Package: cloudfs (via r-universe)

September 13, 2024

Title Streamlined Interface to Interact with Cloud Storage Platforms

Version 0.1.3

Description A unified interface for simplifying cloud storage interactions, including uploading, downloading, reading, and writing files, with functions for both 'Google Drive' (<https://www.google.com/drive/>) and 'Amazon S3' (<https://aws.amazon.com/s3/>).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports aws.s3, googledrive, desc, dplyr, cli, utils, rlang, glue, httr

Suggests googlesheets4, haven, jsonlite, knitr, readr, readxl, rmarkdown, testthat (>= 3.0.0), withr, writexl, xml2

VignetteBuilder knitr

Config/testthat/edition 3

URL https://g6t.github.io/cloudfs/, https://github.com/g6t/cloudfs

BugReports https://github.com/g6t/cloudfs/issues

Repository https://g6t.r-universe.dev

RemoteUrl https://github.com/g6t/cloudfs

RemoteRef HEAD

RemoteSha 811ae7ceb79b1a46f467eae81be24d07859ebe99

Contents

`cloud_drive_attach` *Attach Google Drive folder to project*

Description

This function facilitates the association of a specific Google Drive folder with a project by adding a unique identifier to the project's DESCRIPTION file. The user is prompted to navigate to the Google Drive website, select or create the desired folder for the project, and then provide its URL. The function extracts the necessary information from the URL and updates the `cloudfs.drive` field in the DESCRIPTION file accordingly.

Usage

```
cloud_drive_attach(project = ".")
```

Arguments

`project`	Character. Path to a project. By default it is current working directory.

Value

This function does not return a meaningful value. Its primary purpose is the side effect of updating the project's DESCRIPTION file with the associated Google Drive folder identifier.

Examples

```
cloud_drive_attach()
```

`cloud_drive_browse` *Browse project's Google Drive folder*

Description

Opens project's Google Drive folder in browser.

Usage

```
cloud_drive_browse(path = "", root = NULL)
```

Arguments

`path`	(optional) Path inside the Google Drive folder to open. Defaults to the root level (path = "") of the project's folder.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Details

Google Drive file structure is different from the usual file structure like e.g. on Linux or Windows. A folder on Google Drive can have two or more child folders with the same name. Google Drive marks files and folders with so-called id values to distinguish between them. These values are always unique. You can see them in browser URL for example. The concept of "name" is in the first place for convenience of the end user.

In such a setup a relative file path may correspond to multiple files or folders. This function however works under assumption that the relative path you pass to it defines strictly one object. If there's any ambiguity it throws an error.

Value

Invisibly returns `NULL`. The primary purpose of this function is its side effect: opening the specified Google Drive folder in a browser.

Examples

```
cloud_drive_browse()
cloud_drive_browse("models/kmeans")
```

`cloud_drive_download` *Download a file from Google Drive to the local project folder*

Description

Retrieves a file from the project's Google Drive folder and saves it to the local project folder, maintaining the original folder structure.

Usage

```
cloud_drive_download(file, root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Details

Google Drive file structure is different from the usual file structure like e.g. on Linux or Windows. A folder on Google Drive can have two or more child folders with the same name. Google Drive marks files and folders with so-called id values to distinguish between them. These values are always unique. You can see them in browser URL for example. The concept of "name" is in the first place for convenience of the end user.

In such a setup a relative file path may correspond to multiple files or folders. This function however works under assumption that the relative path you pass to it defines strictly one object. If there's any ambiguity it throws an error.

Value

Invisibly returns `NULL` after successfully downloading the file.

Examples

```
# downloads toy_data/demo.csv from project's Google Drive folder
# (provided it exists) and saves it to local 'toy_data' folder
cloud_drive_download("toy_data/demo.csv")

# clean up
unlink("toy_data", recursive = TRUE)
```

`cloud_drive_download_bulk` *Bulk download contents from Google Drive*

Description

Downloads multiple files from a Google Drive folder based on the output dataframe from cloud_drive_ls. This function streamlines the process of downloading multiple files by allowing you to filter and select specific files from the Google Drive listing and then download them in bulk.

Usage

```
cloud_drive_download_bulk(content, quiet = FALSE)
```

Arguments

`content`	(data.frame) Output of `cloud_drive_ls()`
`quiet`	All caution messages may be turned off by setting this parameter to `TRUE`.

Value

Invisibly returns the input `content` dataframe.

Examples

