Package 'washr'

October 28, 2026

Title Publication Toolkit for Water, Sanitation and Hygiene (WASH) Data

Version 1.2.0

Date 2026-10-28

Description A toolkit to set up an R data package in a consistent structure. Automates tasks like tidy data export, data dictionary documentation, README and website creation, and citation management.

License GPL (>= 3)

URL https://openwashdata.github.io/washr/

BugReports https://github.com/openwashdata/washr/issues

Imports cffr (>= 1.0.1), cli (>= 3.6.0), desc (>= 1.4.3), jsonlite, pkgdown (>= 2.0.7), rlang (>= 1.1.3), tools, usethis (>= 2.2.3), utils, yaml, lifecycle

Suggests brand.yml, devtools (>= 2.4.5), knitr, remotes, rmarkdown, testthat (>= 3.2.1), withr (>= 3.0.0)

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Mian Zhong [aut] (ORCID: <https://orcid.org/0009-0009-4546-7214>), Margaux Götschmann [aut] (ORCID: <https://orcid.org/0009-0002-2567-3343>), Colin Walder [aut] (ORCID: <https://orcid.org/0009-0006-0969-1954>), Lars Schöbitz [aut, cre] (ORCID: <https://orcid.org/0000-0003-2196-5015>), Global Health Engineering, ETH Zurich [cph]

Maintainer Lars Schöbitz <lschoebitz@ethz.ch>

Repository CRAN

Date/Publication 2026-10-28 09:20:02 UTC

Contents

check_publication_readiness 2
setup_ci 4
setup_dictionary 5
setup_rawdata 6
setup_readme 6
setup_roxygen 7
setup_website 8
update_citation 9
update_description 11
update_dictionary 12
update_metadata 13
update_zenodo_json 14
use_brand 16

Index **18**

`check_publication_readiness` *Check whether the data package is ready for publication*

Description

`check_publication_readiness()` walks through the publication workflow and reports, item by item, what is in place and what is missing. For every gap it names the step and the washr function that closes it, so a skipped step shows up here and not as an error in a later function. The function only reads the package. It writes nothing.

The items are the mechanically decidable part of the openwashdata review standard for the metadata, the data dictionary, the documentation and the check workflow of a package. Each item has a stable `id`, so a script or a review tool can pick the rows it needs.

Area	id	Passes when
metadata	`license`	`License` in DESCRIPTION is CC BY 4.0, the license of the standard
metadata	`description_complete`	`Title`, `Description` and `Authors@R` no longer hold the placeholder text of a new
metadata	`citation_cff`	`CITATION.cff` is present
metadata	`citation_version`	the `version` in `CITATION.cff` equals `Version` in DESCRIPTION
metadata	`citation_authors`	`CITATION.cff` and `inst/CITATION` carry real authors, not "Firstname Lastname"
metadata	`zenodo_json`	`.zenodo.json` is present and equals what DESCRIPTION gives now
metadata	`keywords`	DESCRIPTION carries `X-schema.org-keywords`; the detail says whether `CITATI`
metadata	`coverage`	DESCRIPTION carries `X-schema.org-spatialCoverage` and `X-schema.org-te`
metadata	`title_length`	`Title` has at most 65 characters
data	`data_present`	`data/` holds at least one `.rda` file that loads

data	dictionary_present	data-raw/dictionary.csv is present
data	dictionary_coverage	the dictionary covers every variable of every data object
data	dictionary_descriptions	no description is empty or a placeholder
data	dictionary_schema	the five washr columns in order, UTF-8 without a byte order mark, one class name
docs	roxygen_docs	every data object has a help page in man/
docs	readme	README.Rmd and the rendered README.md are present
docs	rd_source	at least one help page in man/ carries a source
docs	readme_extdata_links	README.md links a .csv or .xlsx file under inst/extdata/
docs	readme_citation	README.md names the DOI that CITATION.cff carries
docs	vignettes_location	no .Rmd or .qmd file sits directly in vignettes/
docs	pkgdown_config	_pkgdown.yml is present
docs	pkgdown_analytics	_pkgdown.yml carries the analytics header
docs	pkgdown_url	url in _pkgdown.yml is the Pages URL, not the repository URL
docs	pkgdown_funding	_pkgdown.yml carries the funding text of the organisation
docs	pkgdown_brand	the brand entry in _pkgdown.yml matches the organisation
docs	pkgdown_llm_docs	_pkgdown.yml does not switch off the llms.txt of the site (llm-docs: false)
docs	docs_untracked	git tracks no file under docs/ while the pkgdown workflow deploys the site
tests	check_workflow	.github/workflows/R-CMD-check.yaml is present
tests	check_workflow_dev	every branches: list in that workflow includes dev
tests	check_badge	README.Rmd carries the R CMD check badge

