Package 'galah'

April 12, 2024

Type Package

Title Biodiversity Data from the GBIF Node Network

Version 2.0.2

Description The Global Biodiversity Information Facility
('GBIF', <https://www.gbif.org>) sources data from an international network
of data providers, known as 'nodes'. Several of these nodes - the ``living
atlases'' (<https://living-atlases.gbif.org>) - maintain their own web
services using software originally developed by the Atlas of Living
Australia ('ALA', <https://www.ala.org.au>). 'galah' enables the R community
to directly access data and resources hosted by 'GBIF' and its partner nodes.

Depends R (>= 4.1.0)

Imports cli, crayon, dplyr, glue (>= 1.3.2), httr2, jsonlite (>=
0.9.8), lifecycle (>= 1.0.0), potions (>= 0.2.0), purrr, readr,
rlang, sf, stringr, tibble, tidyr, tidyselect, utils

Suggests covr, gt, kableExtra, knitr, magrittr, pkgdown, reactable,
rmarkdown, testthat

License MPL-2.0

URL <https://galah.ala.org.au/R/>

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

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

LazyLoad yes

VignetteBuilder knitr

RoxygenNote 7.2.3

Encoding UTF-8

NeedsCompilation no

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

Repository CRAN

Date/Publication 2024-04-12 05:50:02 UTC

R topics documented:

---

arrange.data_request   *Arrange rows of a query*

Description

[Experimental]

arrange.data_request() arranges rows of a query on the server side, meaning that prior to sending a query, the query is constructed in such a way that information will be arranged when the query is processed. Any data that is then returned by the query will have rows already pre-arranged.

The benefit of using arrange() within a galah_call() is that it is faster to process arranging rows on the server side than arranging rows locally on downloaded data, especially if the dataset is large or complex.

arrange() can be used within a galah_call() pipe, but only for queries of type = "occurrences-count". The galah_call() pipe must include count() and finish with collect() (see examples).

Usage

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

## S3 method for class 'metadata_request'
arrange(.data, ...)
```

Arguments

`.data`	An object of class `data_request`
`...`	Either `count` or `index`

Examples

```
## Not run:

# Arrange grouped counts by ascending year
galah_call() |>
  identify("Crinia") |>
  filter(year >= 2020) |>
  group_by(year) |>
  arrange(year) |>
  count() |>
  collect()

# Arrange grouped counts by ascending record count
galah_call() |>
  identify("Crinia") |>
  filter(year >= 2020) |>
  group_by(year) |>
  arrange(count) |>
  count() |>
  collect()

# Arrange grouped counts by descending year
galah_call() |>
  identify("Crinia") |>
  filter(year >= 2020) |>
  group_by(year) |>
  arrange(desc(year)) |>
  count() |>
  collect()

## End(Not run)
```

---

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("occurrences", "species")
)

## S3 method for class 'data_request'
count(x, ..., wt, sort, name)
```

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("occurrences-count", "species-count")`. Defaults to `"occurrences-count"`, which returns the number of records that match the selected criteria; alternatively returns the number of species. Formerly accepted arguments (`"records"` or `"species"`) are deprecated but remain functional.
x	An object of class `data_request`, created using [galah_call()]
...	currently ignored
wt	currently ignored
sort	currently ignored
name	currently ignored

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

```
## Not run:
# classic syntax:
galah_call() |>
  galah_filter(year == 2015) |>
  atlas_counts()

# synonymous with:
request_data() |>
  filter(year == 2015) |>
  count() |>
```

```
  collect()

## End(Not run)
```

---

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,
  select = NULL,
  geolocate = NULL,
  data_profile = NULL
)
```

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()
select	list: 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()

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()


# post version 2.0, it is possible to run all steps in sequence
# first, get occurrences, making sure to include media fields:
occurrences_df <- request_data() |>
  identify("Regent Honeyeater") |>
  filter(!is.na(images), year == 2011) |>
  select(group = "media") |>
  collect()

# second, get media metadata
media_info <- request_metadata() |>
  filter(media == occurrences_df) |>
  collect()

# the two steps above + `right_join()` are synonmous with `atlas_media()`
# third, get images
request_files() |>
  filter(media == media_df) |>
  collect(thumbnail = TRUE)
```

```
# step three is synonymous with `collect_media()`

## End(Not run)
```

