Package: hoardr (via r-universe)

October 28, 2024

Type Package

Title Manage Cached Files

Description Suite of tools for managing cached files, targeting use in other R packages. Uses 'rappdirs' for cross-platform paths. Provides utilities to manage cache directories, including targeting files by path or by key; cached directories can be compressed and uncompressed easily to save disk space.

Version 0.5.4

License MIT + file LICENSE

URL https://docs.ropensci.org/hoardr/, https://github.com/ropensci/hoardr

BugReports https://github.com/ropensci/hoardr/issues

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Encoding UTF-8

Imports R6, rappdirs, digest

Suggests testthat, knitr, rmarkdown

RoxygenNote 7.2.3

X-schema.org-applicationCategory Data

X-schema.org-keywords caching, data, files, xml, pdf

X-schema.org-isPartOf https://ropensci.org

Config/testthat/edition 3

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

RemoteUrl https://github.com/ropensci/hoardr

RemoteRef master

RemoteSha f64427345e5f5173981b81aa3cec688cf9a1dc81

Contents

`hoardr-package` *hoardr*

Description

Manage Cached Files

Details

`hoardr` is a tiny package with just a single export `hoard()`. The package goal is to make it easy to setup, use, and manage cached files for another R package. In addition, you can export functions in your own package using `hoardr` for users to manage their cached files.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

`hoard` *hoardr class*

Description

hoardr class

Arguments

`path`	(character) a path to cache files in. required
`type`	(character) type of cache. One of "user_cache_dir" (default), "user_log_dir", "user_data_dir", "user_config_dir", "site_data_dir", "site_config_dir". Can also pass in any function that gives a path to a directory, e.g., `tempdir()`. required.

Details

For the purposes of caching, you'll likely want to stick with `user_cache_dir`, but you can change the type of cache with the `type` parameter.

`hoard` is just a tiny wrapper around `HoardClient$new()`, which isn't itself exported, but you can use it if you want via `:::`

Methods

`cache_path_get()` Get the cache path **return**: (character) path to the cache directory

`cache_path_set(path = NULL, type = "user_cache_dir", prefix = "R", full_path = NULL)` Set the cache path. By default, we set cache path to `file.path(user_cache_dir, prefix, path)`. Note that this does not actually make the directory, but just sets the path to it.

- path (character) the path to be appended to the cache path set by `type`
- type (character) the type of cache, see rappdirs
- prefix (character) prefix to the `path` value. Default: "R"
- full_path (character) instead of using `path`, `type`, and `prefix` just set the full path with this parameter

return: (character) path to the cache directory just set

`list()` List files in the directory (full file paths) **return**: (character) vector of file paths for files in the cache

`mkdir()` Make the directory if doesn't exist already **return**: `TRUE`, invisibly

`delete(files, force = TRUE)` Delete files by name

- files (character) vector/list of file paths
- force (logical) force deletion? Default: `TRUE`

return: nothing

`delete_all(force = TRUE)` Delete all files

- force (logical) force deletion? Default: `FALSE`

return: nothing

`details(files = NULL)` Get file details

- files (character) vector/list of file paths

return: objects of class `cache_info`, each with brief summary info including file path and file size

`keys(algo = "md5")` Get a hash for all files. Note that these keys may not be unique if the files are identical, leading to identical hashes **return**: (character) hashes for the files

`key(x, algo = "md5")` Get a hash for a single file. Note that these keys may not be unique if the files are identical, leading to identical hashes

- x (character) path to a file
- algo (character) the algorithm to be used, passed on to `digest::digest()`, choices: md5 (default), sha1, crc32, sha256, sha512, xxhash32, xxhash64 and murmur32.

return: (character) hash for the file

`files()` Get all files as HoardFile objects **return**: (character) paths to the files

`compress()` Compress files into a zip file - leaving only the zip file **return**: (character) path to the cache directory

`uncompress()` Uncompress all files and remove zip file **return**: (character) path to the cache directory

`exists(files)` Check if files exist

- files: (character) one or more files, paths are optional

return: (data.frame) with two columns:

- files: (character) file path
- exists: (boolean) does it exist or not

Examples

```
(x <- hoard())
x$cache_path_set(path = "foobar", type = 'tempdir')
x
x$path
x$cache_path_get()

# Or you can set the full path directly with `full_path`
mydir <- file.path(tempdir(), "foobar")
x$cache_path_set(full_path = mydir)
x
x$path
x$cache_path_get()

# make the directory if doesn't exist already
x$mkdir()

# list files in dir
x$list()
cat(1:10000L, file = file.path(x$cache_path_get(), "foo.txt"))
x$list()

# add more files
cat(letters, file = file.path(x$cache_path_get(), "foo2.txt"))
cat(LETTERS, file = file.path(x$cache_path_get(), "foo3.txt"))

# see if files exist
x$exists("foo.txt") # exists
x$exists(c("foo.txt", "foo3.txt")) # both exist
x$exists(c("foo.txt", "foo3.txt", "stuff.txt")) # one doesn't exist

# cache details
x$details()

# delete files by name - we prepend the base path for you
x$delete("foo.txt")
x$list()
x$details()

# delete all files
cat("one\ntwo\nthree", file = file.path(x$cache_path_get(), "foo.txt"))
cat("asdfasdf asd fasdf", file = file.path(x$cache_path_get(), "bar.txt"))
x$delete_all()
x$list()

# make/get a key for a file
cat(1:10000L, file = file.path(x$cache_path_get(), "foo.txt"))
x$keys()
x$key(x$list()[1])

# as files
Map(function(z) z$exists(), x$files())
```

```
# compress and uncompress
x$compress()
x$uncompress()

# reset cache path
x$cache_path_set(path = "stuffthings", type = "tempdir")
x
x$cache_path_get()
x$list()

# cleanup
unlink(x$cache_path_get())
```

Index