

Package ‘pkgfilecache’

September 1, 2026

Type Package
Title Download and Manage Optional Package Data
Version 0.4.1
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 Comprehensive R Archive Network (CRAN). Download functions are supposed to be called by users in interactive sessions only.
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), withr
Imports rappdirs, curl
VignetteBuilder knitr
Config/roxygen2/version 8.0.0
NeedsCompilation no
Author Tim Schäfer [aut, cre] (ORCID: <<https://orcid.org/0000-0002-3683-8070>>)
Repository CRAN
Date/Publication 2026-08-31 22:20:02 UTC

Contents

are_files_available	2
ensure_files_available	3
ensure_files_available_from_manifest	4
erase_file_cache	6
get_absolute_path_for_files	6

get_cache_dir	7
get_filepath	8
get_pkg_info	9
list_available	9
manifest_cli	10
manifest_script	11
read_manifest	11
remove_cached_files	12
write_manifest_from_dir	13
Index	15

are_files_available	<i>Check whether the given files exist in the package cache.</i>
---------------------	--

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,
  num_connections = getOption("pkgfilecache.num_connections", 2),
  num_retries = 2
)
```

Arguments

pkg_info	named list. Package identifier, see <code>get_pkg_info()</code> on how to get one.
relative_filenames	vector of strings. A vector of filenames, relative to the package cache.
urls	vector of strings. For each file, a remote URL where to download the file. Will be passed to <code>'curl::curl_download'</code> , 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 <code>'curl::curl_download'</code> docs for details.
md5sums	vector of strings or NULL. A list of MD5 checksums, one for each file in param <code>'relative_filenames'</code> , if not NULL. If given, the files will only be reported as existing if the MD5 sums match. A file that is present in the cache but whose MD5 sum does not match the expected sum is considered invalid and is deleted from the cache, so that it is never mistaken for a valid file.
on_errors	string. What to do if getting the files failed. One of <code>c("warn", "stop", "ignore")</code> . At the end, files are checked using <code>'files_available'</code> (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. This applies whether or not downloading was attempted, so missing files are never silently ignored, e.g., when <code>'download'</code> is FALSE.

download	logical. Whether to try downloading missing files. Defaults to TRUE. Existing files (with correct MD5 if available) will never be downloaded. When set to FALSE, no files are downloaded, but missing files are still reported according to 'on_errors'.
num_connections	integer. The number of parallel connections to use when downloading files. Defaults to 2. With more connections, several files are downloaded at the same time, which can speed up downloading many small files considerably. Use 1 for strictly sequential downloads. A high number of connections may overload the server or trigger rate limits. The default can be changed globally for all calls that do not specify this argument by setting the R option <code>pkgfilecache.num_connections</code> .
num_retries	integer. How many times to retry downloading files that failed, in addition to the first attempt. Defaults to 2. Each attempt uses fresh connections, which makes downloads robust against transient connection failures (e.g., a server that closes or throttles connections). Set to 0 to disable retrying.

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", "85ffec2e6efb476f1ee1e3e7ddd86de");
res = ensure_files_available(pkg_info, local_relative_filenames, urls, md5sums=md5sums);
erase_file_cache(pkg_info); # clear full cache
```

ensure_files_available_from_manifest

Ensure the files described in a manifest are available in the file cache.

Description

Like [ensure_files_available](#), but the files are described in a single declarative manifest (one row per file) instead of three parallel vectors. A manifest can be a CSV file (e.g., shipped with the package in `inst/extdata`) or a `data.frame`, see [read_manifest](#) for the expected format.

Usage

```
ensure_files_available_from_manifest(
  pkg_info,
  manifest,
  base_url = NULL,
  files_are_binary = NULL,
  on_errors = "warn",
  download = TRUE,
  num_connections = getOption("pkgfilecache.num_connections", 2),
  num_retries = 2
)
```

Arguments

<code>pkg_info</code>	named list. Package identifier, see <code>get_pkg_info()</code> on how to get one.
<code>manifest</code>	string or data.frame. The manifest to use, see read_manifest . Either the path to a CSV file, or a data.frame with the manifest columns.
<code>base_url</code>	string or NULL. Base URL to derive the download URL for files that have no explicit URL, see read_manifest . Ignored for files with an explicit url entry.
<code>files_are_binary</code>	logical vector. For each file, whether it is binary. Only required on Windows, when files need to be downloaded. See <code>curl::curl_download</code> docs for details. Ignored when download is FALSE.
<code>on_errors</code>	string. What to do if getting the files failed. One of <code>c("warn", "stop", "ignore")</code> . See ensure_files_available for details.
<code>download</code>	logical. Whether to try downloading missing files. Defaults to TRUE. Existing files (with correct MD5 if available) will never be downloaded. Set to FALSE to only check which files are available without downloading anything.
<code>num_connections</code>	integer. The number of parallel connections to use when downloading files, see ensure_files_available . Defaults to 2, or to the value of the R option <code>pkgfilecache.num_connections</code> if it is set.
<code>num_retries</code>	integer. How many times to retry downloading files that failed, in addition to the first attempt, see ensure_files_available . Defaults to 2.

