Package: easypaper (via r-universe)

November 14, 2026

Title Automate Reproducible 'Quarto' Manuscripts Ready for Submission

Version 0.1.0

Description Creates and builds reproducible scientific manuscripts written in 'Quarto'. create_paper() writes a project that holds only the paper: its sections as separate files, its references and journal style as a CSL (Citation Style Language) file, the setup of the analysis and the data. The functions of the package then render it in the citation style of any journal that has one in CSL, check citations and cross-references before every render, export figures to PNG, JPEG and TIFF at 600 dpi, give the supplementary material a document and a bibliography of its own, and build the submission folder: the manuscript split for double-blind review, together with a data and code compendium ready for a data repository. The renders record the versions of the R packages they used, 'easypaper' included, with 'renv'. Every render also sets the scientific names of the reference list in italics, checking genera and species against the Global Biodiversity Information Facility (GBIF) Backbone Taxonomy <doi:10.15468/39omei> and Wikipedia, and reporting the cases it cannot settle; italicize_species() does the same for any other bibliography.

URL https://github.com/danielsangarci/easypaper, https://danielsangarci.github.io/easypaper/

BugReports https://github.com/danielsangarci/easypaper/issues

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements Quarto command line tool (>= 1.7) (<https://quarto.org>)

Roxygen list(markdown = TRUE)

Depends R (>= 4.1)

Imports dataspice (>= 1.2.1), flextable, haven, here, jsonlite, knitr, magick, quarto, ragg, readxl, renv, rmarkdown, rstudioapi, tools, utils, yaml, zip

Suggests base64enc, curl, maps, sf, testthat (>= 3.0.0), trackdown, vegan

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libmagick++-dev gsfonts libicu-dev libjpeg-dev libjq-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libnode-dev libx11-dev zlib1g-dev

Repository https://danielsangarci.r-universe.dev

Date/Publication 2026-11-07 18:53:57 UTC

RemoteUrl https://github.com/danielsangarci/easypaper

RemoteRef HEAD

RemoteSha e6badc4a058c813f6e20983d50943e7b78dc9c9a

Contents

add_journal . . . 3
affiliations . . . 4
check_species . . . 5
check_species_text . . . 6
checks . . . 8
clean_cache . . . 9
convert_data . . . 10
create_example_paper . . . 12
create_paper . . . 13
deposit_zenodo . . . 15
edit_metadata . . . 16
export_code . . . 18
export_figure_formats . . . 19
get_paper . . . 20
italicize_species . . . 20
list_journals . . . 23
make_preprint . . . 24
make_submission . . . 26
manuscript_stats . . . 28
render . . . 29
set_paper . . . 32
sync_licenses . . . 33
sync_metadata . . . 34
trackdown . . . 35
write_packages_bib . . . 36

Index **38**

add_journal *Add a journal's citation style to the project*

Description

The template ships one `.csl`, `journal-of-ecology.csl`, as an example. This fetches any other from the official CSL repository – a couple of thousand styles, one per journal or publisher – and puts it in the project's `references/` folder, beside the `.bib`, where `render_docx("<journal>")` and `make_submission("<journal>")` look for it.

Usage

```
add_journal(journal, path = NULL, overwrite = FALSE, repo = NULL)
```

Arguments

`journal`	The style's name in the repository, with or without `.csl`, or the URL of a `.csl` file.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.
`overwrite`	Replace a style the project already has. `FALSE` (the default) leaves it and says so.
`repo`	Where to fetch from: the official repository by default. Another URL, or the path of a local checkout of the repository, to work offline or behind a mirror.

Details

Give the name the repository uses: the file name without `.csl`, lower case with hyphens, as in `"nature"`, `"plos-one"`, `"apa"` or `"journal-of-ecology"`. The list is at `https://github.com/citation-style-language/styles`, and Zotero's style finder at `https://www.zotero.org/styles` searches it by journal name; its URLs work here too.

Most journals have a *dependent* style: a few lines of metadata pointing at the parent whose rules they share. Pandoc cannot follow that pointer, so the parent's rules are fetched and saved under the name you asked for, and the message says which parent it was.

Nothing new is installed: base R downloads the file, and it is checked to be a CSL style before it is kept.

Value

The path of the `.csl` written, invisibly.

See Also

`list_journals()` for the styles a project has.

Examples

```
# Offline, from a local copy of the repository. Here the copy is a folder
# holding a single style, the one every project ships.
dir  <- file.path(tempdir(), "my_paper")
repo <- file.path(tempdir(), "styles")
create_paper(dir, git = FALSE)
dir.create(repo, showWarnings = FALSE)
file.copy(file.path(dir, "references", "journal-of-ecology.csl"),
          file.path(repo, "my-journal.csl"))
add_journal("my-journal", path = dir, repo = repo)
list_journals(dir)
unlink(c(dir, repo), recursive = TRUE)
```

`affiliations` *The affiliations of the manuscript, under the authors*

Description

Writes the affiliations and the correspondence line of the title block, numbered from the `affiliations:` of each author in the YAML of `manuscript.qmd`. The template calls it in a chunk right under the YAML, and every render fills it in; you write the authors and never a number:

Usage

```
affiliations(path = NULL)
```

Arguments

`path`	The project, or any folder inside it. While a document is being rendered, the project of that document.

Details

```
author:
  - name: Ada Lovelace
    affiliations: [ecology]
    email: ada@example.org
    corresponding: true
  - name: Alan Turing
    affiliations: [ecology, institute]
affiliations:
  - id: ecology
    address: University X, Department of Ecology, City, Country
  - id: institute
    address: Institute Y, City, Country
```

comes out as `Ada Lovelace^1,*^, Alan Turing^1,2^` on one line, with `^1^ University X, ...`, `^2^ Institute Y, ...` and `^*^ Correspondence: ada@example.org` under it – the markdown of

the superscripts. The affiliations are numbered in the order the authors name them. Each affiliation is one line, `address:`, written as the paper prints it. It may also be written in place, quoted, as `affiliations: ["University X, City"]`, or with Quarto's own fields (`department:`, `name:`, `city:`, `country:` ...), joined department first, then the name, the address, the city and the country.

In a `.docx` they have a paragraph style of their own, *Affiliation*: the body text of the Word template without its first-line indent, which you can restyle in Word for every affiliation at once.

A double-blind main text leaves the chunk out, and the title page of `make_submission()` is given it.

Quarto's own preview, `preview()` or `quarto preview`, gets nothing from it: there Quarto draws the title block itself.

Value

The affiliations as markdown: for the chunk to print while a render of the package runs, and nothing in any other render; printed and returned invisibly when called from the console.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, authors = c("Ada Lovelace", "Alan Turing"), git = FALSE)
affiliations(dir)
unlink(dir, recursive = TRUE)
```

---

check_species *Which scientific names of the references go in italics*

Description

Every render sets the genera and species in the titles of the reference list in italics, and says – once per session – which cases it could not settle and what to write in the `.bib` to settle them. This says it again, without rendering. How the names are found is in `italicize_species()`.

Usage

```
check_species(path = NULL)
```

Arguments

`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

Unlike the checks of checks, which only read, this writes: what GBIF and Wikipedia said about each name it looks up is kept in `references/species_cache.rds`. Commit it: with it every computer renders the same italics, offline, years from now. Delete it to have every name looked up again. `italicize-species: false` in the `easypaper:` block of `_quarto.yml` turns the italics off.

Value

The bibliography files a render would hand Quarto, invisibly.

See Also

`italicize_species()`, the same for any bibliography outside a project.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
# Offline: only the cache answers, and a name it does not know stays in
# roman, with a warning. Online, a render looks each new name up once.
old <- options(easypaper.species_offline = TRUE)
if (rmarkdown::pandoc_available()) try(check_species(dir))
options(old)
unlink(dir, recursive = TRUE)
```

---

check_species_text *Check the scientific names in the text*

Description

Reads the manuscript – every section, the captions of the figures and tables, the supplement – and reports, with the file and the line, every scientific name not written the way journals ask.

