Package: fastmap (via r-universe)

October 22, 2026

Title Fast Data Structures

Version 1.2.0

Description Fast implementation of data structures, including a key-value store, stack, and queue. Environments are commonly used as key-value stores in R, but every time a new key is used, it is added to R's global symbol table, causing a small amount of memory leakage. This can be problematic in cases where many different keys are used. Fastmap avoids this memory leak issue by implementing the map using data structures in C++.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat (>= 2.1.1)

URL https://r-lib.github.io/fastmap/, https://github.com/r-lib/fastmap

BugReports https://github.com/r-lib/fastmap/issues

Roxygen list(markdown = TRUE)

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

Date/Publication 2026-04-20 16:57:33 UTC

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

RemoteRef HEAD

RemoteSha 81ab40d9ecda2a2b2e3868e594ea9b1c86cc211e

Contents

fastmap 2
fastqueue 4
faststack 5
key_missing 6

Index **7**

`fastmap` *Create a fastmap object*

Description

A fastmap object provides a key-value store where the keys are strings and the values are any R objects.

Usage

```
fastmap(missing_default = NULL)
```

Arguments

`missing_default`
: The value to return when `get()` is called with a key that is not in the map. The default is `NULL`, but in some cases it can be useful to return a sentinel value, such as a `key_missing` object.

Details

In R, it is common to use environments as key-value stores, but they can leak memory: every time a new key is used, R registers it in its global symbol table, which only grows and is never garbage collected. If many different keys are used, this can cause a non-trivial amount of memory leakage.

Fastmap objects do not use the symbol table and do not leak memory.

Unlike with environments, the keys in a fastmap are always encoded as UTF-8, so if you call `$set()` with two different strings that have the same Unicode values but have different encodings, the second call will overwrite the first value. If you call `$keys()`, it will return UTF-8 encoded strings, and similarly, `$as_list()` will return a list with names that have UTF-8 encoding.

Note that if you call `$mset()` with a named argument, where the name is non-ASCII, R will convert the name to the native encoding before fastmap has the chance to convert them to UTF-8, and the keys may get mangled in the process. However, if you use `$mset(.list = x)`, then R will not convert the keys to the native encoding, and the keys will be correctly converted to UTF-8. With `$mget()`, the keys will be converted to UTF-8 before they are fetched.

`fastmap` objects have the following methods:

`set(key, value)` Set a key-value pair. `key` must be a string. Returns `value`.

`mset(..., .list = NULL)` Set multiple key-value pairs. The key-value pairs are named arguments, and/or a list passed in as `.list`. Returns a named list where the names are the keys, and the values are the values.

`get(key, missing = missing_default)` Get a value corresponding to `key`. If the key is not in the map, return `missing`.

`mget(keys, missing = missing_default)` Get values corresponding to `keys`, which is a character vector. The values will be returned in a named list where the names are the same as the `keys` passed in, in the same order. For keys not in the map, they will have `missing` for their value.

`has(keys)` Given a vector of keys, returns a logical vector reporting whether each key is contained in the map.

`remove(keys)` Given a vector of keys, remove the key-value pairs from the map. Returns a logical vector reporting whether each item existed in (and was removed from) the map.

`keys(sort = FALSE)` Returns a character vector of all the keys. By default, the keys will be in arbitrary order. Note that the order can vary across platforms and is not guaranteed to be consistent. With `sort=TRUE`, the keys will be sorted according to their Unicode code point values.

`size()` Returns the number of items in the map.

`clone()` Returns a copy of the fastmap object. This is a shallow clone; objects in the fastmap will not be copied.

`as_list(sort = FALSE)` Return a named list where the names are the keys from the map, and the values are the values. By default, the keys will be in arbitrary order. Note that the order can vary across platforms and is not guaranteed to be consistent. With `sort=TRUE`, the keys will be sorted according to their Unicode code point values.

`reset()` Reset the fastmap object, clearing all items.

