluation (NSE), and is designed to be as compatible as possible with `dplyr::filter` syntax.

Usage

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

Arguments

...	filters, in the form field logical value
profile	string: (optional) a data quality profile to apply to the records. See show_all_profiles() for valid profiles. By default no profile is applied.

Details

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.

Value

An object of class data.frame and galah_filter, containing filter values.

Examples

Create a custom filter for records of interest

```
filters <- galah_filter(
    basisOfRecord == "HumanObservation",
    year >= 2010,
    stateProvince == "New South Wales")
```

Add the default ALA data quality profile

```
filters <- galah_filter(
    basisOfRecord == "HumanObservation",
    year >= 2020,
    stateProvince == "New South Wales",
    profile = "ALA")
```

Use filters to exclude particular values

```
filter <- galah_filter(year >= 2010 & year != 2021)

atlas_counts(filter = filter)
#> # A tibble: 1 x 1
#>      count
#>      <int>
#> 1 43916661
```

Separating statements with a comma is equivalent to an AND statement

```
galah_filter(year >= 2010 & year < 2020) # is the same as:
galah_filter(year >= 2010, year < 2020)
```

All statements must include the field name

```
galah_filter(year == 2010 | year == 2021) # this works (note double equals)
galah_filter(year == c(2010, 2021)) # same as above
galah_filter(year == 2010 | 2021) # this fails
```

It is possible to use an object to specify required values

```
# Numeric example

year_value <- 2010

galah_call() %>%
  galah_filter(year > year_value) %>%
  atlas_counts()
#> # A tibble: 1 x 1
#>      count
#>      <int>
#> 1 42816943


# Categorical example

basis_of_record <- c("HumanObservation", "MaterialSample")

galah_call() %>%
  galah_filter(basisOfRecord == basis_of_record) %>%
  atlas_counts()
#> # A tibble: 1 x 1
#>      count
#>      <int>
#> 1 82809464
```

solr supports range queries on text as well as numbers. The following queries all Australian States and Territories alphabetically after "Tasmania"

```
galah_call() %>%
  galah_filter(cl22 >= "Tasmania") %>%
  atlas_counts()
#> # A tibble: 1 x 1
#>      count
#>      <int>
#> 1 30230213
```

See Also

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

---

galah_geolocate *Narrow a query using a WKT string*

Description

Restrict results to those from a specified area. Areas must be polygons and be specified as either an sf object, or a 'well-known text' (wkt) string.

Usage

```
galah_geolocate(...)
```

Arguments

...	a single wkt string or sf object

Details

WKT strings longer than 10000 characters will not be accepted by the ALA- so the sf object or WKT string may need to be simplified.

Value

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

Examples

Search for records using a shapefile

```
galah_config(email = "your-email@email.com")

location <- galah_geolocate(st_read(path/to/shapefile))
atlas_occurrences(geolocate = location)
```

Search for records using a Well-known Text geometry (WKT)

```
wkt <- "POLYGON((142.36228 -29.00703,142.74131 -29.00703,142.74131 -29.39064,142.36228 -29.39064,142.

atlas_counts(geolocate = galah_geolocate(wkt))
#> # A tibble: 1 x 1
#>   count
#>   <int>
#> 1  3300
```

See Also

search_taxa(), galah_filter() and galah_select() for other ways to restrict the information returned by atlas_occurrences() and related functions.

---

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_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.

Examples

Return record counts since 2010 by year

```
records <- galah_call() |>
    galah_filter(year > 2010) |>
    galah_group_by(year) |>
    atlas_counts()

records
#> # A tibble: 12 x 2
#>    year    count
#>    <chr>   <int>
#>  1 2020  5843340
#>  2 2019  5506924
#>  3 2018  5418009
#>  4 2017  4648403
#>  5 2016  3844787
#>  6 2014  3767573
```

```
#>  7 2015  3605917
#>  8 2013  3505658
#>  9 2012  2933981
#> 10 2011  2539004
#> 11 2021  1161557
#> 12 2022    41790
```

Return record counts since 2010 by year and data provider

```
records <- galah_call() |>
    galah_filter(year > 2010) |>
    galah_group_by(year, dataResourceName) |>
    atlas_counts()
```

```
records
#> # A tibble: 1,048 x 3
#>   year  dataResourceName              count
#>   <chr> <chr>                         <int>
#> 1 2020  eBird Australia             4589800
#> 2 2020  iNaturalist Australia        671032
#> 3 2020  NSW BioNet Atlas             372617
#> 4 2020  Earth Guardians Weekly Feed   71783
#> # ... with 1,044 more rows
```