Usage

```
check_species_text(
  authority = TRUE,
  abbreviate = TRUE,
  exclude = NULL,
  quiet = FALSE,
  path = NULL
)
```

Arguments

`authority`	`TRUE` (the default) asks for the authority at the first mention of each species in the main text. `FALSE` for a journal that does not want it, or gives the authorities in a table.
`abbreviate`	`TRUE` (the default) asks for the genus abbreviated after the first mention. `FALSE` for a journal that writes names in full every time.
`exclude`	Pairs of words that look like a name and are not one, or names to leave alone: `"Pinus pinea"`.
`quiet`	`TRUE` returns what it found without printing it.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

What they ask for:

- **In italics**, the genus and the epithet, abbreviated or not, and a genus on its own (*Formica*); `sp.` and `spp.` in roman (*Formica* spp.).
- **In full the first time** (*Formica rufa*), and abbreviated after that (*F. rufa*) – except at the start of a sentence, which never opens with an abbreviation, and when two genera of the text share the initial (*Formica*, *Fagus*), where *F.* would not say which. Headings and captions, which are read on their own, may give the name in full.
- **With its authority the first time** in the main text: *Formica rufa* Linnaeus, 1761, or (Linnaeus, 1761) for a species moved to another genus. The authority GBIF gives is suggested.

The abstract, the main text and the supplement are read as three documents, as a reader meets them: a name is given in full at its first mention in each. The title is only checked for italics.

Nothing is changed: whether a name in full there is a slip or a choice – a sentence that compares two species of the same genus – is yours to say. Nothing calls it either: it runs when you call it, never in a render.

Value

A data frame, invisibly when printed, with one row per thing to look at: `file`, `line`, `name` as it is written, `problem`, and `fix`, what to write instead. The species found are its `species` attribute.

How names are recognised

A pair of words shaped like a genus and an epithet is a name when GBIF lists the species, or – when it is in italics somewhere in the text – its genus and its epithet. An abbreviation (*F. rufa*) is a name when it abbreviates one of those, or GBIF knows its epithet. What GBIF said is kept in `references/species_cache.rds`, the cache of `check_species()`, so only new names need a connection. Without one, the cache answers, a pair already in italics is taken for a name, and a warning says what could not be checked.

See Also

`check_species()`, for the names in the reference list.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
writeLines(c("Colonies of Formica rufa were sampled.", "",
             "*Formica rufa* builds mounds."),
           file.path(dir, "_sections", "02_introduction.qmd"))
# Offline: only the cache answers, and the pair in italics is taken for a
# name. Online, GBIF is asked once for each new name.
old <- options(easypaper.species_offline = TRUE)
suppressWarnings(check_species_text(path = dir))
options(old)
unlink(dir, recursive = TRUE)
```

checks *Check a project before rendering it*

Description

Every render runs these first – all but `check_renv()`: a render writes `renv.lock` instead – and stops on what would otherwise reach the document broken. Call them yourself to know before waiting for a render: called on their own, they also say in one line when all is well; inside a render, only what is not.

Usage

```
check_citations(path = NULL)

check_crossrefs(quiet = FALSE, path = NULL)

check_packages(path = NULL)

check_title(quiet = FALSE, path = NULL)

check_data(quiet = FALSE, path = NULL)

check_renv(quiet = FALSE, path = NULL)
```

Arguments

`path`	The project, or any folder inside it. Left out, the one `set_paper()` named, or else the project the working directory is in: they only read.
`quiet`	`TRUE` keeps the notes to yourself and only stops or warns on what is wrong.

Details

- `check_citations()`: every `@key` cited in the text has an entry in `references/*.bib`. Package citations (`@R-pkg`) are written by the render itself, so a new one is only announced.
- `check_crossrefs()`: every `@fig-`, `@tbl-`, `@sfig-`... cited has a figure or table with that label, and says which ones exist but are never cited – journals ask for every figure to be cited in the text. A citation written inside a string of R code counts as cited.
- `check_packages()`: every package the manuscript, its sections and `R/setup.R` load with `library()` or `require()` is installed. A missing one otherwise surfaces a minute into the render, from inside Quarto.
- `check_title()`: the manuscript has a title of its own, not the template's. It is written once, in the YAML of `manuscript.qmd`.
- `check_data()`: every file the analysis names is in `data/`, and which files there nothing reads – they would travel to the data repository all the same. It reports; it never removes.

- `check_renv()`: `renv.lock` is there and matches the library in use, package by package: renv's own project library when the project has one (`renv::init()`), and otherwise the library R is using, which is the one easypaper records.

An e-mail address is not a citation, however it is written: an `@` glued to a letter or a digit – any letter, accented ones too – is an address, and `\@` is a literal `@`, which is what RStudio's visual editor writes. That is pandoc's own rule, and pandoc decides what gets cited.

Value

`check_citations()`, `check_crossrefs()` and `check_packages()` return `TRUE` invisibly and stop with an error naming what is missing. `check_title()` and `check_data()` return `TRUE` or `FALSE` invisibly, with a warning when `FALSE`. `check_renv()` returns `TRUE` in sync, `FALSE` not, with a warning naming what differs, and `NA` when it cannot tell.

See Also

`check_species()` for the scientific names of the references, and `render_docx()`, which runs every one of these first.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, title = "Ant colonies", git = FALSE)
check_citations(dir)
check_crossrefs(path = dir)
check_title(path = dir)
check_data(path = dir)
unlink(dir, recursive = TRUE)
```

---

clean_cache *Clear the knitr cache*

Description

knitr caches each chunk's results and does not notice when a file in `data/` changes. Run this after changing the data, and the next render runs every chunk again. It deletes `cache/`, `_freeze/` and `.quarto/`, which are regenerable and in `.gitignore`.

Usage

```
clean_cache(path = NULL)
```

Arguments

`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Value

The project's root, invisibly.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
clean_cache(dir)
unlink(dir, recursive = TRUE)
```

`convert_data` *Convert or copy originals into the project's data folder*

Description

The project publishes what is in `data/`: the analysis reads it, the metadata describes it and the submission compendium carries it. This brings an original into that folder in a format that will still open in twenty years. It never writes anywhere else and never touches the original.

Usage

```
convert_data(path, to = NULL, overwrite = FALSE, also = character(0))
```

Arguments

`path`	File or folder to read, as a single string: relative to the working directory – the project root, when you have opened the project's `.Rproj` – or absolute. A folder is read whole, subfolders included. The originals are only ever read.
`to`	Where the results go. Left out, the `data/` folder of the project `set_paper()` named; with no project named, it has to be given. There is no default folder: easypaper never takes the working directory for it.
`overwrite`	`FALSE` (the default) writes only what is missing from `to` or older than its original; `TRUE` rewrites the lot.
`also`	Extensions to treat as already open for this call, with or without the dot, e.g. `"las"`. For good: list them under `open-formats:` in the `easypaper:` block of the project's `_quarto.yml`.

Details

Each file takes one of three roads, decided by its extension alone:

- **Converted** to `.csv`: `.xlsx` and `.xls`, one `.csv` per sheet (needs readxl); `.sav`, `.dta` and `.sas7bdat`, one `.csv` per file (needs haven). These are closed formats with an open equivalent faithful enough to publish. A file that cannot be read – corrupt, or not really what its extension says – is named in a warning and skipped, and the rest of the batch goes on. A missing readxl or haven is reported once per call, with the files it held back.

- **Copied byte for byte**, because they already are the open format and converting one would destroy it rather than open it:
  - text and tables: `.csv`, `.tsv`, `.txt`, `.json`, `.geojson`, `.xml`, `.yml`, `.yaml`
  - containers: `.parquet`, `.nc`, `.h5`, `.hdf5`, `.sqlite`, `.db`, `.gpkg`
  - spatial: `.shp` with its sidecars (`.shx`, `.dbf`, `.prj`, `.cpg`, `.sbn`, `.sbx`, `.qix`), `.kml`, `.gml`, `.tif`, `.tiff`, `.asc`
  - sequences and trees: `.fasta`, `.fa`, `.fastq`, `.fq`, `.nwk`, `.tre`
- **Named on screen and left where it is**: anything else. A proprietary instrument file, an ArcGIS project, a photograph of a field notebook – nothing here can tell whether it belongs in the paper, or what "the table" inside it would even be.

When your field uses an open format this list has not heard of, pass its extension in `also` for a single call, or list it under `open-formats:` in the `easypaper:` block of the project's `_quarto.yml`, and every call copies it from then on.

A format conversion, not a transformation: filtering, recoding and cleaning belong in the analysis chunk that needs them, where a reader can check them. Every conversion costs something – an `.xlsx` loses its formulas, an `.sav` its value labels – which is why the original is never moved or altered.

Subfolders of a source folder are read but not reproduced: everything lands flat in `data/`. When two originals want the same name there, one keeps it – whichever comes first in alphabetical order by path – and the rest are refused with a warning saying which was kept and which was not, rather than one table quietly overwriting another. Names that differ only in case count as the same name, because the deposit has to unpack on a file system that cannot tell `Counts.csv` from `counts.csv`.

Value

The paths written, invisibly.

Examples