An item that cannot be decided is reported as not applicable, with the reason: for example the version comparison when CITATION.cff is missing, or the docs/ item when the package has no pkgdown workflow. Not applicable items never count against the package.

Usage

```
check_publication_readiness(path = ".", profile = NULL)
```

Arguments

path	Path to the root of the data package. Defaults to the working directory.
profile	A named list with what the organisation expects of its packages. With NULL, the default, the package is held to its own values: the Config/washr/ fields in its DESCRIPTION that update_description() writes. A review tool passes the profile of the organisation instead. All entries are optional:

- license: the license the package has to carry, "CC BY 4.0" by default, or "any" to accept every license that is set. A package of another organisation that is held to its own values gets "any".
- analytics: "plausible" (the default) or "none". With "none" the analytics item is not applicable.
- site_url_pattern: the Pages URL of a package, with <package> in place of the package name. Without it, any github.io URL passes.
- funding_text: the funding sentence that _pkgdown.yml has to carry word for word. Without it, the funding item is not applicable.
- keywords_required: keywords every package of the organisation has to carry.

- `brand`: `"none"` when the organisation has no brand, or the name of its brand. Without it, the brand item is not applicable.

Value

A data frame of class `washr_readiness`, invisibly, with one row per item and the columns `id`, `area`, `check` (the item in words), `status` (`"pass"`, `"fail"` or `"na"` for not applicable), `detail` (what was found) and `fix` (the step that closes the gap). The attribute `ready` is `TRUE` when no item fails, `washr_version` holds the installed washr version, and `package` the name and version of the package that was checked. Printing the object shows the report again.

See Also

Before: every step of the workflow, up to `update_citation()` and `setup_website()`. Next: fix the items that fail and run it again, then release the package.

Other publishing functions: `setup_readme()`, `setup_website()`, `use_brand()`

Examples

```
## Not run:
check_publication_readiness()

# Keep the result and work with the items that fail
readiness <- check_publication_readiness()
readiness[readiness$status == "fail", c("id", "detail", "fix")]
attr(readiness, "ready")

## End(Not run)
```

---

setup_ci *Set up the R CMD check workflow for the data package*

Description

`setup_ci()` writes the GitHub Actions workflow that runs `R CMD check` on every push and pull request to `main` and `dev`, on macOS, Windows and three versions of R on Linux. The openwashdata review standard requires this workflow with the `dev` trigger, so a package scaffolded with washr passes that part of the review floor by construction.

The workflow lands in `.github/workflows/R-CMD-check.yaml` and `.github` is added to `.Rbuildignore`. When `README.Rmd` exists with badge markers, the matching R CMD check badge is added between them (the README template from `setup_readme()` carries it already). An existing workflow file is kept as it is, so the function is safe to run again.

Usage

```
setup_ci()
```

Value

The path of the workflow file, invisibly.

See Also

Next: `setup_rawdata()`. The badge appears in the README written by `setup_readme()`.

Other setup functions: `setup_dictionary()`, `setup_rawdata()`, `setup_roxygen()`, `update_dictionary()`

Examples

```
## Not run:
setup_ci()

## End(Not run)
```

---

setup_dictionary *Create a dictionary file for tidy data sets*

Description

`setup_dictionary()` generates a dictionary CSV file in the `data/` directory. The dictionary file contains information on the tidy data sets such as directory, file names, variable names, variable types, and descriptions. If tidy data exists, the dictionary is populated with relevant information; otherwise, it creates an empty dictionary CSV file.

Usage

```
setup_dictionary()
```

Value

The path of the dictionary file, invisibly. Error if raw data is not found or not in a package directory.

See Also

Before: `setup_rawdata()`. Next: `setup_roxygen()` once the descriptions in `data-raw/dictionary.csv` are written. When the data changes later, `update_dictionary()` brings the dictionary in line with it and keeps the descriptions.