```
# provided there's a folder called "toy_data" in the root of your project's
# Google Drive folder, and this folder contains "csv" files
cloud_drive_ls("toy_data") |>
  filter(type == "csv") |>
  cloud_drive_download_bulk()

# clean up
unlink("toy_data", recursive = TRUE)
```

`cloud_drive_ls` *List Contents of Project's Google Drive Folder*

Description

Returns a tibble with names, timestamps, and sizes of files and folders inside the specified Google Drive folder.

Usage

```
cloud_drive_ls(path = "", recursive = FALSE, full_names = FALSE, root = NULL)
```

Arguments

`path`	(optional) Path inside the Google Drive root folder. Specifies the subfolder whose contents should be listed. By default, when `path = ""`, lists root-level files and folders.
`recursive`	(logical) If `TRUE`, lists contents recursively in all nested subfolders. Default is `FALSE`.
`full_names`	(logical) If `TRUE`, folder path is appended to object names to give a relative file path.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Details

Google Drive file structure is different from the usual file structure like e.g. on Linux or Windows. A folder on Google Drive can have two or more child folders with the same name. Google Drive marks files and folders with so-called id values to distinguish between them. These values are always unique. You can see them in browser URL for example. The concept of "name" is in the first place for convenience of the end user.

In such a setup a relative file path may correspond to multiple files or folders. This function however works under assumption that the relative path you pass to it defines strictly one object. If there's any ambiguity it throws an error.

Value

A tibble containing the names, last modification timestamps, sizes in bytes, and Google Drive IDs of files and folders inside the specified Google Drive folder.

Examples

```
# list only root-level files and folders
cloud_drive_ls()

# list all files in all nested folders
cloud_drive_ls(recursive = TRUE)

# list contents of "plots/barplots" subfolder
cloud_drive_ls("plots/barplots")
```

`cloud_drive_read` *Read a file from Google Drive*

Description

Retrieves and reads a file from the project's Google Drive folder. By default, the function attempts to determine the appropriate reading function based on the file's extension. However, you can specify a custom reading function if necessary.

Usage

```
cloud_drive_read(file, fun = NULL, ..., root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`fun`	A custom reading function. If `NULL` (default), the appropriate reading function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the reading function `fun`.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Value

The content of the file read from Google Drive, with additional attributes containing metadata about the file.

Default reading functions

Here's how we identify a reading function based on file extension

- `.csv`: readr::read_csv
- `.json`: jsonlite::read_json
- `.rds`: base::readRDS
- `.sav`: haven::read_sav
- `.xls`: cloud_read_excel
- `.xlsx`: cloud_read_excel
- `.xml`: xml2::read_xml

Examples

```
# provided there are folders called "data" and "models" in the root of your
# project's main Google Drive folder and they contain the files mentioned
# below
cloud_drive_read("data/mtcars.csv")
cloud_drive_read("models/random_forest.rds")
cloud_drive_read("data/dm.sas7bdat", fun = haven::read_sas)
```

`cloud_drive_read_bulk` *Bulk Read Contents from Google Drive*

Description

This function facilitates the bulk reading of multiple files from the project's designated Google Drive folder. By using cloud_drive_ls, you can obtain a dataframe detailing the contents of the Google Drive folder. Applying `cloud_drive_read_bulk` to this dataframe allows you to read all listed files into a named list. The function will, by default, infer the appropriate reading method based on each file's extension. However, if a specific reading function is provided via the `fun` parameter, it will be applied uniformly to all files, which may not be suitable for diverse file types.

Usage

```
cloud_drive_read_bulk(content, fun = NULL, ..., quiet = FALSE)
```

Arguments

`content`	(data.frame) Output of `cloud_drive_ls()`
`fun`	A custom reading function. If `NULL` (default), the appropriate reading function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the reading function `fun`.
`quiet`	All caution messages may be turned off by setting this parameter to `TRUE`.

Value

A named list where each element corresponds to the content of a file from Google Drive. The names of the list elements are derived from the file names.

Examples

```
# provided there's a folder called "data" in the root of the project's main
# Google Drive folder, and it contains csv files
data_lst <-
  cloud_drive_ls("data") |>
  filter(type == "csv") |>
  cloud_drive_read_bulk()
```

`cloud_drive_spreadsheet_autofit` *Automatically resize all columns in a google spreadsheet*

Description

Finds the spreadsheet by path relative to a project root. Applies `googlesheets4::range_autofit()` to each sheet.

