Package: fakin.path.app (via r-universe)

October 11, 2024

Title Shiny App to Visualise File Paths

Version 0.3.0

Description This package contains an R Shiny App that loads file path information from a file and displays the paths in different ways. The aim of the app is to find weaknesses in the folder structure.

License MIT + file LICENSE

URL https://github.com/KWB-R/fakin.path.app

BugReports https://github.com/KWB-R/fakin.path.app/issues

Encoding UTF-8

LazyData true

Suggests covr (>= 3.2.1), kwb.db (>= 0.3.0), RMySQL (>= 0.10.17), knitr (>= 1.23), rmarkdown (>= 1.13), testthat (>= 2.2.1)

RoxygenNote 6.1.1

Imports bit64 (>= 0.9.7), data.table (>= 1.11.8), dplyr (>= 0.8.1), DT (>= 0.7), fs (>= 1.3.1), gdata (>= 2.18.0), ggplot2 (>= 3.2.0), htmltools (>= 0.3.6), jsTree (>= 1.0.1), kwb.file (>= 0.3.0), kwb.utils (>= 0.5.0), magrittr (>= 1.5), networkD3 (>= 0.4), pathlist (>= 0.2.0), plotly (>= 4.9.0), readr (>= 1.3.1), rlang (>= 0.4.0), shiny (>= 1.3.2), shinyFiles (>= 0.7.3), shinyjs (>= 1.0), treemap (>= 2.4.2), wordcloud (>= 2.6)

VignetteBuilder knitr

Remotes github::hsonne/pathlist, github::kwb-r/kwb.db, github::kwb-r/kwb.file, github::kwb-r/kwb.utils

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

RemoteUrl https://github.com/KWB-R/fakin.path.app

RemoteRef HEAD

RemoteSha 3882d91321a5b40a2bda685c1e40f6acb2f503a1

Contents

`get_and_save_file_info` *Get and Save File Information*

Description

Get and Save File Information

Usage

```
get_and_save_file_info(root_dir, output_dir, check_dirs = TRUE,
  format = "%Y-%m-%d_%H%M", ...)
```

Arguments

`root_dir`	path to the directory from which to start searching for files
`output_dir`	path to the output directory. In this directory, a file "path-info_<date-time>_<parent-folder>.csv" will be generated with <date-time> being a date and time string in yyyy-mm-dd_HHMM format and <parent-folder> being the last path segment of `root_dir`
`check_dirs`	if `TRUE` (default) it is checked in advance if both `root_dir` and `output_dir` exist. Switch this off if e.g. network paths are wrongly considered to be non-existing.
`format`	format string specifying how to format the part of the filename intended to contain date (and, if required, time) information. Default: "%Y-%m-%d_%H%M"
`...`	further arguments passed to `get_recursive_file_info` and finally to `fs::dir_info`. Set e.g. `fail = FALSE` to avoid failure due to insufficient access permissions.

Value

full path to the file to which all file information were written

---

get_recursive_file_info   *Call file.info recursively on files below a root folder*

Description

Call file.info recursively on files below a root folder

Usage

```
get_recursive_file_info(root_dir, pattern = NULL, all = TRUE, ...,
  dbg = TRUE)
```

Arguments

`root_dir`	path to the root directory from which to start the recursive search for files
`pattern`	regular expression matching the names of the files to be considered. By default, all files are considered.
`all`	if `TRUE` (default) hidden files are also returned
`...`	further arguments passed to `fs::dir_info`
`dbg`	if `TRUE` (default) progress messages are shown

---

guess_file_metadata   *Guess Metadata about a Text File*

Description

Guess Metadata about a Text File

Usage

```
guess_file_metadata(file, n_first_rows = 1000, ...)
```

Arguments

`file`	path to text file
`n_first_rows`	number of first rows of `file` from which to guess the meta information.
`...`	further arguments passed to `fakin.path.app:::read_lines`, such as `fileEncoding`

Value

data frame with columns

- `paths`: does the file seem to contain path information, i.e. were slashes or backslashes found?
- `forbidden`: does the file contain characters that are forbidden in file paths?
- `header`: does the file seem to contain a header row?
- `windows`: are the paths given in "windows"-style, i.e. are the path segments separated by backslash?
- `sep`: column separator guessed
- `ncol`: number of columns guessed

and attributes

- `file`: a copy of the file path given in `file`
- `first_rows`: first `n_first_rows` rows of `file`
- `columns` (optional): column headers if the file is assumed to contain a header row

---

`left_substring_equals` *Is Left Substring of X Equal To Y?*

Description

Is Left Substring of X Equal To Y?