```
src <- file.path(tempdir(), "originals")
dir.create(src, showWarnings = FALSE)
write.csv(head(iris), file.path(src, "iris.csv"), row.names = FALSE)
out <- file.path(tempdir(), "data")

convert_data(src, to = out)   # copied: a .csv is already open

# A workbook: one .csv per sheet.
convert_data(readxl::readxl_example("datasets.xlsx"), to = out)
list.files(out)

# An extension it does not know, copied as it is when named in `also`.
writeLines("x y z", file.path(src, "points.las"))
convert_data(file.path(src, "points.las"), to = out, also = "las")
list.files(out)
unlink(c(src, out), recursive = TRUE)
```

`create_example_paper` *Create an example project: a small study, written as an easypaper paper*

Description

Writes the same project as `create_paper()`, with a short study in it instead of empty sections: the trees of Barro Colorado Island, Panama, as the vegan package distributes them (`BCI`), analysed, written up and cited. Every part of a manuscript is there, written the way the package expects it, to read, render and copy from:

Usage

```
create_example_paper(path, overwrite = FALSE, git = TRUE, open = FALSE)
```

Arguments

`path`	Directory to create, as a single non-empty string; `~` is expanded. Its base name becomes the name of the `.Rproj` file. A path that exists as a file is refused.
`overwrite`	Write into a directory that already has files in it. `FALSE` (the default) refuses, so an existing manuscript is never silently overwritten.
`git`	Initialise a git repository and make the first commit. `TRUE` by default: it is local and private, costs a second, and gives the manuscript a history from its first line instead of from the day you remember to start one. It has nothing to do with GitHub, which is a later and separate decision. Needs `git` on the PATH and a configured `user.name` / `user.email`; if either is missing, the project is still created and you are told what to run.
`open`	Open the new project in RStudio. Needs an RStudio session and the rstudioapi package; outside one, the project is created and a message says so.

Details

`manuscript.qmd`	`title, short title, authors with shared affiliations and their ORCID iDs, the corresponding author`
`_sections/01_abstract.qmd`	`the abstract, and the keywords under it`
`_sections/02_introduction`	`citations in brackets and in the sentence`
`_sections/03_methods`	`the text of the Methods, and a chunk that reads the data from data/ for every analysis`
`_sections/04.1_richness`	`each analysis beside its text, in a file named`
`_sections/04.2_composition`	`after it: a GLM of richness; a PERMANOVA, a PERMDISP and an NMDS of composition; numbers written by inline R, never typed; scientific names written by R, in italics`
`_sections/10_figures`	`two figures: an NMDS with names in italics`
`_sections/11_tables`	`two flextables, with the three rules`

```
_sections/12.1_suppl        a supplementary figure and table
_sections/06-09             the statements at the end, filled in
data/                       the two tables of BCI, as .csv
references/                 the five references the text cites, one
                            with a species in its title, set in italics
                            by every render; the style is the
                            template's, Journal of Ecology
```

`create_paper()` stays the one to start a paper of your own with: it writes the structure and nothing else.

It needs one package, vegan: its `BCI` and `BCI.env` data are written into `data/`, and the analysis uses it. When it is missing, the function stops before writing anything, and says what to install. The data were collected by Condit et al. (2002), *Science* 295: 666–669, doi:10.1126/science.1066854: cite them, not this example, if you use them. The authors of the example are borrowed from the history of the field – Charles Darwin and Alfred Russel Wallace, who never wrote it – and their affiliations, emails and ORCID iDs are made up.

Value

The absolute path of the created project, invisibly.

See Also

`create_paper()`, for a project of your own.

Examples

```
if (requireNamespace("vegan", quietly = TRUE)) {
  dir <- file.path(tempdir(), "example_paper")
  create_example_paper(dir, git = FALSE)
  list.files(file.path(dir, "data"))
  unlink(dir, recursive = TRUE)
}
```

---

create_paper *Create a reproducible Quarto manuscript project*

Description

Writes, into `path`, the structure of a reproducible manuscript: the sections as separate `.qmd` files, the references and the journal's citation style, `R/setup.R` for the analysis, and the folder for the data. Nothing else: the functions that render and assemble it – `render_docx()`, `make_submission()` and the rest – live in this package, so the project holds only the paper.

Usage

```
create_paper(
  path,
  title = NULL,
  authors = NULL,
  overwrite = FALSE,
  git = TRUE,
  open = FALSE
)
```

Arguments

path	Directory to create, as a single non-empty string; ~ is expanded. Its base name becomes the name of the .Rproj file. A path that exists as a file is refused.
title	Manuscript title, written into the YAML of manuscript.qmd, the one place it lives. Quotes and backslashes are escaped for YAML, so a LaTeX fragment such as \textit{Formica} survives. A vector is joined with spaces. The heading of the project's README.md follows it. NULL or "" leaves the placeholder, which every render warns about.
authors	Character vector of author names, in order, or one comma-separated string, which is how the RStudio wizard sends them. The first one is marked as the corresponding author. Affiliations are not guessed: each author gets a placeholder to fill in, in the affiliations: of the YAML of manuscript.qmd (see affiliations()), and an empty orcid: for the ORCID iD. The project's README.md lists them from the first commit. NULL or an empty string leaves the template's.
overwrite	Write into a directory that already has files in it. FALSE (the default) refuses, so an existing manuscript is never silently overwritten.
git	Initialise a git repository and make the first commit. TRUE by default: it is local and private, costs a second, and gives the manuscript a history from its first line instead of from the day you remember to start one. It has nothing to do with GitHub, which is a later and separate decision. Needs git on the PATH and a configured user.name / user.email; if either is missing, the project is still created and you are told what to run.
open	Open the new project in RStudio. Needs an RStudio session and the rstudioapi package; outside one, the project is created and a message says so.

Details

```
manuscript.qmd       title, authors, journal, and the list of sections
supplementary.qmd    the supplement, as its own document
_quarto.yml          formats, and the settings of easypaper
_sections/           the text, one file per section
R/setup.R            seed, palette and helpers for the analysis
data/                the data, in open formats: what gets published
references/          the .bib you cite from, and the journal styles (.csl)
LICENSE.txt  LICENSE-CODE.txt  README.md  <name>.Rproj
```

The `easypaper:` block of `_quarto.yml` records the version of easypaper that created the project, and every render records the one that built each document in `renv.lock`, with every other package the paper used: `renv::restore()` brings them back years later.

Every argument is checked before anything is written, so a wrong call stops with a message naming the argument and leaves no half-made project behind.

Value

The absolute path of the created project, invisibly.

See Also

`create_example_paper()` for the same project with a small study written in it, to see how each part is written; `render_html()` to see what you have, `convert_data()` to bring the data into the project's `data/` folder, and `vignette("easypaper")` for a tour of what the project can do.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir,
             title   = "Manuscript title here",
             authors = c("Charles Darwin", "Alfred Russel Wallace"),
             git     = FALSE)
list.files(dir)
unlink(dir, recursive = TRUE)
```

---

deposit_zenodo *Reserve the DOI of the data deposit on Zenodo*

Description

Uploads the data and code compendium – `data_and_code.zip`, which `make_submission()` and `make_preprint()` build – to a new Zenodo deposit, fills in its description from the project, and reserves its DOI, so the manuscript can cite the data before they are published. It never publishes: the deposit stays a draft until you review it on Zenodo and press Publish there, which cannot be undone. Nothing in the package needs it; it runs only when you call it.

