Package: cachem (via r-universe)

October 5, 2026

Version 1.1.0

Title Cache R Objects with Automatic Pruning

Description Key-value stores with automatic pruning. Caches can limit either their total size or the age of the oldest object (or both), automatically pruning objects to maintain the constraints.

License MIT + file LICENSE

Encoding UTF-8

ByteCompile true

URL https://cachem.r-lib.org/, https://github.com/r-lib/cachem

Imports rlang, fastmap (>= 1.2.0)

Suggests testthat

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Config/Needs/routine lobstr

Config/Needs/website pkgdown

Repository https://posit-dev-shinycoreci.r-universe.dev

Date/Publication 2025-10-20 21:02:38 UTC

RemoteUrl https://github.com/r-lib/cachem

RemoteRef HEAD

RemoteSha b24174dd4dfce482d1317f9222ae4c738147307b

Contents

cache_disk 2
cache_layered 5
cache_mem 6

Index **9**

cache_disk *Create a disk cache object*

Description

A disk cache object is a key-value store that saves the values as files in a directory on disk. Objects can be stored and retrieved using the `get()` and `set()` methods. Objects are automatically pruned from the cache according to the parameters `max_size`, `max_age`, `max_n`, and `evict`.

Usage

```
cache_disk(
  dir = NULL,
  max_size = 1024 * 1024^2,
  max_age = Inf,
  max_n = Inf,
  evict = c("lru", "fifo"),
  destroy_on_finalize = FALSE,
  read_fn = NULL,
  write_fn = NULL,
  extension = ".rds",
  missing = key_missing(),
  prune_rate = 20,
  warn_ref_objects = FALSE,
  logfile = NULL
)
```

Arguments

`dir`	Directory to store files for the cache. If `NULL` (the default) it will create and use a temporary directory.
`max_size`	Maximum size of the cache, in bytes. If the cache exceeds this size, cached objects will be removed according to the value of the `evict`. Use `Inf` for no size limit. The default is 1 gigabyte.
`max_age`	Maximum age of files in cache before they are evicted, in seconds. Use `Inf` for no age limit.
`max_n`	Maximum number of objects in the cache. If the number of objects exceeds this value, then cached objects will be removed according to the value of `evict`. Use `Inf` for no limit of number of items.
`evict`	The eviction policy to use to decide which objects are removed when a cache pruning occurs. Currently, `"lru"` and `"fifo"` are supported.
`destroy_on_finalize`	If `TRUE`, then when the cache_disk object is garbage collected, the cache directory and all objects inside of it will be deleted from disk. If `FALSE` (the default), it will do nothing when finalized.

read_fn	The function used to read the values from disk. If NULL (the default) it will use readRDS.
write_fn	The function used to write the values from disk. If NULL (the default) it will use writeRDS.
extension	The file extension to use for files on disk.
missing	A value to return when get(key) is called but the key is not present in the cache. The default is a key_missing() object. It is actually an expression that is evaluated each time there is a cache miss. See section Missing keys for more information.
prune_rate	How often to prune the cache. See section Cache Pruning for more information.
warn_ref_objects	Should a warning be emitted when a reference is stored in the cache? This can be useful because serializing and deserializing a reference object (such as environments and external pointers) can lead to unexpected behavior.
logfile	An optional filename or connection object to where logging information will be written. To log to the console, use stderr() or stdout().

Value

A disk caching object, with class cache_disk.

Missing keys

The missing parameter controls what happens when get() is called with a key that is not in the cache (a cache miss). The default behavior is to return a key_missing() object. This is a *sentinel value* that indicates that the key was not present in the cache. You can test if the returned value represents a missing key by using the is.key_missing() function. You can also have get() return a different sentinel value, like NULL. If you want to throw an error on a cache miss, you can do so by providing an expression for missing, as in missing = stop("Missing key").

When the cache is created, you can supply a value for missing, which sets the default value to be returned for missing values. It can also be overridden when get() is called, by supplying a missing argument. For example, if you use cache$get("mykey", missing = NULL), it will return NULL if the key is not in the cache.

The missing parameter is actually an expression which is evaluated each time there is a cache miss. A quosure (from the rlang package) can be used.

If you use this, the code that calls get() should be wrapped with tryCatch() to gracefully handle missing keys.

Cache pruning

Cache pruning occurs when set() is called, or it can be invoked manually by calling prune().

The disk cache will throttle the pruning so that it does not happen on every call to set(), because the filesystem operations for checking the status of files can be slow. Instead, it will prune once in every prune_rate calls to set(), or if at least 5 seconds have elapsed since the last prune occurred, whichever is first.

