Package 'galaxias'

July 7, 2025

Type Package

Title Describe, Package, and Share Biodiversity Data

Version 0.1.0

Description The Darwin Core data standard is widely used to share biodiversity information, most notably by the Global Biodiversity Information Facility and its partner nodes; but converting data to this standard can be tricky. 'galaxias' is functionally similar to 'devtools', but with a focus on building Darwin Core Archives rather than R packages, enabling data to be shared and re-used with relative ease. For details see Wieczorek and colleagues (2012) <doi:10.1371/journal.pone.0029715>.

Depends R (>= 4.3.0), corella

Imports cli, delma, dplyr, fs, glue, httr2, jsonlite, purrr, readr, rlang, tibble, usethis, withr, zip

Suggests gt, here, janitor, knitr, lubridate, rmarkdown, R.utils, testthat (>= 3.0.0), tidyr, xml2

License MPL-2.0

URL https://galaxias.ala.org.au/R/

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

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

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

NeedsCompilation no

Author Martin Westgate [aut, cre], Shandiya Balasubramaniam [aut], Dax Kellie [aut]

Repository CRAN

Date/Publication 2025-07-07 12:30:02 UTC

Contents

`build_archive` *Build a Darwin Core Archive from a folder*

Description

A Darwin Core archive is a zip file containing a combination of data and metadata. `build_archive()` constructs this zip file in the parent directory. The function assumes that all necessary files have been pre-constructed, and can be found inside the `"data-publish"` directory with no additional or redundant information. Structurally, `build_archive()` is similar to `devtools::build()`, in the sense that it takes a repository and wraps it for publication.

Usage

```
build_archive(file = "dwc-archive.zip", overwrite = FALSE, quiet = FALSE)
```

Arguments

`file`	The name of the file to be built in the parent directory. Should end in `.zip`.
`overwrite`	(logical) Should existing files be overwritten? Defaults to `FALSE`.
`quiet`	(logical) Whether to suppress messages about what is happening. Default is set to `FALSE`; i.e. messages are shown.

Details

This function looks for three types of objects in the `data-publish` directory:

- Data

  One or more csv files named `occurrences.csv`, `events.csv` and/or `multimedia.csv`. These csv files contain data standardised using Darwin Core Standard (see `corella::corella-package()` for details). A `data.frame`/`tibble` can be added to the correct folder using `use_data()`.

- Metadata

  A metadata statement in `EML` format with the file name `eml.xml`. Completed metadata statements written markdown as `.Rmd` or `qmd` files can be converted and saved to the correct folder using `use_metadata()`. Create a new template with `use_metadata_template()`.

- Schema
  A 'schema' document in xml format with the file name `meta.xml`. `build_archive()` will detect whether this file is present and build a schema file if missing. This file can also be constructed separately using `use_schema()`.

Value

Doesn't return anything; called for the side-effect of building a 'Darwin Core Archive' (i.e. a zip file).

See Also

`use_data()`, `use_metadata()`, `use_schema()`

`check_archive` *Check whether an archive meets the Darwin Core Standard via API*

Description

Check whether a specified Darwin Core Archive is ready for sharing and publication, according to the Darwin Core Standard. `check_archive()` tests an archive - defaulting to `"dwc-archive.zip"` in the users' parent directory - using an online validation service. Currently only supports validation using GBIF.