Other setup functions: `setup_ci()`, `setup_rawdata()`, `setup_roxygen()`, `update_dictionary()`

Examples

```
## Not run:
setup_rawdata()
# Go to data_processing.R, clean the raw data and export tidy data
setup_dictionary()

## End(Not run)
```

setup_rawdata *Create the data-raw directory with a data-processing.R template*

Description

`setup_rawdata()` creates a directory for raw data and an example script named `data_processing.R` for importing, processing and exporting the tidy data. The template assumes that the dataset name is the same as the data package name.

Usage

```
setup_rawdata()
```

Value

The path of the processing script, invisibly. The function creates the directory "data-raw" under the package directory.

See Also

Before: `setup_ci()`. Next: `setup_dictionary()` once the processing script has exported the tidy data with `usethis::use_data()`.

Other setup functions: `setup_ci()`, `setup_dictionary()`, `setup_roxygen()`, `update_dictionary()`

Examples

```
## Not run:
  setup_rawdata()

## End(Not run)
```

setup_readme *Generate the README RMarkdown file*

Description

`setup_readme()` uses the openwashdata README template to generate README files based on datasets retrieved from the `data/` directory. It helps in creating consistent and informative README documentation for your data packages. The repository links, the install line and the license come from DESCRIPTION, so run `update_description()` first.

The Citation section states in plain text which version to cite with which DOI, links the authors page of the site and `CITATION.cff`, and names the source article of republished data from `X-schema.org-isBasedOn`. The download links point at the release tag of the version in DESCRIPTION, and at the main branch for a development version. pkgdown builds the `llms.txt` of the site from the README, so this is also what AI tools read about the package.

The template documents the first data object in `data/` (alphabetically); add a section per further object by hand. It stops when `data/` holds no data object, because every data section needs one.

Usage

```
setup_readme(force = FALSE, has_example = FALSE)
```

Arguments

`force`	Logical. If FALSE (the default), the function stops when a README.Rmd already exists. Set to TRUE to overwrite the existing file.
`has_example`	Logical. Should the README include an Example section with a commented ggplot2 scaffold for a first plot of the data? Defaults to FALSE. Pairs with the `has_example` argument of `setup_website()`, which adds the matching article to the site.

Value

The path of README.Rmd, invisibly.

See Also

Before: `update_description()`. Next: `setup_website()`, which builds the site from README.md.

Other publishing functions: `check_publication_readiness()`, `setup_website()`, `use_brand()`

Examples

```
## Not run:
# Generate the README file after setting up the dictionary
setup_dictionary()
# Complete and save the dictionary CSV file with variable descriptions
setup_readme()
# With an Example section to fill with a first plot
setup_readme(has_example = TRUE)

## End(Not run)
```

---

setup_roxygen *Set up roxygen documentation for all tidy data sets using the dictionary*

Description

Creates or updates Roxygen documentation for all tidy data sets found in the dictionary file.

When first run, this function creates the Roxygen documentation with placeholders for the title and description field. The dictionary should include columns for directory, file name, variable name, variable type, and description. This function generates Roxygen comments with this information, facilitating consistent and thorough documentation for your data sets.

When re-run this function, this function updates only the variable description entries in the Roxygen documentation files within R/ directory. The title and description fields remain unchanged.

Usage

```
setup_roxygen()
```

Value

The paths of the documentation files inside "R/", invisibly, one per data set. Error if tidy data cannot be found.

See Also

Before: `setup_dictionary()`, or `update_dictionary()` after the data changed. Next: `update_description()`.

Other setup functions: `setup_ci()`, `setup_dictionary()`, `setup_rawdata()`, `update_dictionary()`

Examples

```
## Not run:
setup_dictionary()
# Once the dictionary is created, go to data-raw/dictionary.csv and complete the column description.
setup_roxygen()

## End(Not run)
```

---

setup_website *Set up a pkgdown website for the data package*

Description

`setup_website()` writes the openwashdata pkgdown configuration and builds the site from the package documentation and README.md. The configuration comes from the washr template, filled from DESCRIPTION: the Pages URL as the site URL, the repository link, the analytics header, the funding sidebar, the authors footer, and a reference index with one entry per data object in `data/`. The analytics domain and the funding text are the `Config/washr/analytics-domain` and `Config/washr/funding` fields that `update_description()` writes; a field set to `none` leaves its block out.