Usage

```
left_substring_equals(x, y)
```

Arguments

`x`	String of which the left part is compared with `y`
`y`	String to be compared with the left part of `x`

`name_is_ok` *Is the Name Ok According to Our Best Practices?*

Description

Is the Name Ok According to Our Best Practices?

Usage

```
name_is_ok(x, mildness = 1)
```

Arguments

`x`	vector of character
`mildness`	level of mildness. 1: not mild, all characters must be hyphen or alphanumeric or dot or underscore, 2: more mild, all characters must be one of the above or space

Value

vector of logical as long as `x`

Examples

```
name_is_ok(c("a", "$", ".", " "))
name_is_ok(c("a", "$", ".", " "), mildness = 2)
```

`name_to_traffic_light` *Get Traffic Light Colours for Names*

Description

Get Traffic Light Colours for Names

Usage

```
name_to_traffic_light(x)
```

Arguments

`x`	character of (file or folder) names, e.g. as they appear as node labels in the plot generated with `plot_path_network`

Value

vector of colour strings each of which is `green` (name does comply with naming convention), `yellow` (name does almost comply with naming convention), `red` (name does not comply with naming convention).

Examples

```
# Define a vector of names
x <- c("has_speci&l", "has space", "is_ok")

# Colour names by their compliance with naming convention
name_to_traffic_light(x)
```

`plot_all_treemaps` *Plot Treemaps for All given Path Infos*

Description

Plot Treemaps for All given Path Infos

Usage

```
plot_all_treemaps(path_infos, as_png = TRUE, ...)
```

Arguments

`path_infos`	list of data frames each of which contains file path information as returned by `read_file_info`
`as_png`	if `TRUE` the plots are saved to png-files in `tempdir()`. The name is then taken from the names of the elements in `path_infos`. Otherwise the plot go into the current graphical device.
`...`	further arguments passed to `plot_treemaps_from_path_data`, such as `n_levels`

Value

for `as_png = TRUE` vector of paths to the created png files.

plot_path_network *Plot Paths as Sankey Network*

Description

Plot Paths as Sankey Network

Usage

```
plot_path_network(paths, max_depth = 3, nodePadding = 8,
  nodeHeight = 10, sinksRight = FALSE, remove_common_root = TRUE,
  names_to_colours = name_to_traffic_light, height = NULL, ...,
  method = 1, weight_by = c("n_files", "size", "none")[1],
  sizes = NULL)
```

Arguments

`paths`	character vector of paths
`max_depth`	maximum depth of paths to be shown
`nodePadding`	passed to `sankeyNetwork`, see there. Decrease this value (e.g. ‘nodePadding = 0‘) if there are many nodes to plot and the plot does not look as expected
`nodeHeight`	height of a node in pixels. Used to calculate the total plot height.
`sinksRight`	passed to `sankeyNetwork`, see there
`remove_common_root`	remove the common root parts? (default: TRUE)
`names_to_colours`	if not `NULL` expected to be a function that accepts a vector of (node) names and returns a vector of (colour) names of same length. This function will be called by `plot_path_network` to determine the colour for each node based on its name. By default, the function `name_to_traffic_light` is called.
`height`	plot height in pixels, passed to `sankeyNetwork`. If `NULL`, the height is calculated based on `nodeHeight`, `nodePadding` and the maximum number of nodes at one folder depth.
`...`	further arguments passed to `sankeyNetwork`, such as `nodeWidth`, `nodePadding`, `fontSize`
`method`	if `1` (default) the function behaves as before, another value activates the new preparation of paths accepting/using an object of class **pathlist**
`weight_by`	one of `"n_files"`, `"size"`, `"none"`. Specifies whether to set the link widths according to the total number or total size of files in subsequent folders or by setting all links to the same width.
`sizes`	file sizes corresponding to the `paths`

Value

object representing an HTML page

Examples

```
# Get the paths to all folders on the desktop
paths <- dir(system.file(package = "fakin.path.app"), recursive = TRUE)

# Plot the folder network
plot_path_network(paths)
```

---

plot_treemaps_from_path_data

Plot Treemaps Given File Path Data

Description

Plot Treemaps Given File Path Data

Usage

```
plot_treemaps_from_path_data(path_data, root_path = "", name = "root",
  as_png = FALSE, n_levels = 2, output_dir = tempdir(),
  type = "value", args_png = list(), n_biggest = -1, depth = 1,
  types = c("size", "files"))
```

