Package: SharedObject (via r-universe)

November 1, 2026

Type Package

Title Sharing R objects across multiple R processes without memory duplication

Version 1.26.2

Date 2023-8-9

Description This package is developed for facilitating parallel computing in R. It is capable to create an R object in the shared memory space and share the data across multiple R processes. It avoids the overhead of memory dulplication and data transfer, which make sharing big data object across many clusters possible.

License GPL-3

LinkingTo BH, Rcpp

Depends R (>= 3.6.0)

Imports Rcpp, methods, stats, BiocGenerics

biocViews Infrastructure

BugReports https://github.com/Jiefei-Wang/SharedObject/issues

Suggests testthat, parallel, knitr, rmarkdown, BiocStyle

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

SystemRequirements GNU make, C++11

Encoding UTF-8

Config/pak/sysreqs make

Repository Bioconductor 3.23

Date/Publication 2026-08-12 14:55:28 UTC

RemoteUrl https://github.com/bioc/SharedObject

RemoteRef RELEASE_3_23

RemoteSha 62f14b1fce30499112be8b9040021368748b765f

Contents

getLastIndex 2
is.altrep 4
is.shared 4
listSharedObjects 5
pkgconfig 6
share 7
SharedObject 10
sharedObjectPkgOptions 11
sharedObjectProperties 12
unshare 13

Index **14**

`getLastIndex` *Functions to manipulate shared memory*

Description

These functions are designed for package developers only, they can allocate, open, close and destroy shared memory without touching C++ code. Normal users should not use these functions unless dealing with memory leaking

Usage

```
getLastIndex()

allocateSharedMemory(size, name = "")

mapSharedMemory(id)

unmapSharedMemory(id)

freeSharedMemory(id)

hasSharedMemory(id)

getSharedMemorySize(id)

initialSharedObjectPackageData()

releaseSharedObjectPackageData()
```

Arguments

`size`	The size of the shared memory that you want to allocate
`name`, `id`	The name of the shared memory

Details

Quick explanation

`getLastIndex`: the ID of the last created shared memory.

`allocateSharedMemory`: allocate a shared memory of a given size, the memory ID is returned by the function

`mapSharedMemory`: map the shared memory to the current process memory space

`unmapSharedMemory`: unmap the shared memory(without destroying it)

`freeSharedMemory`: free the shared memory. This function will only unmap the shared memory on Windows.

`hasSharedMemory`: whether the memory exist?

`getSharedMemorySize`: get the actual size of the shared memory, it may be larger than the size that you required.

Details

A complete lifecycle of a shared memory involves four steps: allocating, mapping, unmapping and freeing the shared memory.

The shared memory can be created by `allocateSharedMemory`. The function `allocateSharedMemory` will return the ID of the shared memory. After creating the shared memory, it can be mapped to the current process by `mapSharedMemory`. The return value is an external pointer to the shared memory. Once the shared memory is no longer needed, it can be unmapped and destroyed by `unmapSharedMemory` and `freeSharedMemory` respectively.

Value

`getLastIndex`: An interger ID served as a hint of the last created shared memory ID.

`allocateSharedMemory`: character ID(s) that can be used to find the shared memory

`mapSharedMemory`: External pointer(s) to the shared memory

`unmapSharedMemory`: No return value

`freeSharedMemory`: No return value

`hasSharedMemory`: Logical value(s) indicating whether the shared memory exist

`getSharedMemorySize`: A numeric value

See Also

`listSharedObjects`

Examples

```
size <- 10L
## unnamed shared memory
id <- allocateSharedMemory(size)
hasSharedMemory(id)
ptr <- mapSharedMemory(id)
ptr
getSharedMemorySize(id)
```

```
unmapSharedMemory(id)
freeSharedMemory(id)
hasSharedMemory(id)

## named shared memory
name <- "SharedObjectExample"
if(!hasSharedMemory(name)){
    allocateSharedMemory(size, name = name)
    hasSharedMemory(name)
    ptr <- mapSharedMemory(name)
    ptr
    getSharedMemorySize(name)
    unmapSharedMemory(name)
    freeSharedMemory(name)
    hasSharedMemory(name)
}
```

`is.altrep` *Whether an object is an ALTREP object*

Description

Whether an object is an ALTREP object

Usage

```
is.altrep(x)
```

Arguments

`x`	an R object

Value

A logical value

`is.shared` *Test whether an object is shared*

Description

Test whether an object is shared

Usage

```
is.shared(x, ..., depth = 0, showAttributes = FALSE)

## S4 method for signature 'ANY'
is.shared(x, ..., depth = 0, showAttributes = FALSE)
```

Arguments

x	An R object
...	For generalization purpose only
depth	Whether to recursively check the element of x. This parameter only works for container objects(e.g. list and environment), see details.
showAttributes	Whether to check the attributes of x, default FALSE.

Details

When depth=0, the is.shared function return a single logical value indicating whether x is shared or contains any shared objects. When depth>0 and x is a container(e.g. list), the function will recursively check each element of x and return a list with each elements corresponding to the elements in x. The depth parameter determines the depth of the checking procedure.

if showAttributes = TRUE, the attributes of the object will also be checked. The check result is returned as attributes of the return value by appending Shared to the end of the original attribute names. Note that showAttributes has no effect on an S4 object for the attributes of an S4 object are used to store the slots and should not be treated as the attributes of an object.