Usage

```
cloud_drive_spreadsheet_autofit(file, root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Value

The file ID of the resized Google spreadsheet as an invisible result.

Examples

```
cloud_drive_write(mtcars, "results/mtcars.xlsx")
cloud_drive_spreadsheet_autofit("results/mtcars.xlsx")
```

`cloud_drive_upload` *Upload a local file to Google Drive*

Description

Uploads a local file from the project's directory to its corresponding location within the project's Google Drive root folder.

Usage

```
cloud_drive_upload(file, root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Details

Google Drive file structure is different from the usual file structure like e.g. on Linux or Windows. A folder on Google Drive can have two or more child folders with the same name. Google Drive marks files and folders with so-called id values to distinguish between them. These values are always unique. You can see them in browser URL for example. The concept of "name" is in the first place for convenience of the end user.

In such a setup a relative file path may correspond to multiple files or folders. This function however works under assumption that the relative path you pass to it defines strictly one object. If there's any ambiguity it throws an error.

Value

Invisibly returns a googledrive::dribble object representing the uploaded file on Google Drive.

Examples

```
# create a toy csv file
dir.create("toy_data")
write.csv(mtcars, "toy_data/mtcars.csv")

# uploads toy_data/mtcars.csv to 'data' subfolder of project's
# Google Drive folder
cloud_drive_upload("toy_data/mtcars.csv")

# clean up
unlink("toy_data", recursive = TRUE)
```

`cloud_drive_upload_bulk`

Bulk Upload Files to Google Drive

Description

This function streamlines the process of uploading multiple files from the local project folder to the project's designated Google Drive folder. By using cloud_local_ls, you can obtain a dataframe detailing the contents of the local folder. Applying `cloud_drive_upload_bulk` to this dataframe allows you to upload all listed files to Google Drive.

Usage

```
cloud_drive_upload_bulk(content, quiet = FALSE, root = NULL)
```

Arguments

`content`	(data.frame) Output of `cloud_s3_ls()`
`quiet`	All caution messages may be turned off by setting this parameter to `TRUE`.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Value

Invisibly returns the input `content` dataframe.

Examples

```
# create toy plots: 2 png's and 1 jpeg
dir.create("toy_plots")
png("toy_plots/plot1.png"); plot(rnorm(100)); dev.off()
png("toy_plots/plot2.png"); plot(hist(rnorm(100))); dev.off()
png("toy_plots/plot3.jpeg"); plot(hclust(dist(USArrests), "ave")); dev.off()

# upload only the two png's
cloud_local_ls("toy_plots")  |>
  dplyr::filter(type == "png")  |>
  cloud_drive_upload_bulk()

# clean up
unlink("toy_plots", recursive = TRUE)
```

`cloud_drive_write` *Write an object to Google Drive*

Description

Saves an R object to a designated location in the project's Google Drive folder. If no custom writing function is provided, the function will infer the appropriate writing method based on the file's extension.

Usage

```
cloud_drive_write(x, file, fun = NULL, ..., local = FALSE, root = NULL)
```

Arguments

`x`	An R object to be written to Google Drive.
`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`fun`	A custom writing function. If `NULL` (default), the appropriate writing function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the writing function `fun`.
`local`	Logical, defaulting to `FALSE`. If `TRUE`, the function will also create a local copy of the file at the specified path. Note that some writing functions might not overwrite existing files unless explicitly allowed. Typically, such functions have a parameter (often named `overwrite`) to control this behavior. Check the documentation of the writing function used to determine the exact parameter name and pass it through the `...` argument if necessary. Alternatively, you can define an anonymous function for `fun` that calls a writing function with the overwriting option enabled.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Value

Invisibly returns a googledrive::dribble object representing the written file on Google Drive.