Arguments

path_data	data frame containing file path information as returned by read_file_info
root_path	path to the folder that contains all paths to be considered. By setting the root path to "/path/to/root" you can "zoom into" the treeplot, showing the contents below "/path/to/root" only. If root_path is "" (default) all paths in path_data are considered.
name	name to be used in png file name if as_png is set. If path_data is a list, the names of the list elements are used.
as_png	if TRUE (default) the plots are saved to png files in the directory given in output_dir (tempdir() by default). Otherwise they are plotted into the active graphical device.
n_levels	number of folder depth levels to be shown in the plots
output_dir	path to output directory if as_png = TRUE. Default: tempdir()
type	passed to treemap
args_png	list of arguments passed to png if as_png = TRUE
n_biggest	vector of integer, specifying the number(s) of biggest folders (in terms of size and number of files) in which to "zoom into". The position in the vector represents the folder depth. For example, if n_biggest = c(2, 1), the first element (2) indicates that sub-treemaps are produced for the two biggest subfolders below root_path: for root_path/biggest-1 and for root_path/biggest-2.

	The second element (1) indicates that further treemaps are generated only for the biggest subfolders below `root_path/biggest-1` and `root_path/biggest-2`, respectively, in each case. The length of the vector `n_biggest` determines the maximal depth until which to generate treemaps. By setting an element to `-1L` you specify that sub-treemaps are generated for each subfolder on the corresponding folder depth.
`depth`	current depth of recursion
`types`	type(s) of treeplots: one or both of `c("size", "files")` (the default).

`prepare_for_treemap` *Prepare and Filter Path Data for Treemap Plot*

Description

Prepare and Filter Path Data for Treemap Plot

Usage

```
prepare_for_treemap(path_data, root_path = "", variable = "size", ...)
```

Arguments

`path_data`	data frame as returned by `read_file_info`
`root_path`	path to the folder that contains all paths to be considered. By setting the root path to "/path/to/root" you can "zoom into" the treeplot, showing the contents below "/path/to/root" only. If `root_path` is `""` (default) all paths in `path_data` are considered.
`variable`	name(s) of variable(s) to be selected. Default: "size"
`...`	further arguments passed to `remove_common_root`, such as `n_keep` (number of last segments to be kept from the common first part of all paths)

`read_file_paths` *Read File Paths from a File*

Description

The function tries to guess what type of file is given to the function and calls the appropriate function to read the file. The aim of this function is to provide a common result format independent from the type of file that was read.

Usage

```
read_file_paths(file, metadata = NULL)
```

Arguments

`file`	file containing file path information (path only or additional information such as file type, size or creation/modification time, etc.)
`metadata`	data frame containing metadata about the file. If given, it must look as what `guess_file_metadata` returns. If `NULL` the same function is called to guess metadata about the file.

Value

data frame with columns...

`read_lines` *Read Lines by Giving the File Encoding*

Description

Read Lines by Giving the File Encoding

Usage

```
read_lines(file, ..., encoding = "unknown", fileEncoding = "")
```

Arguments

`file`	a connection object or character string
`...`	arguments passed to `readLines`
`encoding`	passed to `readLines`.
`fileEncoding`	The name of the encoding to be assumed. Passed as `encoding` to `file`, see there.

`run_app` *Run the Shiny App*

Description

Run the Shiny App

Usage

```
run_app(path_database = default_targetdir(), ...)
```

Arguments

`path_database`	if not `NULL` the path to a folder containing text files with path information. Default: `fakin.path.app:::default_targetdir()`
`...`	further `key = value` pairs to be used as global variables

`run_app_scan` *Run the App that Stores File Information to CSV Files*

Description

Run the App that Stores File Information to CSV Files

Usage

```
run_app_scan()
```

`write_csv` *Write Data Frame to CSV File*

Description

Write Data Frame to CSV File

Usage

```
write_csv(data, file, sep = ";", version = 2, ...)
```

Arguments

`data`	data frame
`file`	path to CSV file to be written
`sep`	column separator
`version`	determines which function to use for writing the CSV file 1: `write.table`, 2: `fwrite`
`...`	further arguments passed to `write.table` or `fwrite`

write_file_info *Write File Information to CSV File*

Description

Write File Information to CSV File

Usage

```
write_file_info(file_info, file, version = 2)
```

Arguments

`file_info`	data frame as returned by `get_recursive_file_info`
`file`	path to CSV file to be written
`version`	determines which function to use for writing the CSV file 1: `write.table`, 2: `fwrite`

Index