---

atlas_occurrences   *Collect a set of occurrences*

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 `tibble` 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,
  file = NULL
)
```

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	`string`: (optional) DOI to download. If provided overrides all other arguments. Only available for the ALA.
file	`string`: (Optional) file name. If not given, will be set to `data` with date and time added. The file path (directory) is always given by `galah_config()$package$directory`.

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()](#), or to use compute() to run the call before collecting it later with collect().

Value

An object of class tbl_df and data.frame (aka a tibble) of occurrences, containing columns as specified by [galah_select()](#).

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()

# Or identically with alternative syntax
request_data() |>
  identify("Litoria") |>
  filter(year >= 2010 & year <= 2020) |>
  collect()

# 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	*Collect the set of species observed within the specified filters*

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
)
```

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()](#)

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

```
## Not run:
# 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()

# Or alternatively
request_data(type = "species") |>
  identify("peramelidae") |>
  collect()

# It's good idea to find how many species there are before downloading
galah_call() |>
  galah_identify("Heleioporus") |>
  atlas_counts(type = "species")
# Or alternatively
request_data(type = "species") |>
  identify("Heleioporus") |>
  count() |>
  collect()

## End(Not run)
```

---

collapse_galah   *Generate a query*

Description

collapse() constructs a valid query so it can be inspected before being sent. It typically occurs
at the end of a pipe, traditionally begun with galah_call(), that is used to define a query. As of
version 2.0, objects of class data_request (created using request_data()), metadata_request
(from request_metadata()) or files_request (from request_files()) are all supported by
collapse(). Any of these objects can be created using galah_call() via the method argument.

Usage

```
## S3 method for class 'data_request'
collapse(x, ..., mint_doi, .expand = FALSE)
```

```
## S3 method for class 'metadata_request'
collapse(x, .expand = FALSE, ...)

## S3 method for class 'files_request'
collapse(x, thumbnail = FALSE, ...)
```

Arguments

x	An object of class `data_request`, `metadata_request` or `files_request`
...	Arguments passed on to other methods
mint_doi	Logical: should a DOI be minted for this download? Only applies to `type = "occurrences"` when atlas chosen is "ALA".
.expand	Logical: should the `query_set` be returned? This object shows all the requisite data needed to process the supplied query. Defaults to `FALSE`; if `TRUE` will append the `query_set` to an extra slot in the `query` object.
thumbnail	Logical: should thumbnail-size images be returned? Defaults to `FALSE`, indicating full-size images are required.

Value

An object of class `query`, which is a list-like object containing at least the slots `type` and `url`.

---

collect_galah	*Retrieve a database query*

Description

`collect()` attempts to retrieve the result of a query from the selected API.

Usage

```
## S3 method for class 'data_request'
collect(x, ..., wait = TRUE, file = NULL)

## S3 method for class 'metadata_request'
collect(x, ...)

## S3 method for class 'files_request'
collect(x, ...)

## S3 method for class 'query'
collect(x, ..., wait = TRUE, file = NULL)

## S3 method for class 'computed_query'
collect(x, ..., wait = TRUE, file = NULL)
```

Arguments

x	An object of class `data_request`, `metadata_request` or `files_request` (from `galah_call()`); or an object of class `query_set` or `query` (from `collapse()` or `compute()`)
...	Arguments passed on to other methods
wait	logical; should galah wait for a response? Defaults to FALSE. Only applies for `type = "occurrences"` or `"species"`.
file	(Optional) file name. If not given, will be set to `data` with date and time added. The file path (directory) is always given by `galah_config()$package$directory`.

Value

In most cases, `collect()` returns a `tibble` containing requested data. Where the requested data are not yet ready (i.e. for occurrences when `wait` is set to `FALSE`), this function returns an object of class `query` that can be used to recheck the download at a later time.

---

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, thumbnail = FALSE, path)
```

Arguments

df	tibble: returned by `atlas_media()` or a pipe starting with `request_data(type = "media")`.
thumbnail	logical: If TRUE will download small thumbnail-sized images, rather than full size images (default).
path	string: **[Deprecated]** Use `galah_config(directory = "path-to-directory)"` instead. Supply a path to a local folder/directory where downloaded media will be saved to.

Value

Invisibly returns a `tibble` listing the number of files downloaded, grouped by their HTML status codes. Primarily called for the side effect of downloading available image & media files to a user local directory.

Examples

```
## Not run:
# Use `atlas_media()` to return a `tibble` of records that contain media
x <- 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_config(directory = "media_files")
collect_media(x)