Return record counts of *Litoria* species each year since 2015, limiting results to the top 5 each year

```
records <- galah_call() |>
    galah_identify("Litoria") |>
    galah_filter(year > 2015) |>
    galah_group_by(year, species) |>
    atlas_counts(limit = 5)
```

```
records
#> # A tibble: 35 x 3
#>   year  species            count
#>   <chr> <chr>              <int>
#> 1 2018  Litoria peronii    10497
#> 2 2018  Litoria fallax      7013
#> 3 2018  Litoria caerulea    3073
#> 4 2018  Litoria verreauxii  2980
#> # ... with 31 more rows
```

See Also

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

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. This function allows users to pass scientific names or taxonomic identifiers with pipes to provide data only for the biological group of interest.

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), gbifid, or nbnid (from taxize) for international atlases.
search	(logical); should the results in question be passed to search_taxa? Ignored if an object of class ala_id, gbifid, or nbnid is given to ....

Examples

galah_identify() is used to identify taxa you want returned in a search or a data query. 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.

Specify a taxon. A valid taxon will return an identifier.

```
galah_identify("reptilia")
#> # A tibble: 1 x 1
#>   identifier
#>   <chr>
#> 1 urn:lsid:biodiversity.org.au:afd.taxon:682e1228-5b3c-45ff-833b-550efd40c399
```

Specify more than one taxon at a time.

```
galah_identify("reptilia", "mammalia", "aves", "pisces")
#> # A tibble: 4 x 1
#>   identifier
#>   <chr>
#> 1 urn:lsid:biodiversity.org.au:afd.taxon:682e1228-5b3c-45ff-833b-550efd40c399
#> 2 urn:lsid:biodiversity.org.au:afd.taxon:e9e7db31-04df-41fb-bd8d-e0b0f3c332d6
#> 3 urn:lsid:biodiversity.org.au:afd.taxon:5ed80139-31bb-48a8-9f57-42d8015dacbb
#> 4 urn:lsid:biodiversity.org.au:afd.taxon:e22efeb4-2cb5-4250-8d71-61c48bdaa051
```

Use galah_identify() to narrow your queries

```
galah_call() %>%
  galah_identify("Eolophus") %>%
  atlas_counts()
#> # A tibble: 1 x 1
#>    count
#>    <int>
#> 1 856571
```

If you already know a valid taxon identifier, add it and set `search = FALSE`.

```
galah_call() %>%
 galah_identify("urn:lsid:biodiversity.org.au:afd.taxon:b2de5e40-df8f-4339-827d-25e63454a4a2",
                search = FALSE) %>%
  atlas_counts()
#> # A tibble: 1 x 1
#>    count
#>    <int>
#> 1 856571
```

See Also

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

`galah_select` *Specify fields for occurrence download*

Description

The living atlases store content on 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", "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 `"assertion"`

Details

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

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

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

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

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

Value

An object of class `data.frame` and `galah_select` specifying the name and type of each column to include in the call to `atlas_counts()` or `atlas_occurrences()`.

Examples

Download occurrence records of *Perameles* taken in 2001, only returning scientific name and event date

```
galah_config(email = "your-email@email.com")
galah_call() |>
  galah_identify("perameles")|>
  galah_filter(year == 2001) |>
  galah_select(scientificName, eventDate) |>
  atlas_occurrences()
```

Download occurrence record of *Perameles* taken in 2001, returning the "basic" group of columns plus the Basis of Record

```
galah_call() |>
  galah_identify("perameles") |>
  galah_filter(year == 2001) |>
  galah_select(group = "basic", basisOfRecord) |>
  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.

---

search_fields *Query layers, fields or assertions by free text search*

Description

This function can be used to find relevant fields and/or layers for use in building a set of filters with galah_filter() or specifying required columns with galah_select(). This function returns a data.frame of all fields matching the type specified. Field names are in Darwin Core format, except in the case where the field is specific to the ALA database, in which case the ALA field name is returned.

Usage