Usage

```
deposit_zenodo(file = NULL, sandbox = FALSE, token = NULL, path = NULL)
```

Arguments

`file`	The compendium to upload. `NULL`, the default, takes the newest `data_and_code.zip` in `submission/`.
`sandbox`	`TRUE` deposits on `https://sandbox.zenodo.org`, to try it.
`token`	The personal access token. `NULL`, the default, reads `ZENODO_TOKEN`, or `ZENODO_SANDBOX_TOKEN` with `sandbox = TRUE`.

`path` The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

The description comes from the manuscript: the title (the short title, `short-title:`, when there is one; the title otherwise), the authors with their affiliations and ORCID (`orcid:` on an author), the keywords, and the licence of the data, from the project's `LICENSE.txt` (CC BY 4.0 in a new project). Zenodo refuses a whole deposit over one ORCID it cannot read, so one that is not four groups of four digits – `0000-0000-0000-000X`, the last may be an X – is left out of it, with a warning naming the author: add it on the draft, or correct it in `manuscript.qmd` and call again. The deposit and its DOI are recorded in the `easypaper:` block of `_quarto.yml`, so a second call – after the data changed – replaces the file of the same draft instead of opening another. A deposit already published is left alone: a new version of it is made on Zenodo.

The DOI is written into the text wherever it says `10.5281/zenodo.XXXXXXX` – the placeholder of the data availability statement –, into the cover letters of `submission/` where they say `[repository DOI]`, and into the project's `README.md`, as a badge and in its citation; and it is printed, to cite it wherever else it belongs. A cover letter written after it carries the DOI from the start.

It needs a personal access token of Zenodo, with the scopes `deposit:write` and `deposit:actions`: create one at https://zenodo.org/account/settings/applications/tokens/new/ and put it in your `.Renviron` as `ZENODO_TOKEN=...` (`usethis::edit_r_environ()` opens it). To try it first, use Zenodo's sandbox, a copy of the site for tests: a token from https://sandbox.zenodo.org as `ZENODO_SANDBOX_TOKEN`, and `sandbox = TRUE`. A sandbox DOI does not resolve, so it is never written into the text.

Value

A list with the deposit's `id`, the reserved `doi` and the `url` of the draft, invisibly.

Examples

```
# It needs a Zenodo token (see Details), Quarto for the compendium, and a
# connection, so it does not run here.
## Not run:
make_submission(path = "my_paper")
deposit_zenodo(sandbox = TRUE, path = "my_paper")
deposit_zenodo(path = "my_paper")

## End(Not run)
```

`edit_metadata` *Describe the data deposit by hand*

Description

Opens dataspice's editor for one of the four files that describe the data deposit, in `data/metadata/`: the variables (`"attributes"`: their description and units), the dataset itself (`"biblio"`: description, temporal and geographic coverage), the people (`"creators"`) or the files (`"access"`: the `name` of each is its file name without the extension, to change if you like). Save in the editor and close it. `sync_metadata()` has already filled in what the project knows – title, keywords, the authors with their affiliations, email and ORCID, variable names – so what is left is what no machine can guess.

Usage

```
edit_metadata(
  what = c("attributes", "biblio", "creators", "access", "write"),
  path = NULL
)
```

Arguments

`what`	Which file to edit: `"attributes"` (the default), `"biblio"`, `"creators"` or `"access"`; or `"write"` to compile them.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

`what = "write"` opens nothing: it compiles the four files into `data/metadata/dataspice.json`, the machine-readable description, and writes `data/metadata/index_metadata.html`, a page to read it. The area the data cover, when `biblio.csv` gives its coordinates, is drawn on that page as a map inside it, which opens offline: it needs sf, maps and base64enc, and without them the page has no map, and says so: it never loads one from the internet. You need it only to look at them: `make_submission()` and `make_preprint()` compile them again before they build the data compendium, which carries both.

Value

The metadata folder, invisibly.

See Also

`sync_metadata()`, and https://docs.ropensci.org/dataspice/ for what each field means.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
write.csv(head(iris), file.path(dir, "data", "iris.csv"), row.names = FALSE)

edit_metadata("write", path = dir)   # dataspice.json and its web page
list.files(file.path(dir, "data", "metadata"))
```

```
# The editors open in an interactive session.
if (interactive()) {
  edit_metadata("attributes", path = dir)
  edit_metadata("biblio", path = dir)
}
unlink(dir, recursive = TRUE)
```

`export_code` *Write the analysis code as one script*

Description

Extracts the R code of `R/setup.R` and of every section, in the order the manuscript includes them, into `output/analysis_code.R`, with a header per section. It opens with a table of contents: the line each section starts on, and the labels of its chunks, so each analysis is found at a glance – the richness model in Results, part 1, say. Beside it goes `output/sessionInfo.txt`, with the versions of R, of every package and of Quarto, and a copy of `renv.lock` when there is one. `render_all()` and `make_submission()` run it on their own; the submission compendium carries what it writes.

Usage

```
export_code(path = NULL)
```

Arguments

`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Value

The path of `analysis_code.R`, invisibly.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
export_code(dir)
list.files(file.path(dir, "output"))
unlink(dir, recursive = TRUE)
```

`export_figure_formats` *Copy every figure to JPEG and TIFF*

Description

knitr writes one format per render, PNG, into `figures/png/`. This converts each of them into `figures/jpg/` and `figures/tiff/`, at the resolution they were drawn at – the `dpi` of `_quarto.yml`, 600 in a new project – and with the TIFF compressed losslessly (LZW), which is what journals ask for on acceptance. Converting is faster than rendering three times, and guarantees the three copies are the same figure. Every render does it on its own.

Usage

```
export_figure_formats(quiet = FALSE, path = NULL)
```

Arguments

`quiet`	`TRUE` says nothing.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

JPEG is lossy and a poor choice for line art or figures with text. It is here because some journals require it; if you get to choose, send the TIFF.