## End(Not run)
```

---

compute_galah   *Compute a query*

Description

compute() is useful for several purposes. It's original purpose is to send a request for data, which can then be processed by the server and retrieved at a later time (via collect()).

Usage

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

## S3 method for class 'metadata_request'
compute(x, ...)

## S3 method for class 'files_request'
compute(x, ...)

## S3 method for class 'query'
compute(x, ...)
```

Arguments

x	An object of class data_request, metadata_request or files_request (i.e. constructed using a pipe) or query (i.e. constructed by collapse())
...	Arguments passed on to other methods

Value

An object of class computed_query, which is identical to class query except for occurrence data, where it also contains information on the status of the request.

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(...)

apply_profile(.data, ...)
```

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)`
.data	An object of class `data_request`

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

```
## Not run:
# Apply a data quality profile to a query
galah_call() |>
  galah_identify("reptilia") |>
  galah_filter(year == 2021) |>
  galah_apply_profile(ALA) |>
  atlas_counts()

## End(Not run)
```

galah_call   *Start building a query*

Description

To download data from the selected atlas, one must construct a query. This query tells the atlas API
what data to download and return, as well as how it should be filtered. Using galah_call() allows
you to build a piped query to download data, in the same way that you would wrangle data with
dplyr and the tidyverse.

Since version 2.0, galah_call() is a wrapper to a group of underlying request_ functions. Each
of these functions can begin a piped query and end with collapse(), compute() or collect().

The underlying request_ #' functions are useful because they allow galah to separate different
types of requests to perform better. For example, filter.data_request translates filters in R to
solr, whereas filter.metadata_request searches using a search term.

For more details see the object-oriented programming vignette: vignette("object_oriented_programming",
package = "galah")

Usage

```
galah_call(method = c("data", "metadata", "files"), type, ...)

request_data(
  type = c("occurrences", "occurrences-count", "occurrences-doi", "species",
    "species-count"),
  ...
)

request_metadata(type)

request_files(type = "media")
```

Arguments

method	string: what request function should be called. Should be one of "data" (default), "metadata" or "files"
type	string: what form of data should be returned? Acceptable values are specified by the corresponding request function
...	Zero or more arguments to alter a query. See 'details'.

Details

Each atlas has several types of data that can be chosen. Currently supported are "occurrences" (the
default), "species" and "media" (the latter currently only for ALA). It is also possible to use type
= "occurrences-count" and type = "species-count"; but in practice this is synonymous with
galah_call() |> count(), and is therefore only practically useful for debugging (via collapse()
and compute()).

Other named arguments are supported via `....` In practice, functions with a `galah_` prefix and S3 methods ported from `dplyr` assign information to the correct slots internally. Overwriting these with user-defined alternatives is possible, but not advised. Accepted arguments are:

- filter (accepts galah_filter() or [filter()](#))
- select (accepts galah_select() or [select](#))
- group_by (accepts galah_group_by() or [group_by()](#))
- identify (accepts galah_identify() or [identify()](#))
- geolocate (accepts galah_geolocate(), galah_polygon() galah_bbox() or [st_crop()](#))
- limit (accepts [slice_head()](#))
- doi (accepts a sting listing a valid DOI, specific to collect() when type = "doi")

Unrecognised names are ignored by `collect()` and related functions.

Value

Each sub-function returns a different object class: `request_data()` returns `data_request`. `request_metadata` returns `metadata_request`, `request_files()` returns `files_request`.

See Also

[collapse.data_request()](#), [compute.data_request()](#), [collect.data_request()](#)

Examples

```
## Not run:
# 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(year) |>
  atlas_counts()

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

# 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()


# ----------
```

```
# Since galah 2.0.0, a pipe can start with a `request_` function.
# This allows users to use `collapse()`, `compute()` and `collect()`.