Default writing functions

Here's how we identify a writing function based on file extension

- `.csv`: readr::write_csv
- `.json`: jsonlite::write_json
- `.rds`: base::saveRDS
- `.xls`: writexl::write_xlsx
- `.xlsx`: writexl::write_xlsx
- `.sav`: haven::write_sav
- `.xml`: xml2::write_xml

Examples

```
# write mtcars dataframe to mtcars.csv in data folder
cloud_drive_write(mtcars, "data/mtcars.csv")
cloud_drive_write(random_forest, "models/random_forest.rds")

# provide custom writing function with parameters
cloud_drive_write(c("one", "two"), "text/count.txt", writeLines, sep = "\n\n")
```

cloud_drive_write_bulk

Write multiple objects to Google Drive in bulk

Description

This function allows for the bulk writing of multiple R objects to the project's designated Google Drive folder. To prepare a list of objects for writing, use cloud_object_ls, which generates a dataframe listing the objects and their intended destinations in a format akin to the output of cloud_drive_ls. By default, the function determines the appropriate writing method based on each file's extension. However, if a specific writing function is provided via the `fun` parameter, it will be applied to all files, which may not be ideal if dealing with a variety of file types.

Usage

```
cloud_drive_write_bulk(
  content,
  fun = NULL,
  ...,
  local = FALSE,
  quiet = FALSE,
  root = NULL
)
```

Arguments

`content`	(data.frame) output of `cloud_object_ls()`
`fun`	A custom writing function. If `NULL` (default), the appropriate writing function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the writing function `fun`.
`local`	Logical, defaulting to `FALSE`. If `TRUE`, the function will also create a local copy of the file at the specified path. Note that some writing functions might not overwrite existing files unless explicitly allowed. Typically, such functions have a parameter (often named `overwrite`) to control this behavior. Check the documentation of the writing function used to determine the exact parameter name and pass it through the `...` argument if necessary. Alternatively, you can define an anonymous function for `fun` that calls a writing function with the overwriting option enabled.
`quiet`	all caution messages may be turned off by setting this parameter to `TRUE`.
`root`	Google Drive ID or URL of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.drive` field of the project's DESCRIPTION file.

Value

Invisibly returns the input `content` dataframe.

Examples

```
# write two csv files: data/df_mtcars.csv and data/df_iris.csv
cloud_object_ls(
  dplyr::lst(mtcars = mtcars, iris = iris),
  path = "data",
  extension = "csv",
  prefix = "df_"
) |>
cloud_drive_write_bulk()
```

`cloud_get_roots` *Get cloud roots of a project*

Description

Returns a list with all `cloudfs.*` roots defined in a project's DESCRIPTION.

Usage

```
cloud_get_roots(project = ".")
```

Arguments

`project`	Character. Path to a project. By default it is current working directory.

Value

A named list where each element corresponds to a `cloudfs.*` root defined in the project's DESCRIPTION file. The names of the list elements are derived from the `cloudfs.*` fields by removing the `cloudfs.` prefix.

Examples

```
# create a temp. folder, and put DESCRIPTION file with cloudfs.* fields into it
tmp_project <- file.path(tempdir(), "cloudfs")
if (!dir.exists(tmp_project)) dir.create(tmp_project)
tmp_project_desc <- file.path(tmp_project, "DESCRIPTION")
desc_content <- c(
  "Package: -",
  "cloudfs.s3: my_bucket/my_project",
  "cloudfs.drive: aaaaaa"
)
writeLines(desc_content, tmp_project_desc)

roots <- cloud_get_roots(tmp_project)
roots
```

`cloud_local_ls` *List Contents of local project folder*

Description

Retrieves names, timestamps, and sizes of files and folders inside local project folder.

Usage

```
cloud_local_ls(
  path = "",
  root = ".",
  recursive = FALSE,
  full_names = FALSE,
  ignore = TRUE
)
```

Arguments

`path`	(optional) Path, relative to the specified root to list contents of. By default, when `path = ""`, lists root-level files and folders.
`root`	Local directory path relative to which all other paths are considered.
`recursive`	(logical) If `TRUE`, lists contents recursively in all nested subfolders. Default is `FALSE`.
`full_names`	(logical) If `TRUE`, folder path is appended to object names to give a relative file path.
`ignore`	Logical flag indicating whether to ignore certain directories. Currently, if set to `TRUE`, the 'renv' folder is ignored due to its typically large size. This parameter may be expanded in the future to support more complex ignore patterns.

Value

A tibble containing the names, last modification timestamps, and sizes in bytes of files and folders inside the specified local folder.

Examples

```
# list only root-level files and folders
cloud_local_ls()

# list all files in all nested folders
cloud_local_ls(recursive = TRUE)

## Not run:
# list contents of "plots/barplots" subfolder (if it exists)
cloud_local_ls("plots/barplots")
```

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

cloud_object_ls *Prepare a dataframe for bulk writing of objects to cloud*

Description

`cloud_*_ls` functions for cloud locations (e.g. `cloud_s3_ls`) return content dataframes which can then be passed to `cloud_*_read_bulk` and `cloud_*_download_bulk` functions to read/download multiple files at once. In a similar manner, this function accepts a list of objects as an input and produces a dataframe which can then be passed to `cloud_*_write_bulk` functions to write multiple files at once.