Value

The PNG files converted, invisibly.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
dir.create(file.path(dir, "figures", "png"), recursive = TRUE)
grDevices::png(file.path(dir, "figures", "png", "fig-a-1.png"))
plot(1:10)
invisible(grDevices::dev.off())
export_figure_formats(path = dir)
list.files(file.path(dir, "figures"), recursive = TRUE)
unlink(dir, recursive = TRUE)
```

get_paper *Which paper is named*

Description

The project `set_paper()` named for this session: the one every function works on when it is given no `path`.

Usage

```
get_paper()
```

Value

The project's root, or `NULL` when none is named; printed in a sentence when called from the console, and returned invisibly then.

See Also

`set_paper()`

Examples

```
get_paper()   # NULL: none named yet

dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
old <- set_paper(dir)
get_paper()
set_paper(old)
unlink(dir, recursive = TRUE)
```

italicize_species *Set the scientific names of a bibliography in italics*

Description

Reference lists come out of Quarto and R Markdown with genus and species names in roman, because citeproc cannot tell a scientific name from any other word. This rewrites a bibliography so that it can: every genus, species, abbreviation (*Q. suber*) and subspecies in the titles is found, checked against the GBIF Backbone Taxonomy, and marked in italics in a copy of the bibliography that you then hand to Quarto or R Markdown instead of the original. The original is only ever read.

Usage

```
italicize_species(
  input,
  output,
  names = NULL,
  exclude = NULL,
  gbif = TRUE,
  keep_case = TRUE,
  protect = NULL,
  cache = .sp_default_cache(),
  fields = c("title", "title-short", "container-title", "collection-title",
    "original-title"),
  quiet = FALSE
)
```

Arguments

input	The bibliography to read: a `.bib` (BibTeX or BibLaTeX) or a `.json` (CSL-JSON, as Zotero exports it).
output	The CSL-JSON file to write, which is the one the document must name in `bibliography:`, e.g. `"references_italic.json"`.
names	Names to set in italics whatever GBIF says, for the rare one it does not know: a species ("Pinus halepensis"), an abbreviation ("Q. ruber") or a genus on its own ("Quercus"). Rarely needed.
exclude	Names found that must stay in roman.
gbif	`FALSE` looks nothing up, and only `names` is set in italics.
keep_case	`TRUE`, the default, keeps every title exactly as the bibliography writes it, whatever the style. `FALSE` lets pandoc and the style change it the way they normally would: pandoc lowercases the titles of a `.bib`, a style like Chicago capitalises every word. Use `protect` then for the words that must keep their form. Scientific names keep theirs either way.
protect	Words or phrases whose capitals are kept in any style: "Spain", "Mediterranean", "DNA".
cache	The `.rds` file where the answers of GBIF and Wikipedia are kept, or `NULL` to keep none. By default one file in the user's cache directory (`tools::R_user_dir()`), shared by every bibliography.
fields	The CSL fields of each reference to look into. A `container-title` is only read when it is a book's, the one a chapter belongs to: a journal's name is never touched.
quiet	`TRUE` says nothing, except when there was no connection: that warning is always raised, because the result may then be incomplete.

Details

Inside an easypaper project there is nothing to call: every `render_*()` runs this on the project's bibliography first. This is the same function, for a bibliography anywhere else. Point the document at the file it writes:

```
bibliography: references_italic.json

# first chunk of the .qmd or .Rmd: it runs before pandoc reads the file
easypaper::italicize_species("references.bib", "references_italic.json")
```

Value

The path of `output`, invisibly, with three attributes: `italicized`, the names set in italics; `doubtful`, a data frame of the cases left for you to decide (`id`, `word`, `reason`, `line`); and `offline`, `TRUE` when something could not be looked up.

How names are found

Nothing has to be listed by hand. Anything in a title shaped like a scientific name is looked up in GBIF, and in the English Wikipedia where a word could also be something else:

- **Genus and epithet** ("Quercus suber"): the species exists in GBIF, or its genus and its epithet both do – so a combination GBIF lacks still counts. An epithet written in capitals ("Helicobacter Pylori") is lowercased.
- **New species** ("Haemoproteus trarotraro n. sp."): a known genus followed by `n. sp.`, `sp. nov.` or `comb. nov.`.
- **Abbreviations** ("Q. suber"): the epithet exists in GBIF; the genus need not be spelled out anywhere.
- **Subspecies and varieties** ("Quercus ilex subsp. ballota", "Epilachna sparsa orientalis"): the infraspecific epithet in italics, the rank (`subsp.`, `var.`, `f.`) in roman.
- **A genus on its own** ("Wolbachia infections", "Quercus spp."): it appears in a binomial elsewhere in the bibliography, is followed by `sp.`, `spp.` or `species` or preceded by "genus", or Wikipedia describes it as a genus. That last check is what keeps "China", "Andes" and "America" – all animal genera in GBIF – in roman.
- **A genus that is also an English common name** (gorilla, bison, lynx, eucalyptus, according to GBIF's vernacular names): only in italics where the capital says it is the genus, halfway through a sentence-case title ("Seasonal diet of Gorilla in Gabon"). At the start of a title, or in a title-case one ("Gorilla Conservation in Africa"), the capital says nothing and the word is read as the common name.

Higher taxa, virus names, genes and Latin phrases (*in vitro*) are left alone: they are not genera or species.

Doubtful cases

When a word could be a genus and the title cannot tell – a common name in a position that says nothing, a word Wikipedia gives several meanings ("Iris", "Rosa"), a binomial GBIF does not list – it is left in roman and reported, with the line of the file it is on and what to write there. The mark is made in the bibliography itself and affects that reference only:

- in a `.bib`: `\textit{Gorilla}` (or `\emph{}`) for italics, `\textup{Gorilla}` (or `\textrm{}`, `\textnormal{}`) for roman;
- in CSL-JSON: `<i>Gorilla</i>` and `<span style="font-style:normal;">Gorilla</span>`.

A marked word is not reported again.

The cache, and working offline

Every answer from GBIF and Wikipedia is kept in `cache`, so only a name never seen before needs a connection: the first run over a hundred new references takes a couple of minutes, the next ones a fraction of a second. Without a connection nothing fails: the cache is used, whatever could not be checked is left in roman, and a warning says so. What could not be checked is not written to the cache, so running again online completes it. `options(easypaper.species_offline = TRUE)` works that way on purpose, with a connection or without: nothing is looked up, and only the cache answers.

The cache is small, about 5 KB per hundred references. Kept in git beside the bibliography, it makes the result the same on every computer, with or without a connection, and years later however GBIF and Wikipedia change. That is where an easypaper project keeps it: `references/species_cache.rds`.

See Also

`create_paper()`, whose projects run this before every render.

Examples

```
# A CSL-JSON bibliography, as Zotero exports it. With gbif = FALSE nothing
# is looked up, and only the names given are set in italics.
refs <- file.path(tempdir(), "refs.json")
writeLines('[{"id": "perez2020", "type": "article-journal",
  "title": "Effects of fire on Pinus halepensis regeneration"}]', refs)
out <- file.path(tempdir(), "refs_italic.json")
italicize_species(refs, out, names = "Pinus halepensis", gbif = FALSE,
                  cache = NULL)
jsonlite::read_json(out)[[1]]$title

# A .bib, which pandoc reads. With gbif = TRUE, the default, every name is
# looked up in GBIF instead, and the answers are kept in `cache`.
bib <- file.path(tempdir(), "refs.bib")
writeLines(c("@article{perez2020, author = {Perez, A.}, year = {2020},",
             "  title = {Effects of fire on Pinus halepensis regeneration},",
             "  journal = {Forest Ecology}}"), bib)
if (rmarkdown::pandoc_available()) {
  italicize_species(bib, out, names = "Pinus halepensis", gbif = FALSE,
                    cache = NULL)
  jsonlite::read_json(out)[[1]]$title
}
unlink(c(refs, bib, out))
```

---

list_journals *The journals a project can render for*

Description

Every `.csl` in the project's `references/` folder, by the name the other functions take: the file name without `.csl`. `add_journal()` fetches any other journal's style.

Usage

```
list_journals(path = NULL)
```

Arguments

path	The project, or any folder inside it. Left out, the one `set_paper()` named, or else the project the working directory is in: this only reads.

Value

A character vector of journal names, sorted.

See Also

`add_journal()`, and the `journal` argument of `render_docx()` and `make_submission()`.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
list_journals(dir)
unlink(dir, recursive = TRUE)
```

---

make_preprint *Build a preprint deposit*

Description

`make_submission()`'s sibling, for the other destination. Writes `submission/<label>/`: the manuscript as ONE signed PDF – a preprint carries its authors, and is not reviewed blind – its supplement as PDF too, every figure on its own, and the data and code compendium, zipped. There is no cover letter: there is no editor. Nor is there a short title, even when the manuscript has one: a running head is what a journal asks for, not a preprint server. The supplement names the server where a journal's would name the journal – "Preprint, *bioRxiv*" – since the paper is in no journal yet. A `README.md` inside says what goes to the preprint server and what goes to the data repository, and what to do before pressing submit.