# Get number of records of *Aves* from 2001 to 2004 by year
request_data(type = "occurrences-count") |>
  galah_identify("Aves") |>
  galah_filter(year > 2000 & year < 2005) |>
  galah_group_by(year) |>
  collect()

# Get information for all species in *Cacatuidae* family
request_data(type = "species") |>
  galah_identify("Cacatuidae") |>
  collect()

# Get metadata information about supported atlases in galah
request_metadata(type = "atlases") |>
  collect()


## 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(...)
```

Arguments

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

- api-key string: A registered API key (currently unused).
- 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)
- 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 registered 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 registered username (GBIF only)

- verbose logical: should galah give verbose such as progress bars? Defaults to FALSE.

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 atlas in question.
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   *Deprecated functions*

Description

These include:

- galah_down_to() in favour of galah_filter()

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

```
## Not run:
# Return a taxonomic tree of *Chordata* down to the class level
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)

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

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

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

Arguments

`...`	filters, in the form `field logical value`
`profile`	**[Deprecated]** Use `galah_apply_profile` instead.
`.data`	An object of class `files_request`, created using [`request_files()`](#)

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")`

It is possible to filter by 'assertions', which are statements about data validity, e.g. to remove those lacking critical spatial or taxonomic data: `galah_filter(assertions != c("INVALID_SCIENTIFIC_NAME", "COORDINATE` Valid assertions can be found using `show_all(assertions)`.

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

```
## Not run:
# Filter query results to return records of interest
galah_call() |>
  galah_filter(year >= 2019,
                basisOfRecord == "HumanObservation") |>
  atlas_counts()

# Alternatively, the same call using `dplyr` functions:
request_data() |>
  filter(year >= 2019,
                basisOfRecord == "HumanObservation") |>
  count() |>
  collect()

## End(Not run)
```

---

galah_geolocate   *Narrow a query to within a specified area*

Description

Restrict results to those from a specified area using galah_geolocate(). Areas can be specified as
either polygons or bounding boxes, depending on type. Alternatively, users can call the underlying
functions directly via galah_polygon(), galah_bbox() or galah_radius(). It is possible to use
sf syntax by calling st_crop(), which is synonymous with galah_polygon().

Use a polygon If calling galah_geolocate(), the default type is "polygon", which narrows
queries to within an area supplied as a POLYGON or MULTIPOLYGON. 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.

Usage

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

galah_bbox(...)

galah_polygon(...)
```

```
galah_radius(...)

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

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.
`x`	An object of class `data_request`, created using `galah_call()`
`y`	A valid Well-Known Text string (wkt), a `POLYGON` or a `MULTIPOLYGON`

Details

Use a bounding box 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`.

[Experimental] Use a point radius Alternatively, set `type = "radius"` to narrow queries to within a circular area around a specific point location. Point coordinates can be supplied as latitude/longitude coordinate numbers or as an `sf` object (`sfc_POINT`). Area is supplied as a `radius` in kilometres. Default radius is 10 km.

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. If `type = "radius`, `sfc_POINT` objects will be converted to lon/lat coordinate numbers to query the ALA. Default radius is 10 km.

Value

If `type = "polygon"` or `type = "bbox"`, length-1 string (class `character`) containing a multipolygon WKT string representing the area provided. If `type = "radius"`, `list` of `lat`, `long` and `radius` values.

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()

# Search for records within a polygon using an `sf` object
location <- "POLYGON((142.3 -29.0,142.7 -29.1,142.7 -29.4,142.3 -29.0))" |>
 sf::st_as_sfc()
galah_call() |>
  galah_identify("reptilia") |>
  galah_polygon(location) |>
  atlas_counts()

# Alternatively, we can use `st_crop()` as a synonym for `galah_polygon()`.
# Hence the above example can be rewritten as:
request_data() |>
  identify("reptilia") |>
  st_crop(location) |>
  count() |>
  collect()

# Search for records using a Well-known Text string (WKT)
wkt <- "POLYGON((142.3 -29.0,142.7 -29.1,142.7 -29.4,142.3 -29.0))"
galah_call() |>
  galah_identify("vulpes") |>
  galah_geolocate(wkt) |>
  atlas_counts()

# Search for records within the bounding box extracted from an `sf` object
location <- "POLYGON((142.3 -29.0,142.7 -29.1,142.7 -29.4,142.3 -29.0))" |>
  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()

# Search for records within a radius around a point's coordinates
```

```
galah_call() |>
  galah_identify("manorina melanocephala") |>
  galah_geolocate(lat = -33.7,
                  lon = 151.3,
                  radius = 5,
                  type = "radius") |>
  atlas_counts()

# Search for records with a radius around an `sf_POINT` object
point <- sf::st_sfc(sf::st_point(c(-33.66741, 151.3174)), crs = 4326)
galah_call() |>
  galah_identify("manorina melanocephala") |>
  galah_geolocate(point,
                  radius = 5,
                  type = "radius") |>
  atlas_counts()

## End(Not run)
```

---

galah_group_by   *Specify fields to group when downloading record counts*

Description

`count.data_request()` and `atlas_counts()` support server-side grouping of data. Grouping can be used to return record counts grouped by multiple, valid fields (found by `search_all(fields)`).

Usage

```
galah_group_by(...)