Usage

```
cloud_object_ls(x, path, extension, prefix = "", suffix = "")
```

Arguments

`x`	A **named** list. Names may contain letters, digits, spaces, '.', '-', '_' symbols and cannot contain trailing or leading spaces.
`path`	A directory relative to the project root to write objects to.
`extension`	File extension (string) without the leading dot.
`prefix, suffix`	(optional) strings to attach at the beginning or at the end of file names.

Value

A tibble in which each row represents an object from the input list, comprising the following columns:

- `object` - objects you've provided
- `name` - contains paths where objects are meant to be written.

Examples

```
cloud_object_ls(
  dplyr::lst(mtcars = mtcars, iris = iris),
  path = "data",
  extension = "csv",
  prefix = "df_"
)
```

`cloud_read_excel` *Read excel file as a list of dataframes*

Description

Uses readxl::read_excel under the hood, reads all sheets and returns them as a named list of dataframes.

Usage

```
cloud_read_excel(path)
```

Arguments

`path`	Path to the xlsx/xls file.

Value

A named list of dataframes, where each dataframe corresponds to a sheet in the Excel file. The names of the list elements are derived from the sheet names.

Examples

```
datasets <- readxl::readxl_example("datasets.xlsx")
cloud_read_excel(datasets)
```

`cloud_s3_attach` *Attach S3 folder to project*

Description

This function facilitates the association of a specific S3 folder with a project by adding a unique identifier to the project's DESCRIPTION file. The user is prompted to navigate to the S3 console, select or create the desired folder for the project, and then provide its URL. The function extracts the necessary information from the URL and updates the `cloudfs.s3` field in the DESCRIPTION file accordingly.

Usage

```
cloud_s3_attach(project = ".")
```

Arguments

`project`	Character. Path to a project. By default it is current working directory.

Value

This function does not return a meaningful value but modifies the DESCRIPTION file of the specified project to include the S3 folder path.

Examples

```
cloud_s3_attach()
```

`cloud_s3_browse` *Browse project's S3 folder*

Description

Opens project's S3 folder in browser.

Usage

```
cloud_s3_browse(path = "", root = NULL)
```

Arguments

`path`	(optional) Path inside the S3 folder to open. Defaults to the root level (path = "") of the project's S3 folder.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns `NULL`. The primary purpose of this function is its side effect: opening the specified S3 folder in a browser.

Examples

```
cloud_s3_browse()
cloud_s3_browse("data")
```

`cloud_s3_download` *Download a file from S3 to the local project folder*

Description

Retrieves a file from the project's S3 root folder and saves it to the local project folder, maintaining the original folder structure.

Usage

```
cloud_s3_download(file, root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns `NULL` after successfully downloading the file.

Examples

```
# downloads toy_data/demo.csv from project's S3 folder (provided it exists)
# and saves it to local 'toy_data' folder
cloud_s3_download("toy_data/demo.csv")

# clean up
unlink("toy_data", recursive = TRUE)
```

`cloud_s3_download_bulk` *Bulk Download Contents from S3*

Description

Downloads multiple files from an S3 folder based on the output dataframe from cloud_s3_ls. This function streamlines the process of downloading multiple files by allowing you to filter and select specific files from the S3 listing and then download them in bulk.

Usage

```
cloud_s3_download_bulk(content, quiet = FALSE, root = NULL)
```

Arguments

`content`	(data.frame) Output of `cloud_s3_ls()`
`quiet`	All caution messages may be turned off by setting this parameter to `TRUE`.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns the input `content` dataframe.

Examples

```
# provided there's a folder called "toy_data" in the root of your project's
# S3 folder, and this folder contains "csv" files
cloud_s3_ls("toy_data") |>
  filter(type == "csv") |>
  cloud_s3_download_bulk()

# clean up
unlink("toy_data", recursive = TRUE)
```

`cloud_s3_ls` *List Contents of Project's S3 Folder*

Description

Returns a tibble with names, timestamps, and sizes of files and folders inside the specified S3 folder.

Usage