```
search_fields(
  query,
  type = c("all", "fields", "layers", "assertions", "media", "other")
)

show_all_fields(
  type = c("all", "fields", "layers", "assertions", "media", "other")
)
```

Arguments

query	string: A search string. Not case sensitive.
type	string: What type of parameters should be searched? Should be one or more of fields, layers, assertions, media or all.

Details

Layers are the subset of fields that are spatially appended to each record by the ALA. Layer ids are comprised of a prefix: 'el' for environmental (gridded) layers and 'cl' for contextual (polygon) layers, followed by an id number.

Value

if query is missing, an empty data.frame; otherwise an object of class tbl_df and data.frame (aka a tibble) containing fields that match the search query.

An object of class tbl_df and data.frame (aka a tibble) with three columns:

- id: The identifier for that layer or field. This is the value that should be used when referring to a field in another function.

- description: Detailed information on a given field
- type: Whether the field is a `field` or `layer`
- link: For layers, a link to the source data (if available)

Examples

Search for all fields that use include the word "date"

```
search_fields("date")
#> # A tibble: 33 x 4
#>   id             description     type   link
#>   <chr>          <chr>           <chr>  <chr>
#> 1 dateIdentified Date Identified fields <NA>
#> 2 datePrecision  Date precision  fields <NA>
#> 3 eventDate      Event Date      fields <NA>
#> 4 eventDateEnd   <NA>            fields <NA>
#> # ... with 29 more rows
```

Search for all fields with the string "basisofrecord"

```
search_fields("basisofrecord")
#> # A tibble: 2 x 4
#>   id                description               type   link
#>   <chr>             <chr>                     <chr>  <chr>
#> 1 basisOfRecord     Record type               fields <NA>
#> 2 raw_basisOfRecord Record type (unprocessed) fields <NA>
```

Search for all fields that have information for "marine"

```
search_fields("marine") |>
  head() # only show first 5 results
#> # A tibble: 6 x 4
#>   id      description                                                type   link
#>   <chr>   <chr>                                                      <chr>  <chr>
#> 1 cl2105  "FAO Fishery Statistical Areas FAO Fishing areas (Mari~ layers "http://www.fao.org/geonetwo
#> 2 cl10948 "CAPAD 2018 Terrestrial The Collaborative Australian P~ layers "http://www.environment.gov
#> 3 cl10957 "CAPAD 2018 Marine The Collaborative Australian Protec~ layers "http://www.environment.gov
#> 4 el1056  "Endemism (Non-marine) Endemism (Non-marine) based on ~ layers ""
#> 5 el957   "Averaged Topographic Relief  This data represents the~ layers "http://www.ga.gov.au/meta/A
#> 6 cl11033 "CAPAD 2020 Terrestrial The Collaborative Australian P~ layers "http://www.environment.gov
```

Search for all Wordclim layers

```
search_fields("worldclim", type = "layers")
#> # A tibble: 38 x 4
#>   id     description                                                 type   link
#>   <chr>  <chr>                                                       <chr>  <chr>
#> 1 el10982 WorldClim 2.1: Temperature - warmest month max Max Temperature of Warmest Month layers https
```

```
#> 2 el10981 WorldClim 2.1: Temperature - seasonality Temperature Seasonality          layers https://
#> 3 el10980 WorldClim 2.1: Temperature - isothermality Isothermality                layers https://ww
#> 4 el10990 WorldClim 2.1: Precipitation - wettest month Precipitation of Wettest Month   layers https
#> # ... with 34 more rows
```

See a listing of all valid fields and layers

```
show_all_fields()
#> # A tibble: 790 x 4
#>   id                  description        type   link
#>   <chr>               <chr>              <chr>  <chr>
#> 1 acceptedNameUsage   Accepted name      fields <NA>
#> 2 acceptedNameUsageID Accepted name      fields <NA>
#> 3 accessRights        Access rights      fields <NA>
#> 4 assertionUserId     Assertions by user fields <NA>
#> # ... with 786 more rows
```

Use this function with search_fields() to find fields and layers you wish to use to filter your data queries

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

This function is used to pass valid arguments to galah_select() and galah_filter(). To view valid values for a layer with categorical values, use search_field_values().

---

search_field_values *Search for valid options of a categorical field*

Description

When building a set of filters with galah_filter(), a user can use this function to check that the values provided are valid options.

Usage

```
search_field_values(field, limit = 20)
```

Arguments

field	string: field to return the categories for. Use search_fields() to view valid fields.
limit	numeric: maximum number of categories to return. 20 by default.

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) containing columns `field` (user-supplied) and `category` (i.e. field values).

Examples

Find valid values you can use to filter or categorise results for the field "basisOfRecord"