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

Arguments

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

Value

If any arguments are provided, returns a `data.frame` with columns name and type, as per `select.data_request()`.

Examples

```
## Not run:
galah_call() |>
  galah_group_by(basisOfRecord) |>
  atlas_counts()
```

```
## End(Not run)
```

---

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 = NULL)

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

## S3 method for class 'metadata_request'
identify(x, ...)
```

Arguments

...	One or more scientific names.
search	**[Deprecated]** galah_identify() now always does a search to verify search terms; ergo this argument is ignored.
x	An object of class metadata_request, created using [request_metadata()](#)

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

```
## Not run:
# 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()

# Within a pipe, `identify()` and `galah_identify()` are synonymous.
# hence the following is identical to the previous example:
request_data() |>
  identify("Eolophus") |>
  count() |>
  collect()

# If you know a valid taxon identifier, use `galah_filter()` instead.
# (This was formerly supported by `galah_identify()` with `search = FALSE`)
id <- "https://biodiversity.org.au/afd/taxa/009169a9-a916-40ee-866c-669ae0a21c5c"
galah_call() |>
  galah_filter(lsid == id) |>
  atlas_counts()

## End(Not run)
```

---

galah_select *Specify fields for occurrence download*

Description

GBIF nodes 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 tibble, 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().

The full list of available fields can be viewed with show_all(fields). Note that select() and galah_select() are supported for all atlases that allow downloads, with the exception of GBIF, for which all columns are returned.

Usage

```
galah_select(..., group)

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

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"` `"taxonomy"`, `"media"` and `"assertions"`.
`.data`	An object of class `data_request`, created using [galah_call()](#)

Details

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 = "taxonomy"` returns higher taxonomic information for a given query. It is the only group that is accepted by `atlas_species()` as well as `atlas_occurrences()`.

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

For `atlas_occurrences()`, arguments passed to `...` should be valid field names, which you can check using `show_all(fields)`. For `atlas_species()`, it should be one or more of:

- `counts` to include counts of occurrences per species.
- `synonyms` to include any synonymous names.
- `lists` to include authoritiative lists that each species is included on.

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()

# When used in a pipe, `galah_select()` and `select()` are synonymous.
# Hence the previous example can be rewritten as:
request_data() |>
  identify("perameles") |>
  select(basisOfRecord, group = "basic") |>
  collect()

## End(Not run)
```

---

print_galah_objects   *Print galah objects*

Description

As of version 2.0, galah supports several bespoke object types. Classes data_request, metadata_request and files_request are for starting pipes to download different types of information. These objects are parsed using collapse() into a query object, which contains one or more URLs necessary to return the requested information. This object is then passed to compute() and/or collect(). Finally, galah_config() creates an object of class galah_config which (unsurprisingly) stores configuration information.

Usage

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

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

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

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

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

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

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

Arguments

x	an object of the appropriate `class`
...	Arguments to be passed to or from other methods

Value

Print does not return an object; instead it prints a description of the object to the console

Examples

```
# The most common way to start a pipe is with `galah_call()`
# later functions update the `data_request` object
galah_call() |> # same as calling `request_data()`
  filter(year >= 2020) |>
  group_by(year) |>
  count()

# Metadata requests are formatted in a similar way
request_metadata() |>
  filter(field == basisOfRecord) |>
  unnest()

# Queries are converted into a `query_set` by `collapse()`
x <- galah_call() |> # same as calling `request_data()`
  filter(year >= 2020) |>
  count() |>
  collapse()
```

```
print(x)

# Each `query_set` contains one or more `query` objects
x[[3]]
```

---

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.

[Stable] 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_assertions(query)

search_apis(query)

search_atlases(query)

search_collections(query)

search_datasets(query)

search_fields(query)

search_identifiers(...)

search_licences(query)

search_lists(query)

search_profiles(query)

search_providers(query)

search_ranks(query)

search_reasons(query)
```

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

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.
`...`	A set of strings or a tibble to be queried

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/

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

```
## Not run:
```

```
# 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")

# An alternative is to download the data and then `filter` it. This is
# largely synonymous, and allows greater control over which fields are searched.
request_metadata(type = "fields") |>
 collect() |>
 dplyr::filter(grepl("date", id))

## End(Not run)
```

---

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.