```
cloud_s3_ls(path = "", recursive = FALSE, full_names = FALSE, root = NULL)
```

Arguments

`path`	(optional) Path inside the S3 folder. Specifies the subfolder whose contents should be listed. By default, when `path = ""`, lists root-level files and folders.
`recursive`	(logical) If `TRUE`, lists contents recursively in all nested subfolders. Default is `FALSE`.
`full_names`	(logical) If `TRUE`, folder path is appended to object names to give a relative file path.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

A tibble containing the names, last modification timestamps, and sizes in bytes of files and folders inside the specified S3 folder.

Examples

```
# list only root-level files and folders
cloud_s3_ls()

# list all files in all nested folders
cloud_s3_ls(recursive = TRUE)

# list contents of "plots/barplots" subfolder
cloud_s3_ls("plots/barplots")
```

`cloud_s3_read` *Read a file from S3*

Description

Retrieves and reads a file from the project's S3 folder. By default, the function attempts to determine the appropriate reading function based on the file's extension. However, you can specify a custom reading function if necessary.

Usage

```
cloud_s3_read(file, fun = NULL, ..., root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`fun`	A custom reading function. If `NULL` (default), the appropriate reading function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the reading function `fun`.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

The content of the file read from S3, with additional attributes containing metadata about the file.

Default reading functions

Here's how we identify a reading function based on file extension

- `.csv`: readr::read_csv
- `.json`: jsonlite::read_json
- `.rds`: base::readRDS
- `.sav`: haven::read_sav
- `.xls`: cloud_read_excel
- `.xlsx`: cloud_read_excel
- `.xml`: xml2::read_xml

Examples

```
# provided there are folders called "data" and "models" in the root of your
# project's main S3 folder and they contain the files mentioned below
cloud_s3_read("data/mtcars.csv")
cloud_s3_read("models/random_forest.rds")
cloud_s3_read("data/dm.sas7bdat", fun = haven::read_sas)
```

`cloud_s3_read_bulk` *Bulk Read Contents from S3*

Description

This function facilitates the bulk reading of multiple files from the project's designated S3 folder. By using cloud_s3_ls, you can obtain a dataframe detailing the contents of the S3 folder. Applying `cloud_s3_read_bulk` to this dataframe allows you to read all listed files into a named list. The function will, by default, infer the appropriate reading method based on each file's extension. However, if a specific reading function is provided via the `fun` parameter, it will be applied uniformly to all files, which may not be suitable for diverse file types.

Usage

```
cloud_s3_read_bulk(content, fun = NULL, ..., quiet = FALSE, root = NULL)
```

Arguments

`content`	(data.frame) Output of `cloud_s3_ls()`
`fun`	A custom reading function. If `NULL` (default), the appropriate reading function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the reading function `fun`.
`quiet`	All caution messages may be turned off by setting this parameter to `TRUE`.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

A named list where each element corresponds to the content of a file from S3. The names of the list elements are derived from the file names.

Examples

```
# provided there's a folder called "data" in the root of the project's main
# S3 folder, and it contains csv files
data_lst <-
  cloud_s3_ls("data") |>
  filter(type == "csv")  |>
  cloud_s3_read_bulk()
```

`cloud_s3_upload` *Upload a local file to S3*

Description

Uploads a local file from the project's directory to its corresponding location within the project's S3 root folder.

Usage

```
cloud_s3_upload(file, root = NULL)
```

Arguments

`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns `NULL` after successfully uploading the file.

Examples

```
# create a toy csv file
dir.create("toy_data")
write.csv(mtcars, "toy_data/mtcars.csv")

# uploads toy_data/mtcars.csv to 'data' subfolder of project's S3 folder
cloud_s3_upload("toy_data/mtcars.csv")
```

```
# clean up
unlink("toy_data", recursive = TRUE)
```

`cloud_s3_upload_bulk` *Bulk Upload Files to S3*

Description

This function facilitates the bulk uploading of multiple files from the local project folder to the project's designated S3 folder. By using cloud_local_ls, you can obtain a dataframe detailing the contents of the local folder. Applying `cloud_s3_upload_bulk` to this dataframe allows you to upload all listed files to S3.

Usage

```
cloud_s3_upload_bulk(content, quiet = FALSE, root = NULL)
```

Arguments

`content`	(data.frame) Output of `cloud_s3_ls()`
`quiet`	All caution messages may be turned off by setting this parameter to `TRUE`.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns the input `content` dataframe.