When a pruning occurs, if there are any objects that are older than max_age, they will be removed.

The max_size and max_n parameters are applied to the cache as a whole, in contrast to max_age, which is applied to each object individually.

If the number of objects in the cache exceeds max_n, then objects will be removed from the cache according to the eviction policy, which is set with the evict parameter. Objects will be removed so that the number of items is max_n.

If the size of the objects in the cache exceeds max_size, then objects will be removed from the cache. Objects will be removed from the cache so that the total size remains under max_size. Note that the size is calculated using the size of the files, not the size of disk space used by the files — these two values can differ because of files are stored in blocks on disk. For example, if the block size is 4096 bytes, then a file that is one byte in size will take 4096 bytes on disk.

Another time that objects can be removed from the cache is when get() is called. If the target object is older than max_age, it will be removed and the cache will report it as a missing value.

Eviction policies

If max_n or max_size are used, then objects will be removed from the cache according to an eviction policy. The available eviction policies are:

"lru" Least Recently Used. The least recently used objects will be removed. This uses the filesystem's mtime property. When "lru" is used, each get() is called, it will update the file's mtime using Sys.setFileTime(). Note that on some platforms, the resolution of Sys.setFileTime() may be low, one or two seconds.

"fifo" First-in-first-out. The oldest objects will be removed.

Both of these policies use files' mtime. Note that some filesystems (notably FAT) have poor mtime resolution. (atime is not used because support for atime is worse than mtime.)

Sharing among multiple processes

The directory for a cache_disk can be shared among multiple R processes. To do this, each R process should have a cache_disk object that uses the same directory. Each cache_disk will do pruning independently of the others, so if they have different pruning parameters, then one cache_disk may remove cached objects before another cache_disk would do so.

Even though it is possible for multiple processes to share a cache_disk directory, this should not be done on networked file systems, because of slow performance of networked file systems can cause problems. If you need a high-performance shared cache, you can use one built on a database like Redis, SQLite, mySQL, or similar.

When multiple processes share a cache directory, there are some potential race conditions. For example, if your code calls exists(key) to check if an object is in the cache, and then call get(key), the object may be removed from the cache in between those two calls, and get(key) will throw an error. Instead of calling the two functions, it is better to simply call get(key), and check that the returned object is not a key_missing() object, using is.key_missing(). This effectively tests for existence and gets the object in one operation.

It is also possible for one processes to prune objects at the same time that another processes is trying to prune objects. If this happens, you may see a warning from file.remove() failing to remove a file that has already been deleted.

Methods

A disk cache object has the following methods:

`get(key, missing)` Returns the value associated with `key`. If the key is not in the cache, then it evaluates the expression specified by `missing` and returns the value. If `missing` is specified here, then it will override the default that was set when the `cache_mem` object was created. See section Missing Keys for more information.

`set(key, value)` Stores the `key-value` pair in the cache.

`exists(key)` Returns `TRUE` if the cache contains the key, otherwise `FALSE`.

`remove(key)` Removes `key` from the cache, if it exists in the cache. If the key is not in the cache, this does nothing.

`size()` Returns the number of items currently in the cache.

`keys()` Returns a character vector of all keys currently in the cache.

`reset()` Clears all objects from the cache.

`destroy()` Clears all objects in the cache, and removes the cache directory from disk.

`prune()` Prunes the cache, using the parameters specified by `max_size`, `max_age`, `max_n`, and `evict`.

---

cache_layered *Compose any number of cache objects into a new, layered cache object*

Description

Note that `cache_layered` is currently experimental.

Usage

```
cache_layered(..., logfile = NULL)
```

Arguments

`...`	Cache objects to compose into a new, layered cache object.
`logfile`	An optional filename or connection object to where logging information will be written. To log to the console, use `stderr()` or `stdout()`.

Value

A layered caching object, with class `cache_layered`.

Examples

```
# Make a layered cache from a small memory cache and large disk cache
m <- cache_mem(max_size = 100 * 1024^2)
d <- cache_disk(max_size = 2 * 1024^3)
cl <- cache_layered(m, d)
```

cache_mem *Create a memory cache object*

Description

A memory cache object is a key-value store that saves the values in an environment. Objects can be stored and retrieved using the `get()` and `set()` methods. Objects are automatically pruned from the cache according to the parameters `max_size`, `max_age`, `max_n`, and `evict`.