Value

Named list, like for [ensure_files_available](#): the entries "available" and "missing" contain the manifest paths (as '/'-separated strings) that are available in, or missing from, the local file cache. The entry "file_status" is a logical vector in manifest row order indicating for each file whether it is available.

Examples

```
pkg_info = get_pkg_info("mypackage")
# A manifest as a data.frame: two files, URLs derived from base_url.
manifest = data.frame(path = c("sub/file1.txt", "file2.txt"),
```

```

        stringsAsFactors = FALSE)
# Only check availability, do not download anything (download = FALSE).
# The files are missing, so a warning is expected (see 'on_errors').
res = suppressWarnings(ensure_files_available_from_manifest(pkg_info, manifest,
  base_url = "https://example.com/data/",
  download = FALSE))

```

erase_file_cache	<i>Delete the full package cache directory for the given package.</i>
------------------	---

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	<i>Construct absolute path for package cache files.</i>
-----------------------------	---

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 directory for the given package.

By default, the cache is stored in the directory returned by `'tools::R_user_dir(packagename, "data")'` (for R version 4.0 or later), which is the location recommended by the CRAN repository policy for user-specific data and cache files. If a cache already exists at the legacy location used by older versions of this package (the directory returned by `'rappdirs::user_data_dir'`), that legacy directory is reused instead, so that existing downloads are not lost and do not have to be re-downloaded. On R versions before 4.0, the legacy `'rappdirs::user_data_dir'` location is used.

The location can be overridden with R options:

- `'pkgfilecache.cachedir'`: If set to a directory path, that directory is used as the root of the package cache, and the package name (and version, if any) are appended to it. Useful to place the cache elsewhere, e.g., on a ramdisk or network drive.
- `'pkgfilecache.use_tempdir'`: If set to `'TRUE'`, the cache is placed in a subdirectory of the R session's temporary directory (`'tempdir()'`). This is handy for unit tests and continuous integration, and in particular for tests that must not write to the user's home directory (e.g., on CRAN). The value of `'pkgfilecache.cachedir'` takes precedence if both options are set.

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 directory. A subdirectory of the directory returned by `'tools::R_user_dir'` (for R version 4.0 or later) or `'rappdirs::user_data_dir'` (for older R versions), unless one of the options `'pkgfilecache.cachedir'` or `'pkgfilecache.use_tempdir'` is set.

Examples

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

get_filepath	<i>Retrieve the path to a single file from the package cache.</i>
--------------	---

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 <code>get_pkg_info()</code> 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	<i>Construct a pkg_info object to be used with all other functions.</i>
--------------	---

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 package 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	<i>List files that are available locally in the package cache.</i>
----------------	--

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)
```

manifest_cli

Run the manifest generation command line interface (CLI).

Description

Generate a manifest CSV from a directory of files from the command line. This function is meant to be run with Rscript, not to be called interactively. It reads the command line arguments, parses them, and calls `write_manifest_from_dir`. It is the engine behind the standalone `make_manifest.R` script that ships with the package (see `manifest_script`).

Usage

```
manifest_cli(args = commandArgs(trailingOnly = TRUE))
```

Arguments

`args` character vector. The command line arguments. Defaults to the arguments actually passed to the R script (`commandArgs(trailingOnly = TRUE)`). You normally do not pass this; it exists to make the function testable.

Details

Supported arguments (each flag can be given as `--flag value` or as `--flag=value`):

- `--dir <path>`: The directory containing the data files (required).
- `--out <path>`: The path of the CSV file to write the manifest to (required).
- `--url-base <url>`: Optional base URL used to derive the download URL of every file, as `paste0(url_base, path)`.
- `--help`: Print usage information and exit without generating anything.

A leading `--args` or `--` marker (used by Rscript to separate command line arguments from R options) is ignored, so both the one-liner form and the script form shown in the examples work.

Value

The generated manifest as a data.frame, invisibly, or NULL if --help was given.

Examples

```
## Not run:
# One-liner from the shell (no script file needed):
Rscript -e 'pkgfilecache::manifest_cli()' --args --dir ~/mydata --out files.csv