Usage

```
make_preprint(
  journal = NULL,
  label = "bioRxiv",
  caption_style = "default",
  figure_format = "tiff",
  snapshot = TRUE,
  suppl_figures = "separate",
  line_numbers = TRUE,
  line_spacing = NULL,
```

```
  suppl_line_spacing = NULL,
  figure_dpi = NULL,
  path = NULL
)
```

Arguments

journal	The citation style. A preprint has no house style, so this is only about which convention you prefer to read. NULL takes the manuscript's own.
label	Names the folder inside submission/ and every file in it. Defaults to the server you are most likely to post to; change it for another one. Never overwritten: a second call builds the next version beside it, bioRxiv_v2, with its files named the same way.
caption_style	How figures and tables are named: "default", "abbrev", "colon", "compact", "nature", or one of your own (see render_docx()).
figure_format	The format of the standalone figures: "tiff" (what journals ask for), "png" or "jpg".
snapshot	TRUE (the default) records renv.lock before building the compendium.
suppl_figures	"separate" (the default) leaves the supplementary figures and tables in the supplement, and rewrites their citations in the main text to "Figure S1". "main" keeps them at the end of the main text. Supplementary text (an extended Methods) always goes out on its own, with its own reference list.
line_numbers	TRUE (the default) numbers every line of the manuscript PDF, continuously, which is what readers and reviewers of a preprint cite; FALSE leaves them unnumbered.
line_spacing, suppl_line_spacing	The line spacing of the manuscript and of the supplement: 1, 1.5 or 2. NULL, the default, keeps the linestretch of the pdf format in _quarto.yml – 1.5 in the template.
figure_dpi	The resolution the figures are drawn at, in dpi, for this call: NULL (the default) keeps the dpi of the knitr block of _quarto.yml, 600 in a new project. The standalone figures are named with it and with the size they print at: Figure_1_600dpi_180x120mm.tiff.
path	The project, or any folder inside it. Left out, the one set_paper() named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

It needs a LaTeX installation for the PDF: tinytex::install_tinytex() is enough.

Value

The path of the deposit, invisibly.

See Also

make_submission().

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, title = "Ant colonies", authors = "Ada Lovelace",
             git = FALSE)

# It renders the manuscript, so it needs Quarto (https://quarto.org).
## Not run:
make_preprint(path = dir)
make_preprint(label = "EcoEvoRxiv", path = dir)
list.files(file.path(dir, "submission"))

## End(Not run)
unlink(dir, recursive = TRUE)
```

`make_submission` *Build the folder you send to a journal*

Description

Writes `submission/<label>/`: the manuscript split the way journals with double-blind review ask for – a title page, and a main text that opens with the title and names nobody – or, signed, the main text alone; the supplement as its own document, every figure of the main text on its own, named with its resolution and printed size – the supplementary figures stay inside the supplement, which is how journals take them – a cover letter – dated, with the title of the manuscript, signed by the corresponding author, and with the DOI of the data once `deposit_zenodo()` has reserved it; for a second journal, the text of the last letter with those brought up to date – a checklist of what is left to do by hand, and the data and code compendium, zipped, ready for Zenodo or Dryad. Everything in it is rebuilt on every call except the cover letter, which is never overwritten.

Usage

```
make_submission(
  journal = NULL,
  label = NULL,
  caption_style = "default",
  figure_format = "tiff",
  blind = TRUE,
  snapshot = TRUE,
  suppl_figures = "separate",
  line_numbers = TRUE,
  line_spacing = 2,
  suppl_line_spacing = 1.5,
  figure_dpi = NULL,
  path = NULL
)
```

Arguments

journal	The citation style, by the name of its .csl without the extension (see list_journals()). NULL, the default, takes the one the manuscript declares in its csl: line.
label	Names the folder inside submission/ and every file in it. Left alone it is the journal's own name with the spaces taken out: "ecology-letters" gives submission/EcologyLetters/. A submission is never overwritten: when the folder is already there, the next version is built beside it – JournalofEcology_v2, _v3 – with its files named the same way. Pass your own label for a trial you want to keep apart.
caption_style	How figures and tables are named: "default", "abbrev", "colon", "compact", "nature", or one of your own (see render_docx()).
figure_format	The format of the standalone figures: "tiff" (what journals ask for), "png" or "jpg".
blind	TRUE (the default) splits the manuscript for double-blind review: a title page with the authors and the sections that identify them, a main text that names nobody, and a copy of the compendium that names nobody either. FALSE sends one signed main text, with all of it, and no title page.
snapshot	TRUE (the default) records renv.lock before building the compendium. FALSE leaves a lockfile you maintain by hand alone.
suppl_figures	"separate" (the default) leaves the supplementary figures and tables in the supplement, and rewrites their citations in the main text to "Figure S1". "main" keeps them at the end of the main text. Supplementary text (an extended Methods) always goes out on its own, with its own reference list.
line_numbers	TRUE (the default) numbers every line of the main text and of the title page, continuously, which is what reviewers cite. FALSE leaves them unnumbered.
line_spacing	The line spacing of the main text and of the title page: 2 (the default, double), 1.5 or 1.
suppl_line_spacing	The line spacing of the supplement: 1.5 (the default), 1 or 2. Its lines are not numbered.
figure_dpi	The resolution the figures are drawn at, in dpi, for this call: NULL (the default) keeps the dpi of the knitr block of _quarto.yml, 600 in a new project. The standalone figures are named with it and with the size they print at: Figure_1_600dpi_180x120mm.tiff.
path	The project, or any folder inside it. Left out, the one set_paper() named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

It records renv.lock first: the project's records the environment this submission came out of, the version of easypaper that built it included, and the compendium gets one of its own, with the packages its analysis loads and what they need, and not easypaper, which the analysis does not run.

Double-blind review also needs the sections that identify you off the main text. They move to the title page: by default the Acknowledgements, the CRediT statement, the conflict of interest statement and the data availability statement, which is the list blinded-sections: sets in the

`easypaper:` block of `_quarto.yml`, in the order they come out on the title page. The page is built from the manuscript and needs no file of its own; a `title_page.qmd` in the project, to add a running head or a word count, is used as the page.

The compendium is signed: `data_and_code.zip` is the one deposited, and a deposit names its authors. A double-blind submission also gets `data_and_code_blinded.zip`, for the reviewers: the same compendium with the authors taken out of what the package writes into it – the creators of the metadata (`creators.csv`, `dataspice.json` and its page), the copyright line of `LICENSE-CODE.txt`, the README – and out of `renv.lock`, which records the authors of every package it lists and the pages of each, and easypaper itself, which the analysis does not use and which, installed from GitHub, names the account it came from. A name, email, ORCID, affiliation or account of an author still found in it – in a script, in the data, a package of yours installed from GitHub – is reported with the file it is in, to take out of the project by hand. An account is the user part of an author's email and the owner of the project's GitHub repository.

Value

The path of the submission folder, invisibly.

See Also

`make_preprint()` for a preprint server, and `render_docx()` for the documents alone.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, title = "Ant colonies", authors = "Ada Lovelace",
             git = FALSE)

# It renders the manuscript, so it needs Quarto (https://quarto.org).
## Not run:
make_submission(path = dir)
make_submission(path = dir)
make_submission("ecology-letters", figure_format = "png", blind = FALSE,
                path = dir)
list.files(file.path(dir, "submission"))

## End(Not run)
unlink(dir, recursive = TRUE)
```

---

manuscript_stats *A summary of the manuscript*

Description

What a journal asks for on its submission form: the characters of the title and of the short title when there is one, with and without spaces; the words of the abstract; the keywords; the words of the main text, without and with its reference list; the references cited; and the figures and tables of the paper and of its supplement. Every render and every `make_*()` prints it when it is done, once; this prints it on its own.