Examples

```
# Create the fastmap object
m <- fastmap()

# Set some key-value pairs
m$set("x", 100)
m$set("letters", c("a", "b", "c"))
m$mset(numbers = c(10, 20, 30), nothing = NULL)

# Get values using keys
m$get("x")
m$get("numbers")
m$mget(c("letters", "numbers"))

# Missing keys return NULL by default, but this can be customized
m$get("xyz")

# Check for existence of keys
m$has("x")
m$has("nothing")
m$has("xyz")

# Remove one or more items
m$remove(c("letters", "x"))

# Return number of items
m$size()

# Get all keys
m$keys()

# Return named list that represents all key-value pairs
```

```
str(m$as_list())

# Clear the map
m$reset()


# Specify missing value when get() is called
m <- fastmap()
m$get("x", missing = key_missing())
#> <Key Missing>

# Specify the default missing value
m <- fastmap(missing_default = key_missing())
m$get("x")
#> <Key Missing>
```

`fastqueue` *Create a queue*

Description

A `fastqueue` is backed by a list, which is used in a circular manner. The backing list will grow or shrink as the queue changes in size.

Usage

```
fastqueue(init = 20, missing_default = NULL)
```

Arguments

`init`	Initial size of the list that backs the queue. This is also used as the minimum size of the list; it will not shrink any smaller.
`missing_default`	The value to return when `remove()` or `peek()` are called when the stack is empty. Default is `NULL`.

Details

`fastqueue` objects have the following methods:

`add(x)` Add an object to the queue.

`madd(..., .list = NULL)` Add objects to the queue. `.list` can be a list of objects to add.

`remove(missing = missing_default)` Remove and return the next object in the queue, but do not remove it from the queue. If the queue is empty, this will return `missing`, which defaults to the value of `missing_default` that `queue()` was created with (typically, `NULL`).

`mremove(n, missing = missing_default)` Remove and return the next `n` objects on the queue, in a list. The first element of the list is the oldest object in the queue (in other words, the next item that would be returned by `remove()`). If `n` is greater than the number of objects in the queue, any requested items beyond those in the queue will be replaced with `missing` (typically, `NULL`).

`peek(missing = missing_default)` Return the next object in the queue but do not remove it from the queue. If the queue is empty, this will return `missing`.

`reset()` Reset the queue, clearing all items.

`size()` Returns the number of items in the queue.

`as_list()` Return a list containing the objects in the queue, where the first element in the list is oldest object in the queue (in other words, it is the next item that would be returned by `remove()`), and the last element in the list is the most recently added object.

`faststack` *Create a stack*

Description

A `faststack` is backed by a list. The backing list will grow or shrink as the stack changes in size.

Usage

```
faststack(init = 20, missing_default = NULL)
```

Arguments

`init`	Initial size of the list that backs the stack. This is also used as the minimum size of the list; it will not shrink any smaller.
`missing_default`	The value to return when `pop()` or `peek()` are called when the stack is empty. Default is `NULL`.

Details

`faststack` objects have the following methods:

`push(x)` Push an object onto the stack.

`mpush(..., .list = NULL)` Push objects onto the stack. `.list` can be a list of objects to add.

`pop(missing = missing_default)` Remove and return the top object on the stack. If the stack is empty, it will return `missing`, which defaults to the value of `missing_default` that `stack()` was created with (typically, `NULL`).

`mpop(n, missing = missing_default)` Remove and return the top `n` objects on the stack, in a list. The first element of the list is the top object in the stack. If `n` is greater than the number of objects in the stack, any requested items beyond those in the stack will be replaced with `missing` (typically, `NULL`).

`peek(missing = missing_default)` Return the top object on the stack, but do not remove it from the stack. If the stack is empty, this will return `missing`.

`reset()` Reset the stack, clearing all items.

`size()` Returns the number of items in the stack.

`as_list()` Return a list containing the objects in the stack, where the first element in the list is the object at the bottom of the stack, and the last element in the list is the object at the top of the stack.

`key_missing` *A missing key object*

Description

A `key_missing` object represents a missing key.

Usage

```
key_missing()

is.key_missing(x)
```

Arguments

`x`	An object to test.

Index

fastmap, 2
fastqueue, 4
faststack, 5

is.key_missing *(*key_missing*)*, 6

key_missing, 2, 6