The function is safe to re-run. An existing `_pkgdown.yml` is kept as it is and only the site is rebuilt, so hand edits and the brand wiring from `use_brand()` survive. The example article is created once.

By default the built `docs/` folder is tracked in git, which is how openwashdata packages publish their site from `main`. When the package deploys through the pkgdown GitHub Actions workflow instead (`.github/workflows/pkgdown.yaml` exists), `docs/` stays ignored.

Usage

```
setup_website(has_example = FALSE, track_docs = NULL)
```

Arguments

has_example	Logical. Should the pkgdown website include a vignette page for writing an example? Defaults to FALSE.
track_docs	Logical. Remove docs from .gitignore so the built site is committed? Defaults to TRUE unless a pkgdown workflow exists under .github/workflows/.

Value

The path of _pkgdown.yml, invisibly. Error if no README file is found.

See Also

Before: setup_readme(). Next: use_brand() for the brand, and update_citation() once the release has a DOI.

Other publishing functions: check_publication_readiness(), setup_readme(), use_brand()

Examples

```
## Not run:
# Set up the pkgdown website including a vignette page
 setup_website(has_example = TRUE)

## End(Not run)
```

---

update_citation *Update the citation file for the dataset.*

Description

Create a citation *.cff file for the dataset from a given DOI (Digital Object Identifier). When a DOI is supplied, it adds the DOI badge to the README RMarkdown file (the Zenodo badge, or a generic DOI badge when Config/washr/doi-provider in DESCRIPTION names another provider) and re-builds the README.md and pkgdown website if they exist. Before a release exists, call it without arguments to generate the citation files without a DOI or badge.

The same run writes .zenodo.json through update_zenodo_json(), so the metadata Zenodo reads at release time comes from the same DESCRIPTION as the citation files.

Usage

```
update_citation(doi = NULL, build = TRUE, type = c("dataset", "software"))
```

Arguments

doi	DOI (Digital Object Identifier), e.g., 10.5281/zenodo.11185699. Defaults to NULL for the call before the release, in which case no DOI is recorded and no badge is added.
build	Logical. Rebuild README.md and the pkgdown site after the citation files change? Defaults to TRUE. Set to FALSE to regenerate the citation files alone, e.g., in scripts and tests.
type	The CFF type of the work: "dataset" (the default, a data package) or "software". Before 1.1.1 the file always said software, the cffr default. Zenodo's GitHub integration ignores this field; the resource type of a deposit comes from .zenodo.json, which always says dataset.

Details

When the data comes from a published article, list the article DOI in DESCRIPTION as X-schema.org-isBasedOn, e.g., X-schema.org-isBasedOn: https://doi.org/10.2166/wh.2026.173. Separate several DOIs with commas. update_citation() looks up each DOI at doi.org and writes it as a references entry in CITATION.cff, with a message that asks users to cite both the data package and the article. inst/CITATION then holds both entries, so citation() prints both. The package itself stays the work cited by GitHub's "Cite this repository". A DOI that cannot be looked up keeps its entry from the existing CITATION.cff. Other references in CITATION.cff or inst/CITATION are dropped, because DESCRIPTION is their canonical source (#134).

Value

The paths of the three files written, CITATION.cff, inst/CITATION and .zenodo.json, invisibly.

See Also

Before: setup_website(). Run again with the DOI after the Zenodo release; update_metadata() then picks the DOI up. update_zenodo_json() for the Zenodo metadata file alone.

Other metadata functions: update_description(), update_metadata(), update_zenodo_json()

Examples

```
## Not run:
  update_citation(doi = "10.5281/zenodo.11185699")
  # Regenerate the citation files without rebuilding README.md and the site
  update_citation(build = FALSE)

## End(Not run)
```

update_description *Update the DESCRIPTION file to conform with openwashdata standards*

Description

This function updates the DESCRIPTION file of an R package to comply with openwashdata standards. It ensures that fields such as `License`, `Language`, `Date`, `URL`, and others are correctly specified. Existing `URL` and `Config/Needs/website` entries are preserved and merged with the openwashdata defaults. A CC BY 4.0 license is only set when the package does not have a license yet; an existing license is left untouched.

