

Package ‘collections’

May 8, 2026

Type Package

Title High Performance Container Data Types

Version 0.3.12

Date 2026-03-21

Description Provides high performance container data types such as queues, stacks, dequeues, dicts and ordered dicts. Benchmarks <https://randy3k.github.io/collections/articles/benchmark.html> have shown that these containers are asymptotically more efficient than those offered by other packages.

License MIT + file LICENSE

URL <https://github.com/rand3k/collections/>

Suggests testthat (>= 2.3.1)

ByteCompile yes

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.1.0

Author Randy Lai [aut, cre],
Andrea Mazzoleni [cph] (tommy hash table library),
Yann Collet [cph] (xxhash algorithm)

Maintainer Randy Lai <randy.cs.lai@gmail.com>

Repository CRAN

Date/Publication 2026-03-22 06:10:26 UTC

Contents

collections-package	2
cls	2
deprecated	3
deque	3
dict	4

ordered_dict	6
priority_queue	7
queue	8
stack	9

Index	10
--------------	-----------

collections-package *collections: High Performance Container Data Types*

Description

Provides high performance container data types such as queues, stacks, dequeues, dicts and ordered dicts. Benchmarks <<https://randy3k.github.io/collections/articles/benchmark.html>> have shown that these containers are asymptotically more efficient than those offered by other packages.

Author(s)

Maintainer: Randy Lai <randy.cs.lai@gmail.com>

Other contributors:

- Andrea Mazzoleni (tommy hash table library) [copyright holder]
- Yann Collet (xxhash algorithm) [copyright holder]

See Also

Useful links:

- <https://github.com/rand3k/collections>

cls *Inspect objects*

Description

cls is a replacement for the class function which also works for the collection objects. It falls back to the ordinary class function for other objects.

Usage

```
cls(x)
```

Arguments

x a collection object

Examples

```
d <- dict()
cls(d)
```

deprecated	<i>Deprecated Functions</i>
------------	-----------------------------

Description

Deprecated Functions

Usage

Deque(...)

Dict(...)

OrderedDict(...)

PriorityQueue(...)

Queue(...)

Stack(...)

Arguments

... anything

deque	<i>Double Ended Queue</i>
-------	---------------------------

Description

deque creates a double ended queue.

Usage

deque(items = NULL)

Arguments

items a list of items

Details

Following methods are exposed:

```
.$push(item)
.$pushleft(item)
.$pop()
.$popleft()
.$peek()
.$peekleft()
.$extend(q)
.$extendleft(q)
.$remove(item)
.$clear()
.$size()
.$as_list()
.$print()
```

- item: any R object
- q: a deque object

See Also

[queue](#) and [stack](#)

Examples

```
q <- deque()
q$push("foo")
q$push("bar")
q$pushleft("baz")
q$pop() # bar
q$popleft() # baz

q <- deque(list("foo", "bar"))
q$push("baz")$pushleft("bla")
```

dict

Dictionary

Description

dict creates an ordinary (unordered) dictionary (a.k.a. hash).

Usage

```
dict(items = NULL, keys = NULL)
```

Arguments

items	a list of items
keys	a list of keys, use names(items) if NULL

Details

Following methods are exposed:

```

.$set(key, value)
.$get(key, default)
.$remove(key, silent = FALSE)
.$pop(key, default)
.$has(key)
.$keys()
.$values()
.$update(d)
.$clear()
.$size()
.$as_list()
.$print()

```

- key: a scalar character, an atomic vector, an environment or a function
- value: any R object, value of the item
- default: optional, the default value of an item if the key is not found
- d: a dict object

See Also

[ordered_dict](#)

Examples

```

d <- dict(list(apple = 5, orange = 10))
d$set("banana", 3)
d$get("apple")
d$as_list() # unordered
d$pop("orange")
d$as_list() # "orange" is removed
d$set("orange", 3)$set("pear", 7) # chain methods

# vector indexing
d$set(c(1L, 2L), 3)$set(LETTERS, 26)
d$get(c(1L, 2L)) # 3
d$get(LETTERS) # 26

# object indexing
e <- new.env()
d$set(sum, 1)$set(e, 2)
d$get(sum) # 1
d$get(e) # 2

```

`ordered_dict`*Ordered Dictionary*

Description

`ordered_dict` creates an ordered dictionary.