[Stable] show_all() is a helper function that can display multiple types of information from show_all_ sub-functions.

Usage

```
show_all(..., limit = NULL)

show_all_apis(limit = NULL)

show_all_assertions(limit = NULL)
```

```
show_all_atlases(limit = NULL)

show_all_collections(limit = NULL)

show_all_datasets(limit = NULL)

show_all_fields(limit = NULL)

show_all_licences(limit = NULL)

show_all_lists(limit = NULL)

show_all_profiles(limit = NULL)

show_all_providers(limit = NULL)

show_all_ranks(limit = NULL)

show_all_reasons(limit = NULL)
```

Arguments

`...`	String showing what type of information is to be requested. See `Details` (below) for accepted values.
`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
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
Filters	assertions	Show results of data quality checks run by each atlas	show_all_as
	fields	Show fields that are stored in an atlas	show_all_fi
	licenses	Show what copyright licenses are applied to media	show_all_li
	profiles	Show what data profiles are available	show_all_pr
Taxonomy	lists	Show what species lists are available	show_all_li
	ranks	Show valid taxonomic ranks (e.g. Kingdom, Class, Order, etc.)	show_all_ra

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/

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

```
## Not run:
# 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_all()` is synonymous with `request_metadata() |> collect()`
request_metadata(type = "fields") |>
  collect()

## End(Not run)
```

---

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

```
## Not run:
# Show values in field 'cl22'
search_fields("cl22") |>
  show_values()

# This is synonymous with `request_metadata() |> unnest()`.
# For example, the previous example can be run using:
request_metadata() |>
  filter(field == "cl22") |>
  unnest() |>
  collect()

# 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()

  ## End(Not run)
```

---

slice_head.data_request

Subset first rows of data_request

Description

[Experimental]

This is a simple function to set the limit argument in [atlas_counts()](#) using dplyr syntax. As of galah 2.0.0, slice_head() is only supported in queries of type occurrences-count(), or metadata requests. Note also that slice_head() is lazily evaluated; it only affects a query once it is run by compute() or (more likely) collect().

Usage

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

## S3 method for class 'metadata_request'
slice_head(.data, ..., n, prop, by = NULL)
```

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, but could be added later
by	currently ignored

Examples

```
## Not run:
# Limit number of rows returned to 3.
# In this case, our query returns the top 3 years with most records.
galah_call() |>
  identify("perameles") |>
  filter(year > 2010) |>
  group_by(year) |>
  count() |>
  slice_head(n = 3) |>
  collect()

## End(Not run)
```

---

tidyverse_functions   *Non-generic tidyverse functions*

Description

Several useful functions from tidyverse packages are `generic`, meaning that we can define class-specific versions of those functions and implement them in galah; examples include `filter()`, `select()` and `group_by()`. However, there are also functions that are only defined within tidyverse packages and are not generic. In a few cases we have re-implemented these functions in galah. This has the consequence of supporting consistent syntax with tidyverse, at the cost of potentially introducing conflicts. This can be avoided by using the `::` operator where required (see examples).

Usage

```
desc(...)

unnest(.query)
```

Arguments

`...`	column to order by
`.query`	An object of class `metadata_request`

Details

The following functions are included:

- `desc()` (dplyr): Use within `arrange()` to specify arrangement should be descending
- `unnest()` (tidyr): Use to 'drill down' into nested information on `fields`, `lists`, `profiles`, or `taxa`

These galah versions all use lazy evaluation.

Value

- `galah::desc()` returns a `tibble` used by `arrange.data_request()` to arrange rows of a query.
- `galah::unnest()` returns an object of class `metadata_request`.

See Also

[arrange.data_request()](), [galah_call()]()

Examples

```
## Not run:
# Arrange grouped record counts by descending year
galah_call() |>
  identify("perameles") |>
  filter(year > 2019) |>
  count() |>
  arrange(galah::desc(year)) |>
  collect()

# Return values of field `basisOfRecord`
request_metadata() |>
  galah::unnest() |>
  filter(field == basisOfRecord) |>
  collect()

# Using `galah::unnest()` in this way is equivalent to:
show_all(fields, "basisOfRecord") |>
  show_values()

## End(Not run)
```

Index