Usage

```
update_description(file = ".", github_user = NULL)
```

Arguments

`file`	Character. The file path to the DESCRIPTION file of the R package. Defaults to the current working directory.
`github_user`	Character. The URL of the GitHub user or organization that hosts the package, e.g., `"https://github.com/yourorg"`. Defaults to the organisation of the repository already listed in `URL`, and to `"https://github.com/openwashdata"` when `URL` lists none.

Details

DESCRIPTION is where the other functions read the facts of the package: the GitHub organisation and the repository from `URL`, the license from `License`, the people from `Authors@R`. The values that DESCRIPTION has no standard field for are written as `Config/washr/` fields, and every other function reads them from there:

Field	Used for	openwashdata default
`Config/washr/funding`	the funding sentence in the sidebar of the site	the ETH Board Open Research Dat
`Config/washr/analytics-domain`	the Plausible analytics header of the site	`openwashdata.github.io`
`Config/washr/doi-provider`	the DOI badge of the README	`zenodo`
`Config/washr/zenodo-community`	the community in `.zenodo.json`	`openwashdata`
`Config/washr/brand-source`	the GitHub repository `use_brand()` copies from	`openwashdata/brand`
`Config/washr/version`	the washr version that last ran this function	

A field that you edited is never overwritten, so a group other than openwashdata publishes with its own values by editing them. When `github_user` names another organisation than the one in `URL`, the repository in `URL` is replaced and the fields that still hold the defaults of the old organisation follow. Set a field to `none` to switch its feature off. For a package under another GitHub organisation the funding, analytics, community and brand fields are written as `none`, because the openwashdata values would be wrong there. One more field is only ever written by hand: `Config/washr/pages-domain`, for a site that is not served from `<organisation>.github.io`.

Value

The path of the DESCRIPTION file, invisibly. The fields are updated in the file.

See Also

Before: `setup_roxygen()`. Next: `update_metadata()` for the schema.org metadata, then `setup_readme()`.

Other metadata functions: `update_citation()`, `update_metadata()`, `update_zenodo_json()`

Examples

```
## Not run:
 # Update DESCRIPTION file in the current package
update_description()

 # Update DESCRIPTION file in a specific package
update_description(file = "path/to/your/package/DESCRIPTION")

 # Update DESCRIPTION file with a specific GitHub user
update_description(github_user = "https://github.com/yourusername")

## End(Not run)
```

---

update_dictionary *Bring the dictionary in line with the tidy data*

Description

`update_dictionary()` reads the data objects in `data/` again and updates `data-raw/dictionary.csv` to match them. Run it after the data changed: a variable was added, removed or renamed, its type changed, or a data object was added or removed.

What you wrote in the dictionary is kept. Rows are matched on `file_name` and `variable_name`, and a matched row keeps its description and the value of every other column, including columns you added yourself, such as `unit` or `allowed_values`. Only `variable_type` is refreshed from the data. A new variable gets a row with empty cells to fill in. The row of a variable that no longer exists is removed, and the function names it. A renamed variable or data object cannot be told from one removed and one added, so what you wrote in the removed rows is printed for you to copy into the new rows.

The columns keep the order they have in the file. The rows follow the data: data objects in alphabetical order, variables in the order of the data object. When nothing differs, the file is left untouched, so the function is safe to run again.

Usage

```
update_dictionary()
```

Value

The path of the dictionary file, invisibly. Error if there is no dictionary yet, no tidy data, or the dictionary lacks the `file_name` or `variable_name` column.

See Also

Before: `setup_dictionary()`, which writes the dictionary the first time. Next: `setup_roxygen()`, which regenerates the variable table of the documentation from the dictionary.

Other setup functions: `setup_ci()`, `setup_dictionary()`, `setup_rawdata()`, `setup_roxygen()`

Examples

```
## Not run:
setup_dictionary()
# Later, after data_processing.R exported changed data
update_dictionary()
setup_roxygen()

## End(Not run)
```

---

update_metadata *Generate and embed the schema.org metadata of the data package*

Description

[Experimental]

`update_metadata()` derives a schema.org Dataset description from the canonical sources of the package and writes it as a JSON-LD block into the head of every pkgdown page, where dataset search engines read it. Nothing is hand edited: to change a value, change its source and run the function again. Running it twice produces no change. It ends by listing the fields it could not fill and where to fill them.