```
search_field_values("basisOfRecord")
#> # A tibble: 8 x 2
#>   field         category
#>   <chr>         <chr>
#> 1 basisOfRecord HUMAN_OBSERVATION
#> 2 basisOfRecord PRESERVED_SPECIMEN
#> 3 basisOfRecord OBSERVATION
#> 4 basisOfRecord MACHINE_OBSERVATION
#> 5 basisOfRecord MATERIAL_SAMPLE
#> 6 basisOfRecord UNKNOWN
#> 7 basisOfRecord LIVING_SPECIMEN
#> 8 basisOfRecord FOSSIL_SPECIMEN
```

Find valid values to filter or categorise results for the field "stateProvince"

```
search_field_values("stateProvince")
#> # A tibble: 20 x 2
#>    field         category
#>    <chr>         <chr>
#>  1 stateProvince New South Wales
#>  2 stateProvince Victoria
#>  3 stateProvince Queensland
#>  4 stateProvince South Australia
#>  5 stateProvince Western Australia
#>  6 stateProvince Northern Territory
#>  7 stateProvince Tasmania
#>  8 stateProvince Australian Capital Territory
#>  9 stateProvince Canterbury Land District
#> 10 stateProvince Wellington Land District
#> 11 stateProvince Southland Land District
#> 12 stateProvince Nelson Land District
#> 13 stateProvince North Auckland Land District
#> 14 stateProvince Otago Land District
#> 15 stateProvince North Island
#> 16 stateProvince Morobe
#> 17 stateProvince Westland Land District
#> 18 stateProvince Marlborough Land District
#> 19 stateProvince South Auckland
#> 20 stateProvince South Island
```

Use these values to with galah_filter() tp filter results of atlas_ functions. For example, we can return the number of records only from Tasmania

```
galah_call() |>
  galah_filter(stateProvince == "Tasmania") |>
  atlas_counts()
#> # A tibble: 1 x 1
#>     count
#>     <int>
#> 1 4504419
```

See Also

See search_fields() for ways to use information returned by this function.

---

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 data.frame of taxonomic information can also be passed directly to atlas_ functions to filter records to the specified taxon or taxa.

Usage

```
search_identifiers(identifier)
```

Arguments

identifier	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.

Examples

Look up a unique taxon identifier

```
search_identifiers(identifier = "https://id.biodiversity.org.au/node/apni/2914510")
#> # A tibble: 1 x 13
#>   scientific_name scientific_name~ taxon_concept_id rank  match_type kingdom phylum class order famil
#>   <chr>          <chr>             <chr>           <chr> <chr>      <chr>   <chr>  <chr> <chr> <chr>
#> 1 Eucalyptus bla~ Maiden            https://id.biod~ spec~ taxonIdMa~ Plantae Charo~ Equi~ Myrt~ Myrta~ E
```

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.

`search_profile_attributes` *Search for which quality filters are applied by a data quality profile*

Description

Each data quality profile is made up of a series of filters. While some users may wish to simply trust the default filters, it is often useful to check what information they return, particularly if advanced customization is needed. This function gives all of the arguments built into a specific profile.

Usage

```
search_profile_attributes(profile)
```

Arguments

`profile`	`string`: a data quality profile name, short name or id. See show_all_profiles() for valid filters

Value

A `data.frame` of profile attributes, consisting of a free text `description` and the actual `filter` used.

Examples

To find all the data quality arguments used in the profile "CSDM"

```
search_profile_attributes("CSDM")
#> # A tibble: 4 x 2
#>   description                                                    filter
#>   <chr>                                                           <chr>
#> 1 "Include only records where Spatial validity is \"true\""      "spatiallyValid:\"true\""
#> 2 "Exclude potential duplicate records"                         "-isDuplicateOf:*"
#> 3 "Exclude all records that are an outlier against any environmental layer" "-outlierLayerCount:[1 TO
#> 4 "Include only records where Year is 1970 to 2099"             "year:[1970 TO *]"
```

Then get a free-text description of each filter used in the "CSDM" profile

