Package 'hashtable'

August 20, 2026

Type Package

Title Hash Table and Hash Set

Version 1.0.1

Date 2026-8-8

Depends R (>= 4.0.0)

Imports methods, Rcpp, fastmatch

LinkingTo Rcpp

Suggests knitr, rmarkdown

Description It provides three implementations of hash tables and hash maps: 1. using 'std::unordered_map' and 'std::unordered_set' C++ libraries, 2. wrapping around the 'fastmatch' package, 3. using R environment.

URL https://github.com/jokergoo/hashtable, https://jokergoo.github.io/hashtable/

VignetteBuilder knitr

BugReports https://github.com/jokergoo/hashtable/issues

License MIT + file LICENSE

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Zuguang Gu [aut, cre] (ORCID: <https://orcid.org/0000-0002-7395-8709>)

Maintainer Zuguang Gu <guzuguang@suat-sz.edu.cn>

Repository CRAN

Date/Publication 2026-08-20 11:10:09 UTC

Contents

`generic` *Generic functions*

Description

Generic functions

Details

There are the following generic functions:

- `hash_exists(h, keys)`: tests whether keys exist.
- `hash_keys(h)`: returns a vector of keys.
- `hash_values(h, keys = NULL)`: returns a vector of values.
- `hash_size(h)`: returns the size of the hash table or hash set.
- `hash_insert(h, keys, ...)`: inserts new keys or modify values for existing keys.
- `hash_delete(h, keys)`: deletes keys.
- `hash_copy(h)`: makes a copy of the hash table or hash set.

They can be used on the hash object created by `hash_table()`, `hash_set()`, `hash_fm_table()`, `hash_fm_set()`, `hash_env_table()`, `hash_env_set()`.

Value

`hash_exists()` returns a logical vector. `hash_keys()` returns a character vector. `hash_values()` returns a single vector or a list. `hash_size()` returns an integer scalar. `hash_insert()`, `hash_delete()` and `hash_copy()` all return a hash table object.

`hash_env_table` *Hash table and hash set implemented by environment*

Description

Hash table and hash set implemented by environment

Usage

```
hash_env_table(keys, values)

hash_env_set(keys)

## S4 method for signature 'hash_env'
hash_exists(h, keys)

## S4 method for signature 'hash_env'
hash_delete(h, keys)

## S4 method for signature 'hash_env_table'
hash_insert(h, keys, values)

## S4 method for signature 'hash_env_set'
hash_insert(h, keys)

## S4 method for signature 'hash_env'
hash_size(h)

## S3 method for class 'hash_env'
length(x)

## S4 method for signature 'hash_env'
hash_keys(h)

## S4 method for signature 'hash_env_table'
hash_values(h, keys = NULL)

## S4 method for signature 'hash_env_set'
hash_values(h, keys = NULL)

## S4 method for signature 'hash_env'
hash_copy(h)

## S3 method for class 'hash_env_table'
x[[i]]

## S3 method for class 'hash_env_set'
x[[i]]

## S3 replacement method for class 'hash_env_table'
x[[i]] <- value

## S3 method for class 'hash_env_table'
x[i]

## S3 method for class 'hash_env_set'
```

```
x[i]

## S3 replacement method for class 'hash_env_table'
x[i] <- value

## S3 method for class 'hash_env'
x$name

## S3 replacement method for class 'hash_env_table'
x$i <- value

## S3 replacement method for class 'hash_env_set'
x[[i]] <- value

## S3 replacement method for class 'hash_env_set'
x[i] <- value

## S3 replacement method for class 'hash_env_set'
x$name <- value

as.hash_env(x)

## Default S3 method:
as.hash_env(x)

## S3 method for class 'hash_env_set'
as.vector(x, mode = "any")

## S3 method for class 'hash_env_table'
as.vector(x, mode = "any")

## S3 method for class 'hash_env_table'
as.list(x, ...)

## S4 method for signature 'hash_env_table'
show(object)

## S4 method for signature 'hash_env_set'
show(object)
```

Arguments

keys	A character vector. Keys should have no duplicates.
values	An atomic vector or a list.
h, x, object	A hash_env_table returned by hash_env_table() or a hash_env_set object returned by hash_env_set(). For as.hash_env(), x should be a named (general) vector which will be converted to a hash table, or a character vector which will be converted to a hash set.

i, name, value	Keys and values.
mode, ...	Please ignore.

Details

hash_values() and [methods on the hash_env_table object preserve the original format of values, which means, if values was specified as an atomic vector, the two functions also returns atomic vectors.

