Package 'path.chain'

July 23, 2025

Type Package

Title Concise Structure for Chainable Paths

Version 1.0.0

Description Provides path_chain class and functions, which facilitates loading and saving directory structure in YAML configuration files via 'config' package.
The file structure you created during exploration can be transformed into legible section in the config file, and then easily loaded for further usage.

License MIT + file LICENSE

Encoding UTF-8

BugReports https://github.com/krzjoa/path.chain/issues

URL https://github.com/krzjoa/path.chain,
https://krzjoa.github.io/path.chain/

RoxygenNote 7.3.2

Suggests testthat (>= 2.1.0), knitr, rmarkdown, config, yaml, fs, magrittr, logger

VignetteBuilder knitr

Imports rlang, stringi

NeedsCompilation no

Author Krzysztof Joachimiak [aut, cre] (ORCID: <https://orcid.org/0000-0003-4780-7947>), Peter Wurm [ctb]

Maintainer Krzysztof Joachimiak <joachimiak.krzysztof@gmail.com>

Repository CRAN

Date/Publication 2024-10-08 22:30:02 UTC

Contents

`path.chain-package` *path.chain: Concise Structure for Chainable Paths*

Description

The aim of this package is to provide tools, which allow us to represent directory structure as nested R objects. It can be easily saved as .yaml files so that we can later load it and use in our project.

Author(s)

Krzysztof Joachimiak

See Also

Useful links:

- https://github.com/krzjoa/path.chain
- https://krzjoa.github.io/path.chain/
- Report bugs at https://github.com/krzjoa/path.chain/issues

as.list — *Convert object of type 'path_chain' to list*

Description

Convert object of type 'path_chain' to list

Usage

```
## S3 method for class 'path_chain'
as.list(x, ..., root.name = "root.dir")
```

Arguments

x	a path_chain object
...	elipsis for API consistency, does nothing
root.name	key for root directory; default: 'root.dir'

Examples

```
tmp <- create_temp_dir("files")
create_sample_dir(tmp)
path.chain <- path_chain(tmp)
as.list(path.chain)
```

as_config — *Prepare list to be saved as config .yaml file*

Description

This function is provided to keep compatibility with '{config}' package, which requires existence of default key. Additionally, we can at once wrap our structure with some other keys, in order to not to mix directory structure with different keys.

Usage

```
as_config(x, config = "default", wrap = "dirs", ...)

## S3 method for class 'path_chain'
as_config(x, config = "default", wrap = "dirs", ..., root.name = "root.dir")

## S3 method for class 'list'
as_config(x, config = "default", wrap = "dirs", ...)
```

Arguments

x	list with directory structure
config	configuration name
wrap	key name to wrap directory structure
...	additional arguments (not used at the moment)
root.name	key for root directory (for path_chain only)

Value

list compatible with ‘{config}‘ package

Examples

```
library(magrittr)
# Initializing sample directory
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
full_path_chain(tmp, "kRoot", naming_k) %>%
   list(kDirs = .) %>%
   list(default = .) %>%
   yaml::write_yaml(temp_path("config.yaml"))
# We can simply use such function
full_path_chain(tmp, "kRoot", naming_k) %>%
   as_config("default", "kDirs") %>%
   yaml::write_yaml(temp_path("config.yaml"))
```

---

as_path_chain *Create chainable path*

Description

This function always treats first object in the nested list as a subdirectory root path

Usage

```
as_path_chain(nested.list, root.name = "kRoot")
```

Arguments

nested.list	‘list‘ object with nested lists/strings inside
root.name	key for root directory

Value

path_chain object

Examples

```
library(magrittr)
# Manually created nested list
nested.list <- list(kRoot = "root", "file1.txt", list("subdir", "file2.csv"))
chainable.path <- as_path_chain(nested.list)
class(chainable.path)
chainable.path$.
chainable.path$subdir$files2.csv
# Nested list from config file
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
fs::dir_tree(tmp)
path_chain(tmp, naming = naming_k) %>%
  as.list(root.name = "kRoot") %>%
  as_config("default", "kDirs") %>%
  yaml::write_yaml(temp_path("config.yaml"))
chainable.path <- config::get("kDirs", "default", temp_path("config.yaml")) %>%
 as_path_chain()
class(chainable.path)
chainable.path$.
chainable.path$kData$kExample1
```

`create_sample_dir` *Create sample directory*

Description

Creates sample nested directory to test and learn path.chain package

Usage

```
create_sample_dir(path = "files", override = FALSE)
```

Arguments

`path`	path for the new sample folder
`override`	boolean: override folder if it already exists

Examples

```
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
list.files(tmp, all.files = TRUE, recursive = TRUE, include.dirs = TRUE)
fs::dir_tree(tmp)
```

create_temp_dir *Create temporary diectory and return its name*

Description

Create temporary diectory and return its name

Usage

```
create_temp_dir(
  ...,
  warn = FALSE,
  recursive = FALSE,
  fsep = .Platform$file.sep
)
```

Arguments

...	arbitrary character objects
warn	warn, if folder already exists
recursive	ogical. Should elements of the path other than the last be created? If true, like the Unix command mkdir -p
fsep	the path separator to use

Examples

```
# Simply create and return temporal directory
create_temp_dir()
# Create temp dir and return concatenated path
# Keep in mind, that 'files' and 'report_2020.xls' will not be created.
create_temp_dir("files", "report_2020.xls")
```

file_path *Construct path to file without doubled separators*

Description

Construct path to file without doubled separators

Usage

```
file_path(..., fsep = .Platform$file.sep)
```

Arguments

...	character vectors
fsep	the path separator to use

Value

character file path

Examples

```
file.path("files/", "data/", "cars.RData")
file_path("files/", "data/", "cars.RData")
```

---

full_path_chain *Full path chain*

Description

'full_path_chain' represents another approach to creating chainable paths In contrast to 'path_chain', this functon creates just a list with nested list with full paths as a leaves.