Examples

```
# create toy plots: 2 png's and 1 jpeg
dir.create("toy_plots")
png("toy_plots/plot1.png"); plot(rnorm(100)); dev.off()
png("toy_plots/plot2.png"); plot(hist(rnorm(100))); dev.off()
png("toy_plots/plot3.jpeg"); plot(hclust(dist(USArrests), "ave")); dev.off()

# upload only the two png's
cloud_local_ls("toy_plots")  |>
  dplyr::filter(type == "png")  |>
  cloud_s3_upload_bulk()

# clean up
unlink("toy_plots", recursive = TRUE)
```

`cloud_s3_write` *Write an object to S3*

Description

Saves an R object to a designated location in the project's S3 storage. If no custom writing function is specified, the function will infer the appropriate writing method based on the file's extension.

Usage

```
cloud_s3_write(x, file, fun = NULL, ..., local = FALSE, root = NULL)
```

Arguments

`x`	An R object to be written to S3.
`file`	Path to a file relative to project folder root. Can contain only letters, digits, '-', '_', '.', spaces and '/' symbols.
`fun`	A custom writing function. If `NULL` (default), the appropriate writing function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the writing function `fun`.
`local`	Logical, defaulting to `FALSE`. If `TRUE`, the function will also create a local copy of the file at the specified path. Note that some writing functions might not overwrite existing files unless explicitly allowed. Typically, such functions have a parameter (often named `overwrite`) to control this behavior. Check the documentation of the writing function used to determine the exact parameter name and pass it through the `...` argument if necessary. Alternatively, you can define an anonymous function for `fun` that calls a writing function with the overwriting option enabled.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns `NULL` after successfully writing the object to S3.

Default writing functions

Here's how we identify a writing function based on file extension

- `.csv`: readr::write_csv
- `.json`: jsonlite::write_json
- `.rds`: base::saveRDS
- `.xls`: writexl::write_xlsx
- `.xlsx`: writexl::write_xlsx
- `.sav`: haven::write_sav
- `.xml`: xml2::write_xml

Examples

```
# write mtcars dataframe to mtcars.csv in data folder
cloud_s3_write(mtcars, "data/mtcars.csv")
cloud_s3_write(random_forest, "models/random_forest.rds")

# provide custom writing function with parameters
cloud_s3_write(c("one", "two"), "text/count.txt", writeLines, sep = "\n\n")
```

`cloud_s3_write_bulk` *Write multiple objects to S3 in bulk*

Description

This function allows for the bulk writing of multiple R objects to the project's designated S3 folder. To prepare a list of objects for writing, use cloud_object_ls, which generates a dataframe listing the objects and their intended destinations in a format akin to the output of cloud_s3_ls. By default, the function determines the appropriate writing method based on each file's extension. However, if a specific writing function is provided via the `fun` parameter, it will be applied to all files, which may not be ideal if dealing with a variety of file types.

Usage

```
cloud_s3_write_bulk(
  content,
  fun = NULL,
  ...,
  local = FALSE,
  quiet = FALSE,
  root = NULL
)
```

Arguments

`content`	(data.frame) output of `cloud_object_ls()`
`fun`	A custom writing function. If `NULL` (default), the appropriate writing function will be inferred based on the file's extension.
`...`	Additional arguments to pass to the writing function `fun`.
`local`	Logical, defaulting to `FALSE`. If `TRUE`, the function will also create a local copy of the file at the specified path. Note that some writing functions might not overwrite existing files unless explicitly allowed. Typically, such functions have a parameter (often named `overwrite`) to control this behavior. Check the documentation of the writing function used to determine the exact parameter name and pass it through the `...` argument if necessary. Alternatively, you can define an anonymous function for `fun` that calls a writing function with the overwriting option enabled.

`quiet`	all caution messages may be turned off by setting this parameter to `TRUE`.
`root`	S3 path of the project root. This serves as the reference point for all relative paths. When left as `NULL`, the root is automatically derived from the `cloudfs.s3` field of the project's DESCRIPTION file.

Value

Invisibly returns the input `content` dataframe.

Examples

```
# write two csv files: data/df_mtcars.csv and data/df_iris.csv
cloud_object_ls(
  dplyr::lst(mtcars = mtcars, iris = iris),
  path = "data",
  extension = "csv",
  prefix = "df_"
) |>
cloud_s3_write_bulk()
```

Index