Value

hash_env_table() returns a hash_env_table object. hash_env_set() returns a hash_env_set object. hash_env_delete(), hash_env_insert(), hash_env_copy() return a hash_env_table object or a hash_env_set object. hash_exists() returns a logical vector. hash_size() returns an integer. hash_keys() returns a character vector. hash_values() on the hash_env_table object returns a vector of a list which has the same format as in the constructor function. hash_values() on the hash_env_set object throws an error.

Examples

```
h = hash_env_table(letters, 1:26)
hash_keys(h)
hash_values(h)
h$a

h = hash_env_set(letters)
hash_exists(h, "a")
```

---

hash_fm_table *Hash table and hash set implemented by fastmatch*

Description

Hash table and hash set implemented by fastmatch

Usage

```
hash_fm_table(keys, values)

hash_fm_set(keys)

## S4 method for signature 'hash_fm'
hash_exists(h, keys)

## S4 method for signature 'hash_fm'
hash_delete(h, keys)

## S4 method for signature 'hash_fm'
```

```
hash_size(h)

## S3 method for class 'hash_fm'
length(x)

## S4 method for signature 'hash_fm'
hash_keys(h)

## S4 method for signature 'hash_fm_table'
hash_values(h, keys = NULL)

## S4 method for signature 'hash_fm_set'
hash_values(h, keys = NULL)

## S4 method for signature 'hash_fm_table'
hash_insert(h, keys, values)

## S4 method for signature 'hash_fm_set'
hash_insert(h, keys, values)

## S4 method for signature 'hash_fm'
hash_copy(h)

## S3 method for class 'hash_fm_table'
x[[i]]

## S3 method for class 'hash_fm_set'
x[[i]]

## S3 replacement method for class 'hash_fm_table'
x[[i]] <- value

## S3 method for class 'hash_fm_table'
x[i]

## S3 method for class 'hash_fm_set'
x[i]

## S3 replacement method for class 'hash_fm_table'
x[i] <- value

## S3 method for class 'hash_fm'
x$name

## S3 replacement method for class 'hash_fm_table'
x$i <- value

## S3 replacement method for class 'hash_fm_set'
```

```
x[[i]] <- value

## S3 replacement method for class 'hash_fm_set'
x[i] <- value

## S3 replacement method for class 'hash_fm_set'
x$name <- value

as.hash_fm(x)

## Default S3 method:
as.hash_fm(x)

## S3 method for class 'hash_fm_set'
as.vector(x, mode = "any")

## S3 method for class 'hash_fm_table'
as.vector(x, mode = "any")

## S3 method for class 'hash_fm_table'
as.list(x, ...)

## S4 method for signature 'hash_fm_table'
show(object)

## S4 method for signature 'hash_fm_set'
show(object)
```

Arguments

keys	A character vector. Keys should have no duplicates.
values	An atomic vector or a list.
h, x, object	A hash_fm_table object returned by hash_fm_table() or a hash_fm_set object returned by hash_fm_set(). For as.hash_fm(), x should be a named (general) vector which will be converted to a hash table, or a character vector which will be converted to a hash set.
i, name, value	Keys and values.
mode, ...	Please ignore.

Details

hash_values() and [methods on the hash_fm_table object preserve the original format of values, which means, if values was specified as an atomic vector, the two functions also returns atomic vectors.

Once the hash table or the hash set is created, it is not allowed to insert or delete keys.

Value

hash_fm_table() returns a hash_fm_table object. hash_fm_set() returns a hash_fm_set object. hash_fm_delete(), hash_fm_insert(), hash_fm_copy() return a hash_fm_table object or a hash_fm_set object. hash_exists() returns a logical vector. hash_size() returns an integer. hash_keys() returns a character vector. hash_values() on the hash_fm_table object returns a vector of a list which has the same format as in the constructor function. hash_values() on the hash_fm_set object throws an error.

Examples

```
h = hash_fm_table(letters, 1:26)
hash_keys(h)
hash_values(h)
h$a
try(h$a <- 2L)

h = hash_fm_set(letters)
hash_exists(h, "a")
```

---

hash_set *Hash set implemented by std::unordered_set*

Description

Hash set implemented by std::unordered_set

Usage

```
hash_set(keys)

## S4 method for signature 'hash_unordered_set'
hash_exists(h, keys)

## S4 method for signature 'hash_unordered_set'
hash_insert(h, keys)

## S4 method for signature 'hash_unordered_set'
hash_delete(h, keys)

## S4 method for signature 'hash_unordered_set'
hash_size(h)

## S4 method for signature 'hash_unordered_set'
hash_copy(h)

## S4 method for signature 'hash_unordered_set'
hash_keys(h)
```

```
## S4 method for signature 'hash_unordered_set'
hash_values(h, keys = NULL)

## S3 method for class 'hash_unordered_set'
x[[i]]

## S3 method for class 'hash_unordered_set'
x[i]

## S3 method for class 'hash_unordered_set'
x$name

## S3 replacement method for class 'hash_unordered_set'
x[[i]] <- value

## S3 replacement method for class 'hash_unordered_set'
x[i] <- value

## S3 replacement method for class 'hash_unordered_set'
x$name <- value

## S3 method for class 'hash_unordered_set'
length(x)

## S4 method for signature 'hash_unordered_set'
show(object)

as.hash_set(x)

## Default S3 method:
as.hash_set(x)

## S3 method for class 'hash_unordered_set'
as.vector(x, mode = "any")
```