```
profile_info <- search_profile_attributes("CSDM")
profile_info$description
#> [1] "Include only records where Spatial validity is \"true\""
#> [2] "Exclude potential duplicate records"
#> [3] "Exclude all records that are an outlier against any environmental layer"
#> [4] "Include only records where Year is 1970 to 2099"
```

See Also

show_all_profiles() for a list of valid profiles; galah_filter() for how to include this information in a data query.

---

search_taxa *Taxon information*

Description

In the ALA, all records are associated with an identifier that uniquely identifies the taxon to which that record belongs. However, taxonomic names are often ambiguous due to homonymy; i.e. reuse of names (common or scientific) in different clades. Hence, search_taxa provides a means to search for taxonomic names and check the results are 'correct' before proceeding to download data via atlas_occurrences(), atlas_species() or atlas_counts(). The resulting data.frame of taxonomic information can be passed to galah_identify() to provide the identify argument of atlas_ functions, which then filters the resulting records to the specified taxon or taxa.

Usage

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

Arguments

...	: One or more scientific names, separated by commas and given as strings. If greater control is required to disambiguate search terms, taxonomic levels can be provided explicitly via a data.frame (see examples). Note that searches are not case-sensitive.

Value

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

Examples

Search using a single term

```
search_taxa("Reptilia")
#> # A tibble: 1 x 9
#>   search_term scientific_name taxon_concept_id                    rank  match_type kingdom  phylum
#>   <chr>      <chr>           <chr>                               <chr> <chr>      <chr>    <chr>
#> 1 Reptilia   REPTILIA        urn:lsid:biodiversity.org.au:afd.taxon:682e1~ class exactMatch Animalia
```

Note that search_taxa() is not case sensitive

```
search_taxa("reptilia") # not case sensitive
#> # A tibble: 1 x 9
#>   search_term scientific_name taxon_concept_id                    rank  match_type kingdom  phylum
#>   <chr>      <chr>           <chr>                                <chr> <chr>      <chr>    <chr>
#> 1 reptilia   REPTILIA        urn:lsid:biodiversity.org.au:afd.taxon:682e1~ class exactMatch Animalia
```

Search multiple taxa. `search_taxa()` will return one row per taxon

```
search_taxa(c("reptilia", "mammalia"))
#> # A tibble: 2 x 10
#>   search_term scientific_name taxon_concept_id          rank  match_type kingdom  phylum class issue
#>   <chr>      <chr>           <chr>                     <chr> <chr>      <chr>    <chr>  <chr> <chr>
#> 1 reptilia   REPTILIA        urn:lsid:biodiversity.org.au:a~ class exactMatch Animalia Chord~ Rept~ no
#> 2 mammalia   MAMMALIA        urn:lsid:biodiversity.org.au:a~ class exactMatch Animalia Chord~ Mamm~ no
```

`galah_identify()` uses `search_taxa()` to narrow data queries

```
galah_call() |>
  galah_identify("reptilia") |>
  atlas_counts()
#> # A tibble: 1 x 1
#>     count
#>     <int>
#> 1 1317131
```

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.

---

show_all_atlases *List supported Living Atlases*

Description

galah supports downloading data from a number of International Living Atlases. Use this function to get a list of all currently supported atlases.

Usage

```
show_all_atlases()
```

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) of Living Atlas information, including taxonomy source and information for each atlas.

Examples

See all supported atlases

```
show_all_atlases()
#> # A tibble: 6 x 3
#>   atlas     taxonomy_source taxonomy_info
#>   <chr>     <chr>           <chr>
#> 1 Australia ALA             https://bie.ala.org.au/
#> 2 Austria   GBIF        https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c
#> 3 Guatemala GBIF         https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c
#> 4 Spain     GBIF        https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c
#> 5 Sweden    GBIF        https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c
#> 6 UK        NBN         https://www.nhm.ac.uk/our-science/data/uk-species.html
```

See Also

This function is helpful in setting up `galah_config()`.

---

show_all_cached_files *List previously cached files*

Description

When using caching by setting `galah_config(caching = TRUE)`, show a list of all previously cached files. This function acheives this by using query metadata stored in metadata.rds in the cache directory

Usage

```
show_all_cached_files()
```

Value

A `list` of available cached files, the function used to create them, and the filter object

Examples

Configure caching and create a query to cache with `galah_config()`

```
galah_config(caching = TRUE)
dat <- atlas_counts(group_by = galah_group_by(year))
```

Show a listing of previously cached files

```
show_all_cached_files()
```

show_all_profiles *List data quality profiles*

Description

The ALA provides a number of pre-built data quality profiles for filtering data according to quality checks. A data quality profile can be specified in the `profile` argument in `galah_filter()` and used to filter searches in `atlas_occurrences()`, `atlas_counts()` and `atlas_species()`.

Usage

```
show_all_profiles()
```

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) of available profiles

Examples

Show a list of all available data quality profiles