Usage

```
check_archive(
  file = "dwc-archive.zip",
  username = NULL,
  email = NULL,
  password = NULL,
  wait = TRUE,
  quiet = FALSE
)

get_report(
  obj,
  username = NULL,
  password = NULL,
  n = 5,
  wait = TRUE,
  quiet = FALSE
)

view_report(x, n = 5)

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

Arguments

file	The name of the file in the parent directory to pass to the validator API, ideally created using build_archive().
username	Your GBIF username.
email	The email address used to register with gbif.org.
password	Your GBIF password.
wait	(logical) Whether to wait for a completed report from the API before exiting (TRUE, the default), or try the API once and return the result regardless (FALSE).
quiet	(logical) Whether to suppress messages about what is happening. Default is set to FALSE; i.e. messages are shown.
obj	Either an object of class character containing a key that uniquely identifies your query; or an object of class gbif_validator. returned by check_archive() or get_report()
n	Maximum number of entries to print per file. Defaults to 5.
x	An object of class gbif_validator.
...	Additional arguments, currently ignored.

Details

Internally, check_archive() both POSTs the specified archive to the GBIF validator API and then calls get_report() to retrieve (GET) the result. get_report() is exported to allow the user to download results at a later time should they wish; this is more efficient than repeatedly generating queries with check_archive() if the underlying data are unchanged. A third option is simply to assign the outcome of check_archive() or get_report() to an object, then call view_report() to format the result nicely. This approach doesn't require any further API calls and is considerably faster.

Note that information returned by these functions is provided verbatim from the institution API, not from galaxias.

Value

Both check_archive() and get_report() return an object of class gbif_validator to the workspace. view_report() and print.gbif_validator() don't return anything, and are called for the side-effect of printing useful information to the console.

See Also

check_directory() which runs checks on a directory (but **not** an archive) locally, rather than via API.

check_directory *Check whether contents of directory comply with the Darwin Core Standard*

Description

Checks that files in the `data-publish` directory meet Darwin Core Standard. `check_directory()` runs `corella::check_dataset()` on `occurrences.csv` and `events.csv` files, and `delma::check_metadata()` on the `eml.xml` file, if they are present. These `check_` functions run tests to determine whether data and metadata pass Darwin Core Standard criteria.

Usage

```
check_directory()
```

Value

Doesn't return anything; called for the side-effect of generating a report in the console.

See Also

`check_archive()` checks a Darwin Core Archive via a GBIF API, rather than locally.

submit_archive *Submit a Darwin Core Archive to the ALA*

Description

The preferred method for submitting a dataset for publication via the ALA is to raise an issue on our 'Data Publication' GitHub Repository, and attached your archive zip file (constructed using `build_archive()`) to that issue. If your dataset is especially large (>100MB), you will need to post it in a publicly accessible location (such as a GitHub release) and post the link instead. This function simply opens a new issue in the users' default browser to enable dataset submission.

Usage

```
submit_archive(quiet = FALSE)
```

Arguments

`quiet`	Whether to suppress messages about what is happening. Default is set to `FALSE`; i.e. messages are shown.

Details

The process for accepting data for publication at ALA is not automated; this function will initiate an evaluation process, and will not result in your data being instantly visible on the ALA. Nor does submission guarantee acceptance, as ALA reserves the right to refuse to publish data that reveals the locations of threatened or at-risk species.

This mechanism is **entirely public**; your data will be visible to others from the point you put it on this webpage. If your data contains sensitive information, contact support@ala.org.au to arrange a different delivery mechanism.

Value

Does not return anything to the workspace; called for the side-effect of opening a submission form in the users' default browser.

Examples

```
if(interactive()){
  submit_archive()
}
```

`use_data` *Use standardised data in a Darwin Core Archive*

Description

Once data conform to Darwin Core Standard, `use_data()` makes it easy to save data in the correct place for building a Darwin Core Archive with `build_archive()`.

`use_data()` is an all-in-one function for accepted data types "occurrence", "event" and "multimedia". `use_data()` attempts to detect and save the correct data type based on the provided `tibble`/`data.frame`. Alternatively, users can call the underlying functions `use_data_occurrences()` or `use_data_events()` to specify data type manually.

Usage