The sources are:

Field	Source
name, description, version, datePublished, license	`Title`, `Description`, `Version`, `Date`, `License` in DESCRIPTION
url	the pkgdown site (a `github.io` entry in `URL`), else the repository
keywords	`X-schema.org-keywords` in DESCRIPTION, comma separated
spatialCoverage, temporalCoverage	`X-schema.org-spatialCoverage` and `X-schema.org-temporalCovera`
isBasedOn	the source article DOIs in `X-schema.org-isBasedOn` in DESCRIPTION
creator, maintainer, funder, publisher	`Authors@R` roles `aut`/`cre`, `cre`, `fnd`, `cph`; ORCID from the `comment` field
identifier, sameAs	the DOI in `CITATION.cff`, written by `update_citation()`
variableMeasured	`data-raw/dictionary.csv`
distribution	every file in `inst/extdata` that belongs to a dataset, one entry per file

The JSON-LD lands in `pkgdown/templates/in-header.html`, which pkgdown picks up on the next site build. The file also keeps the `in_header` includes from `_pkgdown.yml` working. It is not shipped in the package tarball.

Usage

```
update_metadata(quiet = FALSE)
```

Arguments

`quiet`	Logical. Suppress the messages and the report of blank fields. Defaults to `FALSE`, or to `TRUE` when `options(washr.quiet = TRUE)` is set.

Value

The Dataset description as a list, invisibly. The `"blank"` attribute names the fields that could not be filled and says where to fill them.

See Also

Before: `update_description()`, which writes the DESCRIPTION fields this reads. Next: `setup_readme()`. The DOI comes from `update_citation()`.

Other metadata functions: `update_citation()`, `update_description()`, `update_zenodo_json()`

Examples

```
## Not run:
update_metadata()

## End(Not run)
```

---

update_zenodo_json *Write the Zenodo metadata file of the data package*

Description

`update_zenodo_json()` writes `.zenodo.json` at the package root. Zenodo reads this file when a GitHub release is archived and takes the metadata of the record from it: the resource type (Dataset), the community, the title, the description, the creators with their ORCID iDs, the license, the keywords and the version. Without the file, Zenodo files a data package as Software in no community, because its GitHub integration reads only a few fields of `CITATION.cff`.

Nothing in the file is hand edited: to change a value, change its source and run the function again. Running it twice produces no change. `update_citation()` calls it, so the citation files and `.zenodo.json` are written in one step and stay in line with each other.

The sources are:

Key	Source

title, description, version	`Title`, `Description`, `Version` in DESCRIPTION
upload_type	always `dataset`
creators	`Authors@R` roles `aut` and `cre`; ORCID and affiliation from the `comment` field
license	`License` in DESCRIPTION, as the identifier Zenodo uses (e.g., `cc-by-4.0`)
keywords	`X-schema.org-keywords` in DESCRIPTION, comma separated
communities	`Config/washr/zenodo-community` in DESCRIPTION: `openwashdata` for an openwashdata pack
related_identifiers	the source article DOIs in `X-schema.org-isBasedOn` in DESCRIPTION

Usage

```
update_zenodo_json()
```

Details

Zenodo uses `.zenodo.json` in place of `CITATION.cff` for the record, and it reads the file from the release tag. Commit the file before you create the release, and run the function again whenever DESCRIPTION changes.

Set `Config/washr/zenodo-community` in DESCRIPTION to file the record in another community, to several (comma separated), or to `none` for no community.

A source article is written as a related identifier with the relation `isDerivedFrom`. Zenodo then drops its own link to the release tag on GitHub, so the repository from the `URL` field is added as a second related identifier.

When washr does not know the license, the file carries none. Zenodo then takes the license GitHub detects for the repository, and CC0 1.0 for a dataset when GitHub detects none.

Value

The path of `.zenodo.json`, invisibly.

See Also

Before: `update_description()`, which writes the DESCRIPTION fields this reads. `update_citation()` calls this function. Next: the GitHub release that Zenodo archives.