Usage

```
full_path_chain(path = ".", root.name = ".", naming = basename)
```

Arguments

path	root path
root.name	naming convention for root directory
naming	naming function

Value

list of lists and character objects

Examples

```
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
fs::dir_tree(tmp)
chainable.path <- full_path_chain(tmp)
chainable.path
```

naming_k *Naming convention, which adds k prefix for each key, capitalizes and removes file extension*

Description

Naming convention, which adds k prefix for each key, capitalizes and removes file extension

Usage

```
naming_k(path)
```

Arguments

path	full path or its element

Examples

```
library(magrittr)
naming_k("path/to/myfile.txt")
# Using with full_path_chain
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
full.path.chain <- full_path_chain(tmp, naming = naming_k)
full.path.chain
tmp <- create_temp_dir("files")
create_sample_dir(tmp)
# Using with path_chain / create_path_chain
path.chain <- path_chain(tmp, naming = naming_k)
path.chain %>%
  as.list()
```

on_path_not_exists *Function called if path does not exists*

Description

Function called if path does not exists

Usage

```
on_path_not_exists(fun)
```

Arguments

fun	a function, one-side formula or NULL; if missing, returns value of the path.chain.on.path.not.exists option

Examples

```
# We'll create an options backup for this example
old.options <- options()
on_path_not_exists(print)
on_path_not_exists()
options(old.options)
```

---

on_validate_path *Function called to validate path correctness*

Description

Function called to validate path correctness

Usage

```
on_validate_path(fun)
```

Arguments

fun	a function; if missing, returns value of the path.chain.on.path.not.exists option

Examples

```
# We'll create an options backup for this example
old.options <- options()
is_path_valid <- function(x) grepl("\\.fst", x)
on_validate_path(is_path_valid)
on_validate_path()
options(old.options)
```

---

path_chain *Get directory structure and create path_chain object*

Description

Returns ‘path_chain‘ object, which reflects structure of the folder passed with ‘path‘ param

Usage

```
path_chain(path, naming = basename, levels = +Inf, only.directories = FALSE)
```

Arguments

path	root of the directory structure
naming	function which defines naming convention
levels	number of hierarchy levels that recursion should go deep; defaults to +Inf
only.directories	boolean to ignore files and only considers directories.

Value

path_chain object

Examples

```
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
fs::dir_tree(tmp)
chainable.path <- path_chain(tmp)
chainable.path$data$persons.csv
# With customized naming convention
chainable.path <- path_chain(tmp, naming = naming_k)
chainable.path$kData$kPersons
```

---

path_children *Get children nodes, i.e. all the suddiectories in the given directory*

Description

Get children nodes, i.e. all the suddiectories in the given directory

Usage

```
path_children(path.chain)
```

Arguments

path.chain	object of ‘path_chain‘ class

Value

a list of ‘path_chain‘ objects

Examples

```
tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
path.chain <- path_chain(tmp)
path_children(path.chain)
```

path_link *Creates a link of path chain - a directory or a file*

Description

It returns basic package's object: an object representing a link in the chain. Each link has the path_chain class - it can represents a one-element path chain

Usage

```
path_link(node = NULL, children = NULL)
```

Arguments

node	Current node name; character
children	list of children - path_chains

Value

path_chain object

Examples

```
# If we want to create our chain manually, we have start from the leaves
level2.b <- path_link("fileA.RData")
level2.a <- path_link("fileB.RData")
level1   <- path_link("data", list(level2.a = level2.a , level2.b = level2.b))
root     <- path_link("files", list(level1))
# Print root path
root$.
# Print file path using chaining
root$data$level2.a
```

print *Print path_chain object*

Description

Print path_chain object

Usage

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

Arguments

x	'path_chain' object
...	elipsis for API consistency, does nothing

Examples

```
level2.b <- path_link("fileA.RData")
level2.a <- path_link("fileB.RData")
level1   <- path_link("data", list(level2.a = level2.a , level2.b = level2.b))
root     <- path_link("files", list(level1))
print(root)

tmp <- create_temp_dir("files")
create_sample_dir(tmp, override = TRUE)
chainable.path <- path_chain(tmp)
print(chainable.path)
```

---

temp_path *Construct path to file in a temporary directory*

Description

Construct path to file in a temporary directory

Usage

```
temp_path(..., fsep = .Platform$file.sep)
```

Arguments

...	arbitrary character objects
fsep	the path separator to use.

Details

Be careful: if you call this function, it only creates a path for temporary file/dir. All the rest has to be created on your own, e.g. calling dir.create function.

Value

a path

Examples

```
temp_path("files", "report.csv")
```

`$.path_chain` *Access path_chain object*

Description

Access path_chain object

Usage

```
## S3 method for class 'path_chain'
node$child
```

Arguments

node	path_chain
child	nested path_chain name

Value

path_chain or character, if path indicates leaf of structure tree

Examples

```
#' If we want to create our chain manually, we have start from the leaves
level2.b <- path_link("fileA.RData")
level2.a <- path_link("fileB.RData")
level1   <- path_link("data", list(level2.a = level2.a , level2.b = level2.b))
root     <- path_link("files", list(level1))
# Print root path
root$.
# Print file path using chaining
root$data$level2.a
```

Index