Usage

```
manuscript_stats(quiet = FALSE, path = NULL)
```

Arguments

quiet	TRUE returns the counts without printing them.
path	The project, or any folder inside it. Left out, the one set_paper() named, or else the project the working directory is in: this only reads.

Details

Words are counted the way Word counts them in the rendered document, not in the source: a citation counts as the text the journal's style prints for it – [@smith2020] as "(Smith 2020)" – a cross-reference as "Figure 1", and code, comments and markup not at all. The abstract is the # Abstract section without its keywords line – or # Summary, as the journals of the British Ecological Society call it, or # Resumen; any other heading is named with abstract-heading: in the easypaper: block of _quarto.yml. The main text is every section from the one after the abstract to the reference list, less the statements at the end – acknowledgements, funding, author contributions, conflicts of interest, data availability, ethics – whatever they are called: Introduction to Discussion, in the template. The reference list is the one the paper prints, in the journal's style. The value of inline R code is not known until it runs, and counts as one word.

Value

A named integer vector, invisibly when printed: title_chars, title_chars_no_spaces, short_title_chars and short_title_chars_no_spaces (NA without one), abstract_words, keywords, main_words, main_words_with_refs, references, figures, tables, suppl_figures and suppl_tables.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, title = "Ant colonies", git = FALSE)
if (rmarkdown::pandoc_available()) manuscript_stats(path = dir)
unlink(dir, recursive = TRUE)
```

---

render *Render the manuscript*

Description

Each render runs the checks first (checks), keeps the licences and the deposit's metadata in step with the manuscript, sets the scientific names of the references in italics, renders with Quarto, repairs what Word would refuse to open, and copies every figure to figures/ as PNG, JPEG and TIFF at the dpi of _quarto.yml, 600 in a new project. What it writes goes to the project's output/ folder.

Usage

```
render_docx(
  journal = NULL,
  caption_style = "default",
  suppl_figures = "separate",
  path = NULL
)

render_pdf(
  journal = NULL,
  caption_style = "default",
  suppl_figures = "separate",
  path = NULL
)

render_html(journal = NULL, caption_style = "default", path = NULL)

render_supplementary(
  journal = NULL,
  caption_style = "default",
  output_format = "docx",
  files = NULL,
  blind = FALSE,
  path = NULL
)

render_all(journal = NULL, caption_style = "default", path = NULL)

preview(path = NULL)
```

Arguments

journal	The citation style, by the name of its `.csl` without the extension (see `list_journals()`). `NULL`, the default, takes the one the manuscript declares in its `csl:` line. Naming one here overrides that for this call only, and changes no file.
caption_style	How figures and tables are named, in their captions and where the text cites them: `"default"` is what the `crossref:` block of `_quarto.yml` says (Figure 1. and Table 1. in a new project). The others are `"abbrev"` (Fig. 1.), `"colon"` (Figure 1:), `"compact"` (Fig. 1:) and `"nature"` (Figure 1 |), or one you define, under a name of your choosing, in `caption-styles:` of the `easypaper:` block of `_quarto.yml`. The text cites with the caption's word unless the style gives it one of its own – `fig-prefix` beside `fig-title` in `crossref:`, or `fig-text` in a caption style – "Figure 1." under the figure, "Fig. 1" in the text. It does not follow `journal`: a citation style says nothing about figures or tables.
suppl_figures	`"separate"` (the default) leaves the supplementary figures and tables in the supplement, and rewrites their citations in the main text: `@sfig-map` becomes "Figure S1", and `-@sfig-map`, the number alone, "S1". `"main"` keeps them at the end of the manuscript instead.

path	The project, or any folder inside it. Left out, the one set_paper() named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.
output_format	"docx", "pdf" or "html".
files	The supplementary sections to render. NULL, the default, renders every supplementary section: the numbered files of _sections/ with suppl in their name, 12.1_suppl_material.qmd in the template.
blind	TRUE leaves the authors off the supplement, for one that travels with a double-blind submission.

Details

Every document is named after the project, the folder its .Rproj is in: manuscript_<project>, supporting_information_<project>. Rendered in the style of a journal other than the manuscript's own, the journal is added – manuscript_<project>_ecology-letters.docx – so it leaves yours as it was. Rename the folder and the next render follows.

- render_docx() writes output/manuscript_<project>.docx, on the project's Word template, and the supplement beside it, as render_supplementary() does.
- render_pdf() writes output/manuscript_<project>.pdf, its lines numbered as in the .docx, and the supplement beside it as .pdf: the title block flush left, with an ORCID icon beside each author who has one, the captions justified, and the Greek letters of the text printed, which LaTeX's font lacks. It needs a LaTeX installation: tinytex::install_tinytex() is enough.
- render_html() writes output/manuscript_<project>.html: the working copy, in seconds instead of minutes. It does not record renv.lock, so rendering an old version to look into something never rewrites your record.
- render_supplementary() writes the supplement on its own: output/supporting_information_<project>.docx, or one file per supplementary section when there are several (_sections/12.1_suppl_*.qmd, 12.2_suppl_*.qmd ...). Its own numbering (Figure S1...), its own Word template and its own reference list, left out when it cites nothing. It opens with "Supporting Information" and, under it, the reference of the paper it belongs to.
- render_all() runs render_docx(), render_pdf() and export_code(), in that order: every document to read, the supplements included, into output/. It does not build a submission: that is make_submission().
- preview() opens a live preview in the browser, rendered again every time you save a .qmd.

The manuscript and its supplement always come out as separate documents, each with its own reference list. That is what a journal asks for, and the only way the tables survive: merging them means handing both to pandoc, which rebuilds the document and loses every column width.

The Word templates ship with the package. To use your own, put a file of the same name – word_plain_paper_style.docx for the manuscript, word_plain_paper_style_supplementary_material.docx for the supplement, word_cover_letter.docx for the letter – in a format/ folder in the project; it is used instead.

Value

The path of the file written, invisibly: several for `render_supplementary()` when there are several supplements. `render_all()` returns the `output/` folder, and `preview()` nothing useful.

See Also

`make_submission()` and `make_preprint()` for what you send, and checks for what every render verifies first.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, title = "Ant colonies", authors = "Ada Lovelace",
             git = FALSE)

# The renders need Quarto, a program of its own (https://quarto.org).
## Not run:
render_html(path = dir)
render_docx(path = dir)
render_docx("ecology-letters", "abbrev", path = dir)
render_pdf(path = dir)
render_supplementary(path = dir)
render_all(path = dir)
list.files(file.path(dir, "output"))

## End(Not run)
unlink(dir, recursive = TRUE)
```

---

set_paper *Name the paper to work on*

Description

The functions that write into a project – the renders, the submissions, `convert_data()` and the rest – write into the one you name: here, once a session, or with `path` in each call. easypaper never takes the working directory for it: with neither, a function that writes stops before writing anything. The functions that only read, such as `check_citations()` or `manuscript_stats()`, use the project named here too, and otherwise the one the working directory is in.

Usage

```
set_paper(path)
```

Arguments

`path`	The project, or any folder inside it: `"."` from the project's folder. `NULL` forgets the one named, and then nothing is written until a project is named again.

Details

Once the project's `.Rproj` is open, the working directory is its root, so `set_paper(".")` names it. The project stays named until the session ends or you name another; `path` in a call works on that project for that call alone.

Value

The project named before, invisibly – `NULL` if none was – so that it can be put back.