```
show_all_profiles()
#> # A tibble: 7 x 4
#>     id name                                          shortName      description
#>   <int> <chr>                                           <chr>          <chr>
#> 1    2 ALA General (pre-pipelines)                   ALA-legacy     "The default ALA profile filters
#> 2   35 Species Distribution Modelling (CSDM) (pre-pipelines) CSDM-legacy    "Base filters for the Co
#> 3   44 Data licensed for all uses (pre-pipelines)       re-usable-legacy "Data licensed for re-use,
#> 4   92 ALA General                                  ALA          "The default ALA profile filters out reco
#> 5  124 Species Distribution Modelling (CSDM)           CSDM         "Base filters for the Collabora
#> 6  133 Data licensed for all uses                   re-usable      "Data licensed for re-use, includi
#> 7  224 AVH                                      AVH          "AVH data quality profile"
```

Values in the `shortName` column can be used with `galah_filter()` to narrow your data query results

```
galah_filter(profile == "ALA")
#> Warning: Invalid field(s) detected.
#> i See a listing of all valid fields with `show_all_fields()`.
#> i Search for the valid name of a desired field with `search_fields()`.
#> x Invalid field(s): profile.
#> # A tibble: 1 x 4
#>   variable logical value query
#>   <chr>    <chr>   <chr> <chr>
#> 1 profile  ==      ALA   "(profile:\"ALA\")"
```

See Also

This function gives viable profile names for passing to `galah_filter()`. For more detail on a given profile see `search_profile_attributes()`.

`show_all_ranks` *Find valid taxonomic ranks*

Description

Return taxonomic ranks recognised by the ALA.

Usage

```
show_all_ranks()
```

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) of available ranks

Examples

Show a listing of all taxonomic ranks

```
show_all_ranks()
#> # A tibble: 69 x 2
#>      id name
#>   <int> <chr>
#> 1     1 root
#> 2     2 superkingdom
#> 3     3 kingdom
#> 4     4 subkingdom
#> # ... with 65 more rows
```

Use ranks with `galah_down_to()` and `atlas_taxonomy()` to get taxonomic trees

```
galah_call() %>%
  galah_identify("fungi") %>%
  galah_down_to(subphylum) %>%
  atlas_taxonomy()
#>                         levelName
#> 1  Fungi
#> 2   ¦--Ascomycota
#> 3   ¦   ¦--Pezizomycotina
#> 4   ¦   ¦--Saccharomycotina
#> 5   ¦   °--Taphrinomycotina
#> 6   ¦--Basidiomycota
#> 7   ¦   ¦--Agaricomycotina
```

```
#> 8   ¦   ¦--Pucciniomycotina
#> 9   ¦   ¦--Ustilaginomycotina
#> 10  ¦   °--Wallemiomycotina
#> 11  ¦--Mucoromycota
#> 12  ¦   °--Glomeromycotina
#> 13  °--Zygomycota
#> 14      ¦--Entomophthoromycotina
#> 15      ¦--Kickxellomycotina
#> 16      ¦--Mortierellomycotina
#> 17      ¦--Mucoromycotina
#> 18      °--Zoopagomycotina
```

See Also

This function provides a reference that is useful when specifying the `down_to` argument of `atlas_taxonomy()`.

`show_all_reasons` *List valid download reasons*

Description

When downloading occurrence data with `atlas_occurrences()` the ALA APIs require a reason for download to be specified. By default, a download reason of 'scientific research' is set for you, but if you wish to change this you can do so with `galah_config()`. Use this function to view the list of download reason code and names. When specifying a reason, you can use either the download code or name.

Usage

```
show_all_reasons()
```

Value

An object of class `tbl_df` and `data.frame` (aka a tibble) of valid download reasons, containing the id and name for each reason.

Examples

Show a listing of all accepted reasons for downloading occurrence data

```
show_all_reasons()
#> # A tibble: 13 x 2
#>       id name
#>    <int> <chr>
#>  1     0 conservation management/planning
#>  2     1 biosecurity management/planning
#>  3     2 environmental assessment
#>  4     3 education
```

```
#>  5     4 scientific research
#>  6     5 collection management
#>  7     6 other
#>  8     7 ecological research
#>  9     8 systematic research/taxonomy
#> 10    10 testing
#> 11    11 citizen science
#> 12    12 restoration/remediation
#> 13    13 species modelling
```

Add your download reason when configuring your session with `galah_config()`

```
galah_config(download_reason_id = 3)
```

See Also

This function is helpful in setting up `galah_config()`.

Index