```
use_data(..., overwrite = FALSE, quiet = FALSE)

use_data_occurrences(df, overwrite = FALSE, quiet = FALSE)

use_data_events(df, overwrite = FALSE, quiet = FALSE)
```

Arguments

`...`	Unquoted name of `tibble`/`data.frame` to save.
`overwrite`	By default, `use_data_events()` will not overwrite existing files. If you really want to do so, set this to `TRUE`.
`quiet`	Whether to message about what is happening. Default is set to `FALSE`.
`df`	A `tibble`/`data.frame` to save.

Details

This function saves data in the `data-publish` folder. It will create that folder if it is not already present.

Data type is determined by detecting type-specific column names in supplied data.

- Event: (`eventID`, `parentEventID`, `eventType`)
- Multimedia: not yet supported

Value

Does not return anything to the workspace; called for the side-effect of saving a `.csv` file to `/data-publish`.

See Also

`use_metadata()` to save metadata to `/data-publish`.

Examples

```
# Build an example dataset
df <- tibble::tibble(
  occurrenceID = c("a1", "a2"),
  species = c("Eolophus roseicapilla", "Galaxias truttaceus"))

# The default function *always* asks about data type
if(interactive()){
  use_data(df)
}

# To manually specify the type of data - and avoid questions in your
# console - use the underlying functions instead
use_data_occurrences(df, quiet = TRUE)

# Check that file has been created
list.files("data-publish")

# returns "occurrences.csv" as expected
```

---

use_metadata *Use a metadata statement in a Darwin Core Archive*

Description

A metadata statement lists the owner of the dataset, how it was collected, and how it can be used (i.e. its' licence). This function reads and converts metadata saved in markdown (.md), Rmarkdown (.Rmd) or Quarto (.qmd) to xml, and saves it in the `data-publish` directory.

This function is a convenience wrapper function of `delma::read_md()` and `delma::write_eml()`.

Usage

```
use_metadata(file = NULL, overwrite = FALSE, quiet = FALSE)
```

Arguments

file	A metadata file in Rmarkdown (.Rmd) or Quarto markdown (.qmd) format.
overwrite	By default, use_metadata() will not overwrite existing files. If you really want to do so, set this to TRUE.
quiet	Whether to message about what is happening. Default is set to FALSE.

Details

To be compliant with the Darwin Core Standard, the schema file **must** be called eml.xml, and this function enforces that.

Value

Does not return an object to the workspace; called for the side effect of building a file in the data-publish directory.

See Also

use_metadata_template() to create a metadata statement template; use_data() to save data to /data-publish.

Examples

```
# Get a boilerplate metadata statement
use_metadata_template(file = "my_metadata.Rmd", quiet = TRUE)

# Once editing is complete, call `use_metadata()` to convert to an EML file
use_metadata("my_metadata.Rmd", quiet = TRUE)

# Check that file has been created
list.files("data-publish")

# returns "eml.xml" as expected
```

use_schema *Create a* schema *for a Darwin Core Archive*

Description

A schema is an xml document that maps the files and field names in a DwCA. This map makes it easier to reconstruct one or more related datasets so that information is matched correctly. It works by detecting column names on csv files in a specified directory; these should all be Darwin Core terms for this function to produce reliable results. This function assumes that the publishing directory is named `"data-publish"`. This function is primarily internal and is called by `build_archive()`, but is exported for clarity and debugging purposes.

Usage

```
use_schema(overwrite = FALSE, quiet = FALSE)
```

Arguments

`overwrite`	By default, `use_schema()` will not overwrite existing files. If you really want to do so, set this to `TRUE`.
`quiet`	(logical) Should progress messages be suppressed? Default is set to `FALSE`; i.e. messages are shown.

Details

To be compliant with the Darwin Core Standard, the schema file **must** be called `meta.xml`, and this function enforces that.

Value

Does not return an object to the workspace; called for the side effect of building a schema file in the publication directory.

See Also

`build_archive()` which calls this function.

Examples

```
# First build some data to add to our archive
df <- tibble::tibble(
  occurrenceID = c("a1", "a2"),
  species = c("Eolophus roseicapilla", "Galaxias truttaceus"))

use_data_occurrences(df, quiet = TRUE)

# Now we can build a schema document to describe that dataset
```

```
use_schema(quiet = TRUE)

# Check that specified files have been created
list.files("data-publish")

# The publish directory now contains:
#  - "occurrences.csv" which contains data
#  - "meta.xml" which is the schema document
```

Index