# Same thing using the standalone script shipped with the package
# (--url-base is optional and derives the download URLs):
Rscript "$(Rscript -e 'cat(pkgfilecache::manifest_script())') --dir ~/mydata --out files.csv

## End(Not run)
```

manifest_script	<i>Get the path of the standalone manifest generation script.</i>
-----------------	---

Description

Return the full path to the `make_manifest.R` script that ships with the package (installed into the package's `exec` subdirectory). This script is a thin wrapper around [manifest_cli](#) and can be run with `Rscript` to generate a manifest from the command line. On Unix-like systems you can also copy it to a directory on your `PATH` and make it executable with `chmod +x`, then run it like a normal command.

Usage

```
manifest_script()
```

Value

character string. The full path to the script, or `""` if the script cannot be found (e.g., because the package is not installed correctly).

read_manifest	<i>Read a file manifest for optional data files.</i>
---------------	--

Description

Read a manifest that describes a set of optional data files, one file per row. A manifest is either a CSV file or a data.frame with the columns described below. Manifests can be written by hand or generated with [write_manifest_from_dir](#).

Usage

```
read_manifest(manifest, base_url = NULL)
```

Arguments

<code>manifest</code>	string or data.frame. The manifest to read: either the path to a CSV file, or a data.frame that is used directly. Comment lines in a CSV manifest that start with # are ignored.
<code>base_url</code>	string or NULL. A base URL used to derive the download URL for all files that do not have an explicit URL. For such files, the URL is set to <code>paste0(base_url, path)</code> . This is convenient when the remote directory layout mirrors the layout inside the package cache. Files with an explicit <code>url</code> entry are never modified.

Details

The manifest must contain a column named `path`, giving the file path relative to the package cache directory, using `/` as separator. Two further columns are optional:

- `url`: The remote URL to download the file from. If empty (or missing), it is derived from `base_url` (see above).
- `md5`: The MD5 checksum of the file, used to verify file integrity. If empty (or missing), no integrity check is performed for that file (only its existence is checked).

Additional columns are allowed and ignored.

Value

data.frame with the columns `path`, `url` and `md5`. The `url` column never contains missing values after this function ran: either it was present in the input, or it was derived from `base_url` and `path`. The `md5` column may contain NA for files without a checksum.

<code>remove_cached_files</code>	<i>Delete all the given files from the package cache.</i>
----------------------------------	---

Description

Delete all the given files from the package cache.

Usage

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

Arguments

<code>pkg_info</code>	named list. Package identifier, see <code>get_pkg_info()</code> on how to get one.
<code>relative_filenames</code>	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")
```

```
write_manifest_from_dir
```

Generate a file manifest from a directory of local files.

Description

Create a manifest (CSV file) that describes all files in the given directory. This is meant to remove the tedious manual work of adding many files to a package: put the files into a directory (subdirectories are preserved), run this function once, and a ready-to-use manifest is written. The MD5 checksum of every file is computed automatically.

Usage

```
write_manifest_from_dir(dir, out, url_base = NULL)
```

Arguments

dir	string. The directory containing the local files to describe. Files in subdirectories are included, and their relative paths (using '/' as separator) become the path entries in the manifest. Hidden files (starting with a dot) are not included.
out	string. The path of the CSV file to write the manifest to. Typically something like inst/extdata/files.csv in your package. The file is overwritten if it exists. Comment lines describing the manifest are written to the top of the file.
url_base	string or NULL. If given, the url column is filled with paste0(url_base, path) for every file, i.e., the remote URLs are derived from the paths. If NULL (the default), the url column is left empty, and the URLs have to be derived at read time by passing base_url to read_manifest or ensure_files_available_from_manifest (or by editing the file).

Value

The generated manifest as a data.frame, invisibly.

Examples

```
## Not run:
manifest = write_manifest_from_dir("~/mydata", "inst/extdata/files.csv",
                                   url_base = "https://example.com/data/")

## End(Not run)
```

Index

are_files_available, [2](#)

ensure_files_available, [3](#), [4](#), [5](#)
ensure_files_available_from_manifest,
[4](#), [13](#)
erase_file_cache, [6](#)

get_absolute_path_for_files, [6](#)
get_cache_dir, [7](#)
get_filepath, [8](#)
get_pkg_info, [9](#)

list_available, [9](#)

manifest_cli, [10](#), [11](#)
manifest_script, [10](#), [11](#)

read_manifest, [4](#), [5](#), [11](#), [13](#)
remove_cached_files, [12](#)

write_manifest_from_dir, [10](#), [11](#), [13](#)