

# Package ‘hashmapR’

November 9, 2025

**Title** Fast, Vectorized Hashmap

**Version** 1.0.1

**Description** A fast, vectorized hashmap that is built on top of 'C++' `std::unordered_map` <[https://en.cppreference.com/w/cpp/container/unordered\\_map](https://en.cppreference.com/w/cpp/container/unordered_map)>.

The map can hold any 'R' object as key / value as long as it is serializable and supports vectorized insertion, lookup, and deletion.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**URL** <https://github.com/svensglinz/hashmapR>

**Suggests** testthat (>= 3.0.0)

**NeedsCompilation** yes

**Author** Sven Glinz [aut, cre]

**Maintainer** Sven Glinz <svenglinz@proton.me>

**Repository** CRAN

**Date/Publication** 2025-11-09 10:00:13 UTC

## Contents

hashmap . . . . .	1
<b>Index</b>	<b>5</b>

---

hashmap	<i>Create a Hashmap Object</i>
---------	--------------------------------

---

## Description

A hashmap provides a key-value store where keys and values can be any R object that is serializable. Internally, it uses a C++ `std::unordered_map` for efficient storage and lookup.

**Usage**

```
hashmap()
```

**Details****Serialization:**

External pointers cannot be directly serialized in R. To save or transfer a hashmap, you can convert it to a list using `map$to_list()`, which returns a standard R list representation. You can restore a hashmap from such a list using `map$from_list(...)`.

**Key Equality:**

Keys are considered equal if `identical(k1, k2) == TRUE`. This strict equality means that numeric objects like 1L (integer) and 1 (double) are not considered equal.

*Memory:*

As R uses copy-on-write, the hashmap does not make a copy of keys and values that are inserted but only stores a reference to them. Thus, operations such as cloning a hashmap or inserting existing objects into a hashmap come at very little memory overhead. If any data structure is inserted that does not follow copy-on-write semantics (unlikely), it is up to you to copy the object before insertion should you wish for the hashmap to take ownership of the data.

The same applies for all data returned from the hashmap (eg. through `$keys()`, `$values()`, `$get()`). If any returned element does not follow copy-on-write semantics, changes to the data will result in changes to the data in the hashmap.

**Methods:**

A hashmap object exposes the following methods:

`map$set(key, value, replace = FALSE, vectorize = FALSE)` Set the value associated with a key in the hashmap.

If `replace=TRUE`, the value will replace the old value in case the map already contains an element with the same key.

If `vectorize = TRUE`, both key and value must be lists of the same length. Each element of the key list is paired with the corresponding element of the value list, and all pairs are inserted into the hashmap.

`map$get(key, vectorize = FALSE)` Retrieve the value associated with a key. Returns NULL if the element does not exist

If `vectorize=TRUE`, key must be a list and each element of the key list will be looked up separately. The result is returned as a list.

`map$delete(key, vectorize = FALSE)` Remove a key-value pair from the map.

If `vectorize=TRUE`, key must be a list. Each element in the list will be deleted separately.

`map$contains(key, vectorize=FALSE)` Check if a key exists in the map. Returns TRUE/FALSE.

If `vectorize=TRUE`, key must be a list. Existence will be checked for each item in the list.

`map$keys()` Returns a list of all keys.

`map$values()` Returns a list of all values.

`map$to_list()` Serialize the hashmap to a standard R list. Returns a list with two sublists (keys, values)

`map$from_list(lst)` Restore hashmap contents from a list. Returns the restored hashmap

`map$clone()` Returns a duplicate of the map

`map$invert(duplicates=c("stack", "first"))` Invert the hashmap by swapping keys and values. Returns a new hashmap where the original values become keys and the original keys become values.

The `duplicates` parameter controls how to handle cases where multiple keys map to the same value:

- "first": Keep only the first key encountered for each value
- "stack": Store all keys that map to the same value as a list

## Value

A list of class `hashmap` which contains an external pointer object (the hashmap) and methods to access and store elements in the map.

## Examples

```
# create hashmap object
library(hashmapR)
m <- hashmap()

# insert key, value pair into hashmap
# any serializable R value can be used as a key and value
m$set("key", "value")
m$set(1, 2)
m$set(mtcars, Sys.Date())

#' # alternative to $set()
m["key"] <- "value"

# insert key, value pairs (vectorized)
# if vectorized is not set to TRUE, the list itself is inserted as a single key / value
m$set(list(1, 2, 3), list("one", "two", "three"), vectorize = TRUE)

# retrieve values
m$get(1)
m$get(list(1, 2, 3), vectorize = TRUE)

#' # alternatively (for single lookups)
m[1]

# remove values
m$remove(1)
m$remove(mtcars)

# remove values (vectorized)
m$remove(list(1, 2, 3), vectorize = TRUE)

# return number of items in map
m$size()

# return keys as a list
m$keys()
```

```
# return values as a list
m$values()

# clear map
m$clear()

# duplicate map
m$clone()

# invert map (stack duplicates)
m$invert(duplicates = "stack")
```

# Index

hashmap, 1