Package 'pkgfilecache'

July 23, 2025

Type Package

Title Download and Manage Optional Package Data

Version 0.1.5

Maintainer Tim Schäfer <ts+code@rcmd.org>

Description Manage optional data for your package. The data can be hosted anywhere, and you have to give a Uniform Resource Locator (URL) for each file. File integrity checks are supported. This is useful for package authors who need to ship more than the 5 Megabyte of data currently allowed by the the Comprehensive R Archive Network (CRAN).

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/dfsp-spirit/pkgfilecache

BugReports https://github.com/dfsp-spirit/pkgfilecache/issues

Suggests knitr, rmarkdown, testthat (>= 2.1.0)

Imports downloader, rappdirs, curl

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation no

Author Tim Schäfer [aut, cre] (ORCID: <https://orcid.org/0000-0002-3683-8070>)

Repository CRAN

Date/Publication 2024-02-02 20:30:02 UTC

Contents

`are_files_available` *Check whether the given files exist in the package cache.*

Description

Check whether the given files exist in the package cache. You can pass MD5 sums, which will be verified and only files with correct MD5 hash will count as existing.

Usage

```
are_files_available(pkg_info, relative_filenames, md5sums = NULL)
```

Arguments

`pkg_info`	named list. Package identifier, see get_pkg_info() on how to get one.
`relative_filenames`	vector of strings. A vector of filenames, relative to the package cache.
`md5sums`	vector of strings or NULL. A list of MD5 checksums, one for each file in param 'relative_filenames', if not NULL. If given, the files will only be reported as existing if the MD5 sums match.

Value

logical vector. For each file, whether it passed the check.

Examples

```
pkg_info = get_pkg_info("mypackage")
is_available = are_files_available(pkg_info, c("file1.txt", "file2.txt"))
```

`ensure_files_available` *Ensure all given files exist in the file cache, download them if they are not.*

Description

Ensure all given files exist in the file cache, download them if they are not.

Usage

```
ensure_files_available(
  pkg_info,
  relative_filenames,
  urls,
  files_are_binary = NULL,
  md5sums = NULL,
  on_errors = "warn",
  download = TRUE
)
```

Arguments

`pkg_info`	named list. Package identifier, see get_pkg_info() on how to get one.
`relative_filenames`	vector of strings. A vector of filenames, realtive to the package cache.
`urls`	vector of strings. For each file, a remote URL where to download the file. Will be passed to 'downloader::download', see that function for URL encoding details.
`files_are_binary`	logical vector. For each file, whether it is binary. Only required on Windows, when files need to be downloaded. See 'downloader::download' docs for details.
`md5sums`	vector of strings or NULL. A list of MD5 checksums, one for each file in param 'relative_filenames', if not NULL. If given, the files will only be reported as existing if the MD5 sums match.
`on_errors`	string. What to do if getting the files failed. One of c("warn", "stop", "ignore"). At the end, files are checked using 'files_available'(including MD5 if given). Depending on the check results, the behaviours triggered are: "warn": Print a warning for each file that failed the check. "stop": Stop the script, i.e., the whole application. "ignore": Do nothing. You can still react using the return value.
`download`	logical. Whether to try downloading missing files. Defaults to TRUE. Existing files (with correct MD5 if available) will never be downloaded.

Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using get_filepath(). "missing": vector of strings. The names of the files that this function was unable to retrieve. "file_status": Logical array indicating whether the files are available. Order is identical to the one in argument 'relative_filenames'.

Examples

```
pkg_info = get_pkg_info("mypackage");
local_relative_filenames = c("local_file1.txt", "local_file2.txt");
bu = "https://raw.githubusercontent.com/dfsp-spirit/";
url1 = paste(bu, "pkgfilecache/master/inst/extdata/file1.txt", sep="");
url2 = paste(bu, "pkgfilecache/master/inst/extdata/file2.txt", sep="");
urls = c(url1, url2);
md5sums = c("35261471bcd198583c3805ee2a543b1f", "85ffec2e6efb476f1ee1e3e7fddd86de");
res = ensure_files_available(pkg_info, local_relative_filenames, urls, md5sums=md5sums);
erase_file_cache(pkg_info); # clear full cache
```

---

erase_file_cache *Delete the full package cache directory for the given package.*

Description

Delete the full package cache directory for the given package.

Usage

```
erase_file_cache(pkg_info)
```

Arguments

pkg_info	named list. Package identifier, see get_pkg_info() on how to get one.