Usage

```
cache_mem(
  max_size = 512 * 1024^2,
  max_age = Inf,
  max_n = Inf,
  evict = c("lru", "fifo"),
  missing = key_missing(),
  logfile = NULL
)
```

Arguments

`max_size`	Maximum size of the cache, in bytes. If the cache exceeds this size, cached objects will be removed according to the value of the `evict`. Use `Inf` for no size limit. The default is 512 megabytes.
`max_age`	Maximum age of files in cache before they are evicted, in seconds. Use `Inf` for no age limit.
`max_n`	Maximum number of objects in the cache. If the number of objects exceeds this value, then cached objects will be removed according to the value of `evict`. Use `Inf` for no limit of number of items.
`evict`	The eviction policy to use to decide which objects are removed when a cache pruning occurs. Currently, `"lru"` and `"fifo"` are supported.
`missing`	A value to return when `get(key)` is called but the key is not present in the cache. The default is a `key_missing()` object. It is actually an expression that is evaluated each time there is a cache miss. See section Missing keys for more information.
`logfile`	An optional filename or connection object to where logging information will be written. To log to the console, use `stderr()` or `stdout()`.

Details

In a `cache_mem`, R objects are stored directly in the cache; they are not *not* serialized before being stored in the cache. This contrasts with other cache types, like `cache_disk()`, where objects are serialized, and the serialized object is cached. This can result in some differences of behavior. For example, as long as an object is stored in a cache_mem, it will not be garbage collected.

Value

A memory caching object, with class `cache_mem`.

Missing keys

The `missing` parameter controls what happens when `get()` is called with a key that is not in the cache (a cache miss). The default behavior is to return a `key_missing()` object. This is a *sentinel value* that indicates that the key was not present in the cache. You can test if the returned value represents a missing key by using the `is.key_missing()` function. You can also have `get()` return a different sentinel value, like `NULL`. If you want to throw an error on a cache miss, you can do so by providing an expression for `missing`, as in `missing = stop("Missing key")`.

When the cache is created, you can supply a value for `missing`, which sets the default value to be returned for missing values. It can also be overridden when `get()` is called, by supplying a `missing` argument. For example, if you use `cache$get("mykey", missing = NULL)`, it will return `NULL` if the key is not in the cache.

The `missing` parameter is actually an expression which is evaluated each time there is a cache miss. A quosure (from the rlang package) can be used.

If you use this, the code that calls `get()` should be wrapped with `tryCatch()` to gracefully handle missing keys.

@section Cache pruning:

Cache pruning occurs when `set()` is called, or it can be invoked manually by calling `prune()`.

When a pruning occurs, if there are any objects that are older than `max_age`, they will be removed.

The `max_size` and `max_n` parameters are applied to the cache as a whole, in contrast to `max_age`, which is applied to each object individually.

If the number of objects in the cache exceeds `max_n`, then objects will be removed from the cache according to the eviction policy, which is set with the `evict` parameter. Objects will be removed so that the number of items is `max_n`.

If the size of the objects in the cache exceeds `max_size`, then objects will be removed from the cache. Objects will be removed from the cache so that the total size remains under `max_size`.

Another time that objects can be removed from the cache is when `get()` is called. If the target object is older than `max_age`, it will be removed and the cache will report it as a missing value.

Eviction policies

If `max_n` or `max_size` are used, then objects will be removed from the cache according to an eviction policy. The available eviction policies are:

`"lru"` Least Recently Used. The least recently used objects will be removed.

`"fifo"` First-in-first-out. The oldest objects will be removed.

Methods

A disk cache object has the following methods:

`get(key, missing)` Returns the value associated with `key`. If the key is not in the cache, then it evaluates the expression specified by `missing` and returns the value. If `missing` is specified here, then it will override the default that was set when the `cache_mem` object was created. See section Missing Keys for more information.

`set(key, value)` Stores the `key`-`value` pair in the cache.

`exists(key)` Returns `TRUE` if the cache contains the key, otherwise `FALSE`.

`remove(key)` Removes `key` from the cache, if it exists in the cache. If the key is not in the cache, this does nothing.

`size()` Returns the number of items currently in the cache.

`keys()` Returns a character vector of all keys currently in the cache.

`reset()` Clears all objects from the cache.

`destroy()` Clears all objects in the cache, and removes the cache directory from disk.

`prune()` Prunes the cache, using the parameters specified by `max_size`, `max_age`, `max_n`, and `evict`.

Index

cache_disk, 2
cache_disk(), 6
cache_layered, 5
cache_mem, 6

is.key_missing(), 3, 7

key_missing(), 3, 6, 7

Sys.setFileTime(), 4

tryCatch(), 3, 7