See Also

`get_paper()`, which says which project is named, and `create_paper()`, which makes one.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, git = FALSE)
old <- set_paper(dir)
get_paper()
list_journals()
set_paper(old)
unlink(dir, recursive = TRUE)
```

---

sync_licenses *Copy the authors into the licences*

Description

The authors in the YAML of `manuscript.qmd` are the copyright holders of the project: this writes them into the copyright line of `LICENSE-CODE.txt` (MIT) and into the licence notice of `README.md`, between its `<!-- license:start -->` and `<!-- license:end -->` markers. `LICENSE.txt` (CC BY 4.0) is the official text and is never touched. Every render does this on its own; it can be run as many times as you like.

Usage

```
sync_licenses(year = format(Sys.Date(), "%Y"), quiet = FALSE, path = NULL)
```

Arguments

`year`	The year of the copyright line. This year by default.
`quiet`	`TRUE` says nothing.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Value

The holders as written, invisibly.

See Also

`sync_metadata()`, which does the same for the data deposit.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, authors = c("Ada Lovelace", "Alan Turing"), git = FALSE)
sync_licenses(path = dir)
readLines(file.path(dir, "LICENSE-CODE.txt"), n = 3)
unlink(dir, recursive = TRUE)
```

---

sync_metadata *Fill in what the data deposit's metadata can know by itself*

Description

The deposit is described by four `.csv` files in `data/metadata/`, in the format of the dataspice package: `biblio.csv`, `creators.csv`, `attributes.csv` and `access.csv`. Three of their columns are already written down elsewhere in the project – the title and the keywords of the manuscript; the authors in its YAML, with their affiliations, the email of the corresponding author and their ORCID; the licence of the data, from the project's `LICENSE.txt`; and the variable names inside the data files – and this copies them across. Copying them by hand is how a deposit ends up disagreeing with its paper.

Usage

```
sync_metadata(quiet = FALSE, path = NULL)
```

Arguments

`quiet`	`TRUE` says nothing.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

It only ever ADDS. A cell you have filled is never touched and a variable you have described is never rewritten, so every render runs it without eating your work. What no machine can guess – units, descriptions, the temporal and geographic coverage – is yours to write, with `edit_metadata()`.

Value

What was filled in, as a character vector, invisibly.

See Also

`edit_metadata()` for the rest.

Examples

```
dir <- file.path(tempdir(), "my_paper")
create_paper(dir, title = "Ant colonies", authors = "Ada Lovelace",
             git = FALSE)
write.csv(data.frame(colony = 1:3, richness = c(4, 7, 5)),
          file.path(dir, "data", "colonies.csv"), row.names = FALSE)
sync_metadata(path = dir)
read.csv(file.path(dir, "data", "metadata", "attributes.csv"))
unlink(dir, recursive = TRUE)
```

---

trackdown *Edit a section with co-authors in Google Docs*

Description

Uploads one section of the manuscript to Google Drive, where co-authors comment and edit it in Google Docs, and brings their changes back. One section at a time: `manuscript.qmd` only holds the includes, so uploading it would show them no text.

Usage

```
td_upload(section, gpath = NULL, hide_code = TRUE, path = NULL)

td_update(section, gpath = NULL, hide_code = TRUE, path = NULL)

td_download(section, gpath = NULL, path = NULL)
```

Arguments

`section`	The section, by the start of its file name in `_sections/`: `"02"` or `"02_introduction"`.
`gpath`	The folder inside your Google Drive, e.g. `"papers/richness"`. `NULL`, the default, takes `trackdown-folder:` from the `easypaper:` block of `_quarto.yml`, or the root of the Drive.
`hide_code`	`TRUE` (the default) hides the code chunks in the Google Doc, so co-authors see prose.
`path`	The project, or any folder inside it. Left out, the one `set_paper()` named; with none named, this stops before writing: it writes into the project, and easypaper never takes the working directory for it.

Details

- `td_upload()` uploads a section for the first time.
- `td_update()` overwrites the Google Doc with your local version.
- `td_download()` brings the co-authors' edits back into the `.qmd`.

Always download before editing locally, and commit right after downloading: git then shows exactly what each co-author changed. If they would rather read the typeset paper, send them the `.docx` from `output/` and keep this for the sections they are actively editing.

Experimental. These need the development version of trackdown, which reads `.qmd` files: `remotes::install_github("Cla` The version on CRAN (1.1.1) does not, and trackdown has not been released since; while it is not, these may change, or go.

Value

Whatever trackdown returns, invisibly.

Examples

```
# It needs a Google account, authorised for trackdown, so it does not run
# here.
## Not run:
td_upload("02_introduction", path = "my_paper")
td_update("02_introduction", path = "my_paper")
td_download("02_introduction", path = "my_paper")

## End(Not run)
```

`write_packages_bib` *Write the references of the R packages a manuscript uses*

Description

Writes a `.bib` entry for every package, as `knitr::write_bib()` does – keyed `R-<package>`, so the text cites `[@R-vegan]` – with the names in their titles protected from the citation style. A journal style that sets titles in sentence case would otherwise print *Vegan: Community ecology package* and *easypaper: Automate reproducible quarto manuscripts*: the name of the package capitalised as the first word, and the names of other software lowercased. Three things are kept as written, between braces, the way BibTeX protects a word:

Usage

```
write_packages_bib(x = .packages(), file)
```

Arguments

`x`	The packages, by name. `.packages()`, the default, are those attached in the session that calls it.
`file`	The `.bib` file to write.

Details

- the name of the package at the head of its title – `{vegan}:`;
- the software its DESCRIPTION names in single quotes, as CRAN asks – `{Quarto}`, `{Excel}` – which knitr writes without the quotes;
- R, on its own.

The template calls it in the last chunk of `manuscript.qmd`, so the file is written again on every render, with the packages of that render.

Value

The path of `file`, invisibly.

Examples

```
f <- tempfile(fileext = ".bib")
write_packages_bib(c("base", "knitr"), f)
grep("title", readLines(f), value = TRUE)
unlink(f)
```

Index

add_journal, 3
add_journal(), *23, 24*
affiliations, 4
affiliations(), *14*

check_citations *(*checks*)*, 8
check_citations(), *32*
check_crossrefs *(*checks*)*, 8
check_data *(*checks*)*, 8
check_packages *(*checks*)*, 8
check_renv *(*checks*)*, 8
check_species, 5
check_species(), *7*, *9*
check_species_text, 6
check_title *(*checks*)*, 8
checks, *5*, 8, *29*, *32*
clean_cache, 9
convert_data, 10
convert_data(), *15*, *32*
create_example_paper, 12
create_example_paper(), *15*
create_paper, 13
create_paper(), *12, 13*, *23*, *33*

deposit_zenodo, 15
deposit_zenodo(), *26*

edit_metadata, 16
edit_metadata(), *34, 35*
export_code, 18
export_code(), *31*
export_figure_formats, 19

get_paper, 20
get_paper(), *33*

italicize_species, 20
italicize_species(), *5, 6*

knitr::write_bib(), *36*

list_journals, 23
list_journals(), *3*, *27*, *30*

make_preprint, 24
make_preprint(), *15*, *17*, *28*, *32*
make_submission, 26
make_submission(), *5*, *13*, *15*, *17, 18*, *24, 25*, *31, 32*
manuscript_stats, 28
manuscript_stats(), *32*

preview *(*render*)*, 29
preview(), *5*

render, 29
render_all *(*render*)*, 29
render_all(), *18*
render_docx *(*render*)*, 29
render_docx(), *9*, *13*, *24, 25*, *27, 28*
render_html *(*render*)*, 29
render_html(), *15*
render_pdf *(*render*)*, 29
render_supplementary *(*render*)*, 29

set_paper, 32
set_paper(), *3*, *5, 6*, *8–10*, *16–20*, *24, 25*, *27*, *29*, *31*, *33–35*
sync_licenses, 33
sync_metadata, 34
sync_metadata(), *17*, *34*

td_download *(*trackdown*)*, 35
td_update *(*trackdown*)*, 35
td_upload *(*trackdown*)*, 35
tools::R_user_dir(), *21*
trackdown, 35

write_packages_bib, 36