Value

integer. The return value of the unlink() call: 0 for success, 1 for failure. See the unlink() documentation for details.

get_absolute_path_for_files *Construct absolute path for package cache files.*

Description

Construct absolute path for package cache files.

Usage

```
get_absolute_path_for_files(pkg_info, relative_filenames)
```

Arguments

pkg_info	named list. Package identifier, see get_pkg_info() on how to get one.
relative_filenames	vector of strings. A vector of filenames, relative to the package cache.

Value

vector of strings. The absolute paths.

Examples

```
rel_files = c("file1.txt", "file2.txt")
pkg_info = get_pkg_info("mypackage")
abs_paths = get_absolute_path_for_files(pkg_info, rel_files)
```

get_cache_dir *Get the absolute path of the package cache.*

Description

Get the absolute path of the package cache.

Usage

```
get_cache_dir(pkg_info)
```

Arguments

pkg_info	named list. Package identifier, see get_pkg_info() on how to get one.

Value

string. The absolute path of the package cache. It is constructed by calling 'rappdirs::user_data_dir' with the package, author, and version if available. If the author is null, the package name is also used as the author name.

Examples

```
pkg_info = get_pkg_info("mypackage")
opt_data_dir = get_cache_dir(pkg_info)
```

---

get_filepath *Retrieve the path to a single file from the package cache.*

Description

Retrieve the path to a single file from the package cache.

Usage

```
get_filepath(pkg_info, relative_filename, mustWork = TRUE)
```

Arguments

pkg_info	named list. Package identifier, see get_pkg_info() on how to get one.
relative_filename	string. A filename, relative to the package cache.
mustWork	logical. Whether an error should be created if the file does not exist.

Value

string. The path to the file. If mustWork=TRUE, the file is guaranteed to exist if the function returns (an error will occur if it does not). If mustWork=FALSE and the file does not exist, the empty string is returned.

Examples

```
pkg_info = get_pkg_info("mypackage")
full_path_of_file = get_filepath(pkg_info, "file1.txt", mustWork=FALSE)
```

get_pkg_info *Construct a pkg_info object to be used with all other functions.*

Description

This functions constructs an object that uniquely identifies your package, i.e., the package that want to use the package cache. This is not a secret.

Usage

```
get_pkg_info(packagename, author = NULL, version = NULL)
```

Arguments

packagename	string. The name of the package using the package cache. Must be a valid directory name. Should not contain spaces. Passed as 'appname' to 'rappdirs::user_data_dir'.
author	string. The author of the package using the package cache, or NULL. Must be a valid directory name if given, no need for the real author name. Should not contain spaces. Defaults to NULL. Passed as 'appauthor' to 'rappdirs::user_data_dir'. Leave at NULL if in doubt.
version	string or NULL. An optional version path element to append to the path. You might want to use this if you want multiple versions of your pacakge to be able to have independent data. If used, this would typically be "<major>.<minor>". Must be a valid directory name. Should not contain spaces or special characters.

Value

named list. This can be passed to all function which require a 'pkg_info' argument. You should not care for the inner structure and treat it as some identifier.

Examples

```
pkg_info = get_pkg_info("mypackage")
pkg_info = get_pkg_info("mypackage", author="me")
pkg_info = get_pkg_info("mypackage", author="me", version="0.3")
```

list_available *List files that are available locally in the package cache.*

Description

List files that are available locally in the package cache.

Usage

```
list_available(pkg_info)
```

Arguments

pkg_info	named list. Package identifier, see get_pkg_info() on how to get one.

Value

vector of strings. The file names available, relative to the package cache. The returned names may include a subdirectory part. The subdirectories are not listed separately.

Examples

```
pkg_info = get_pkg_info("mypackage")
available_files_in_cache = list_available(pkg_info)
```

---

remove_cached_files *Delete all the given files from the package cache.*

Description

Delete all the given files from the package cache.

Usage

```
remove_cached_files(pkg_info, relative_filenames)
```

Arguments

pkg_info	named list. Package identifier, see get_pkg_info() on how to get one.
relative_filenames	vector of strings. A vector of filenames, relative to the package cache.

Value

logical vector. For each file, whether it was deleted. Note that files which did not exist were not deleted! You should check the results using ‘files_available‘.

Examples

```
pkg_info = get_pkg_info("mypackage")
deleted = remove_cached_files(pkg_info, "some_file.txt")
```

Index