Usage

```
ordered_dict(items = NULL, keys = NULL)
```

Arguments

<code>items</code>	a list of items
<code>keys</code>	a list of keys, use <code>names(items)</code> if NULL

Details

Following methods are exposed:

```
.$set(key, value)  
.$get(key, default)  
.$remove(key, silent = FALSE)  
.$pop(key, default)  
.$popitem(last = TRUE)  
.$has(key)  
.$keys()  
.$values()  
.$update(d)  
.$clear()  
.$size()  
.$as_list()  
.$print()
```

- `key`: scalar character, environment or function
- `value`: any R object, value of the item
- `default`: optional, the default value of an item if the key is not found
- `d`: an `ordered_dict` object

See Also

[dict](#)

Examples

```
d <- ordered_dict(list(apple = 5, orange = 10))
d$set("banana", 3)
d$get("apple")
d$as_list() # the order the item is preserved
d$pop("orange")
d$as_list() # "orange" is removed
d$set("orange", 3)$set("pear", 7) # chain methods
```

priority_queue	<i>Priority Queue</i>
----------------	-----------------------

Description

priority_queue creates a priority queue (a.k.a heap).

Usage

```
priority_queue(items = NULL, priorities = rep(0, length(items)))
```

Arguments

items	a list of items
priorities	a vector of interger valued priorities

Details

Following methods are exposed:

```
.$push(item, priority = 0)
.$pop()
.$clear()
.$size()
.$as_list()
.$print()
```

- item: any R object
- priority: a real number, item with larger priority pops first

Examples

```
q <- priority_queue()
q$push("not_urgent")
q$push("urgent", priority = 2)
q$push("not_as_urgent", priority = 1)
q$pop() # urgent
q$pop() # not_as_urgent
q$pop() # not_urgent
```

```
q <- priority_queue(list("not_urgent", "urgent"), c(0, 2))
q$push("not_as_urgent", 1)$push("not_urgent2")
```

queue

Queue

Description

queue creates a queue.

Usage

```
queue(items = NULL)
```

Arguments

items a list of items

Details

Following methods are exposed:

```
.$push(item)
.$pop()
.$peek()
.$clear()
.$size()
.$as_list()
.$print()
```

- item: any R object

See Also

[stack](#) and [deque](#)

Examples

```
q <- queue()
q$push("first")
q$push("second")
q$pop() # first
q$pop() # second

q <- queue(list("foo", "bar"))
q$push("baz")$push("bla")
```

stack	<i>Stack</i>
-------	--------------

Description

stack creates a stack.

Usage

```
stack(items = NULL)
```

Arguments

items a list of items

Details

Following methods are exposed:

```
.$push(item)
.$pop()
.$peek()
.$clear()
.$size()
.$as_list()
.$print()
```

- item: any R object

See Also

[queue](#) and [deque](#)

Examples

```
s <- stack()
s$push("first")
s$push("second")
s$pop() # second
s$pop() # first

s <- stack(list("foo", "bar"))
s$push("baz")$push("bla")
```

Index

`cls`, [2](#)
`collections` (`collections-package`), [2](#)
`collections-package`, [2](#)

`deprecated`, [3](#)
`Deque` (`deprecated`), [3](#)
`deque`, [3](#), [8](#), [9](#)
`Dict` (`deprecated`), [3](#)
`dict`, [4](#), [6](#)

`ordered_dict`, [5](#), [6](#)
`OrderedDict` (`deprecated`), [3](#)

`priority_queue`, [7](#)
`PriorityQueue` (`deprecated`), [3](#)

`Queue` (`deprecated`), [3](#)
`queue`, [4](#), [8](#), [9](#)

`Stack` (`deprecated`), [3](#)
`stack`, [4](#), [8](#), [9](#)