Arguments

keys	A character vector. Keys should have no duplicates.
h, x, object	A hash_unordered_set object returned by hash_set(). For as.hash_set(), x should be a character vector which will be converted to a hash set.
i, name, value	Keys and values.
mode	Please ignore.

Details

Hash set has no values associated.

$, [[and [return logical vectors. $<-, [[<- and [<- insert or delete keys.

Value

`hash_set()`, `hash_insert()`, `hash_delete()`, `hash_copy()` returns a `hash_unordered_set` object. `hash_exists()` returns a logical vector. `hash_size()` returns an integer. `hash_keys()` returns a character vector. `hash_values()` throws an error.

Examples

```
h = hash_set(letters)
hash_exists(h, c("a", "b", "foo"))
hash_insert(h, "foo")
hash_delete(h, "foo")
h$a
h$foo

as.hash_set(letters)
```

---

hash_table *Hash table implemented by std::unordered_map*

Description

Hash table implemented by std::unordered_map

Usage

```
hash_table(keys, values)

## S4 method for signature 'hash_unordered_map'
hash_values(h, keys = NULL)

## S4 method for signature 'hash_unordered_map'
hash_exists(h, keys)

## S4 method for signature 'hash_unordered_map'
hash_insert(h, keys, values)

## S4 method for signature 'hash_unordered_map'
hash_delete(h, keys)

## S4 method for signature 'hash_unordered_map'
hash_size(h)

## S3 method for class 'hash_unordered_map'
length(x)

## S4 method for signature 'hash_unordered_map'
hash_keys(h)
```

```
## S4 method for signature 'hash_unordered_map'
hash_copy(h)

## S3 method for class 'hash_unordered_map'
x[[i]]

## S3 method for class 'hash_unordered_map'
x[i]

## S3 method for class 'hash_unordered_map'
x$name

## S3 replacement method for class 'hash_unordered_map'
x[[i]] <- value

## S3 replacement method for class 'hash_unordered_map'
x[i] <- value

## S3 replacement method for class 'hash_unordered_map'
x$name <- value

## S4 method for signature 'hash_unordered_map'
show(object)

as.hash_table(x)

## Default S3 method:
as.hash_table(x)

## S3 method for class 'hash_unordered_map'
as.vector(x, mode = "any")

## S3 method for class 'hash_unordered_map'
as.list(x, ...)
```

Arguments

keys	A character vector. Keys should have no duplicates.
values	An atomic vector or a list.
h, x, object	A hash_unordered_map object returned by hash_table(). For as.hash_table(), x should be a named (general) vector which will be converted to a hash table.
i, name, value	Keys and values.
mode, ...	Please ignore.

Details

hash_values() and [methods preserve the original format of values, which means, if values

was specified as an atomic vector, the two functions also returns atomic vectors.

Value

`hash_table()`, `hash_insert()`, `hash_delete()`, `hash_copy()` returns a `hash_unordered_map` object. `hash_exists()` returns a logical vector. `hash_size()` returns an integer. `hash_keys()` returns a character vector. `hash_values()` returns a vector of a list which has the same format as in the constructor function.

Examples

```
hash_table(c("a", "b"), 1:2L)
hash_table(c("a", "b"), c(TRUE, FALSE))
hash_table(c("a", "b"), c(0.1, 0.2))
hash_table(c("a", "b"), c("one", "two"))
hash_table(c("a", "b"), as.Date(c("2025-01-01", "2025-02-01")))
hash_table(c("a", "b"), as.POSIXct(c("2025-01-01 00:00:01", "2025-02-01 00:00:01")))
hash_table(c("a", "b"), list(1:10, letters))

h = hash_table(letters, 1:26)
hash_keys(h)
hash_values(h)
hash_exists(h, c("a", "b", "foo"))
hash_delete(h, letters[1:20]); h
hash_insert(h, "foo", 100L)

h = hash_table(letters, 1:26)
h$a
h[["a"]]
h[c("a", "b")]

as.vector(h)
as.list(h)
```

Index