Other metadata functions: `update_citation()`, `update_description()`, `update_metadata()`

Examples

```
## Not run:
update_zenodo_json()

## End(Not run)
```

use_brand *Install or refresh the openwashdata brand in the active package*

Description

use_brand() copies the openwashdata brand definition (_brand.yml) and the logo files it references from the central openwashdata/brand repository into the package root. Re-running the function refreshes an existing copy and reports which files changed, so consuming packages stay in sync with the central definition.

Brand values are never edited locally: change them in openwashdata/brand first and cut a release tag there, then refresh consumers with use_brand(). After a brand release the order is the Quarto extension, the website, then the data packages.

By default the brand is copied at the latest release tag of the brand repository, and the tag is recorded in DESCRIPTION as Config/washr/brand, so the package says which brand it carries. A second run with no new tag changes no file. The brand repository is the Config/washr/brand-source field that update_description() writes (openwashdata/brand for openwashdata packages); a group with its own brand names its repository there, and none means there is no brand to install.

The brand file and the logo directory are added to .Rbuildignore, so they stay out of the built package and R CMD check does not report them as non-standard top-level files.

Usage

```
use_brand(ref = NULL, pkgdown = TRUE, source = NULL)
```

Arguments

ref	Character. Git reference (branch or tag) of the brand repository to copy from. Defaults to the latest release tag. Pass "main" to try brand changes that are not released yet.
pkgdown	Logical. Should _pkgdown.yml be wired to use the brand via bslib? Defaults to TRUE.
source	Character. Advanced: an alternative source for the brand files, either a local directory or a URL prefix. When NULL (the default), the raw GitHub content of the brand repository at ref is used. Mainly useful for tests and offline work.

Details

With pkgdown = TRUE (the default), an existing _pkgdown.yml is pointed at the brand through bslib (template.bslib.brand), so the next pkgdown::build_site() renders the site with the brand fonts and colors. The wiring adds its lines to _pkgdown.yml and leaves the rest of the file, comments included, as it is. Only when the template block already carries other bslib settings, or is written on one line, is the file rewritten through the yaml package, which does not preserve comments. When no _pkgdown.yml exists, the wiring is skipped with a hint to run setup_website() first. Building the wired site requires the brand.yml package (bslib asks for it at build time); it is listed in Suggests and installed on demand.

Value

Invisibly, a character vector of the files written or updated (empty when everything was already current).

See Also

Before: `setup_website()`, which writes the `_pkgdown.yml` this wires. For branded PDF and Word reports that read the installed `_brand.yml`, the openwashdata Quarto extension quarto-owd.

Other publishing functions: `check_publication_readiness()`, `setup_readme()`, `setup_website()`

Examples

```
## Not run:
# Install the brand and wire the pkgdown site
use_brand()

# Refresh later, without touching _pkgdown.yml
use_brand(pkgdown = FALSE)

## End(Not run)
```

Index

∗ **metadata functions**
  update_citation, 9
  update_description, 11
  update_metadata, 13
  update_zenodo_json, 14
∗ **publishing functions**
  check_publication_readiness, 2
  setup_readme, 6
  setup_website, 8
  use_brand, 16
∗ **setup functions**
  setup_ci, 4
  setup_dictionary, 5
  setup_rawdata, 6
  setup_roxygen, 7
  update_dictionary, 12

check_publication_readiness, 2
check_publication_readiness(), *7*, *9*, *17*

pkgdown::build_site(), *16*

setup_ci, 4
setup_ci(), *5, 6*, *8*, *13*
setup_dictionary, 5
setup_dictionary(), *5, 6*, *8*, *13*
setup_rawdata, 6
setup_rawdata(), *5*, *8*, *13*
setup_readme, 6
setup_readme(), *4, 5*, *9*, *12*, *14*, *17*
setup_roxygen, 7
setup_roxygen(), *5, 6*, *12, 13*
setup_website, 8
setup_website(), *4*, *7*, *10*, *16, 17*

update_citation, 9
update_citation(), *4*, *9*, *12–15*
update_description, 11
update_description(), *3*, *6–8*, *10*, *14–16*
update_dictionary, 12
update_dictionary(), *5, 6*, *8*
update_metadata, 13
update_metadata(), *10*, *12*, *15*
update_zenodo_json, 14
update_zenodo_json(), *9, 10*, *12*, *14*
use_brand, 16
use_brand(), *4*, *7–9*, *11*