Value

a single logical value or a list.

Examples

```
x1 <- share(1:10)
is.shared(x1)

x2 <- share(list(a=1:10, b = list(d = letters, e = runif(10))))
is.shared(x2, depth=0)
is.shared(x2, depth=0, showAttributes = TRUE)
is.shared(x2, depth=1)
is.shared(x2, depth=2)
```

---

listSharedObjects *List all shared Objects*

Description

List all shared Objects

Usage

```
listSharedObjects(end = NULL, start = NULL)
```

Arguments

end	the end value of the ID. The default is NULL. See details.
start	the start value of the ID. The default is NULL. See details.

Details

The parameter `start` and `end` specify the range of the ID. If not specified, all IDs will be listed.

On Ubuntu or some other linux systems, the shared objects can be found in the folder `/dev/shm`. The function can find all shared objects if the folder exists.

On Windows, since there is no easy way to find all shared objects. the function will guess the range of the shared object IDs and search for all IDs within the range. Therefore, if there are too many shared objects(over 4 billions) ,the object id can be out of the searching range and the result may not be complete. Furthermore, there will be no named shared memory in the returned list.

Note that the size in the return value is the true memory size that is reserved for the shared object, so it might be larger than the object size.

Value

A data.frame object with shared object id and size

See Also

`getLastIndex`, `allocateSharedMemory`, `mapSharedMemory`, `unmapSharedMemory`, `freeSharedMemory`, `hasSharedMemory`, `getSharedMemorySize`

Examples

```
x <- share(runif(10))
## Automatically determine the search range
listSharedObjects()

## specify the search range
listSharedObjects(start = 10, end = 20)

## Search from 0 to 20
listSharedObjects(20)
```

---

pkgconfig *Find path of the shared memory header file*

Description

This function will return the path of the shared memory header or the flags that are used to compile the package for the developers who want to use C++ level implementation of the `SharedObject` package

Usage

```
pkgconfig(x)
```

Arguments

`x`	Character, "PKG_LIBS" or "PKG_CPPFLAGS"

Value

path to the header or compiler flags

Examples

```
SharedObject:::pkgconfig("PKG_LIBS")
SharedObject:::pkgconfig("PKG_CPPFLAGS")
```

---

share *Create a shared object*

Description

This function will create a shared object for the object x. The behavior of the shared object is exactly the same as x, but the data of the shared object is allocated in the shared memory space. Therefore, a shared object can be easily exported to the other R workers without duplicating the data, which can reduce the memory consumption and the overhead of data transmission.

Usage

```
share(x, ...)

## S4 method for signature 'ANY'
share(
  x,
  ...,
  copyOnWrite,
  sharedSubset,
  sharedCopy,
  sharedAttributes,
  mustWork,
  minLength
)
```

Arguments

x	An R object that will be shared, see details.
...	For generalization purpose.
copyOnWrite, sharedSubset, sharedCopy	The parameters controlling the behavior of a shared object, see details.
sharedAttributes	Whether to share the attributes of the object x (default TRUE). Note that attribute class and names will never be shared.
mustWork	Whether to throw an error if x is not sharable (e.g. x is a function). This parameter has no effect on the object's attributes and S4 object.
minLength	The minimum length of a shared object(default 3). If length(x) is smaller than the minimum length, it would not be shared. This parameter can be used to reduce the memory fragmentation.

Details

The function returns a shared object corresponding to the argument `x` if it is sharable. There should be no different between `x` and the return value except that the latter one is shared. The attributes of `x` will also be shared if possible.

Supported types

For the basic R type, the function supports `raw`, `logical`,`integer`, `double`, `complex`. `character` can be shared, but sharing a `character` is beneficial only when there are a lot repetitions in the elements of the vector. Due to the complicated structure of the character vector, you are not allowed to set the value of a shared character vector to a value which haven't presented in the vector before. It is recommended to treat a shared character vector as read-only.

For the container, the function supports `list`, `pairlist` and `environment`. Note that sharing a container is equivalent to share all elements in the container, the container itself will not be shared.

The function `share` is an S4 generic. The default share method works for most S3/S4 objects. Therefore, there is no need to define a S4 share method for each S3/S4 class unless the S3/S4 class has a special implementation (e.g. on-disk data). The default method will share all slots the object contains and the object itself if possible. No error will be given if any of these objects are not sharable and they will be kept unchanged.

Behavior control

The behavior of a shared object can be controlled through three parameters: `copyOnWrite`, `sharedSubset` and `sharedCopy`.

`copyOnWrite` determines Whether a shared object needs to be duplicated when the data of the shared object is changed. The default value is `TRUE`, but can be altered by passing `copyOnWrite = FALSE` to the function. This parameter can be used to let workers directly write the result back to a shared object.

Please note that the no-copy-on-write feature is not fully supported by R. When `copyOnWrite` is `FALSE`, a shared object might not behaves as one expects. Please refer to the example code to see the exceptions.

`sharedSubset` determines whether the subset of a shared object is still a shared object. The default value is `FALSE`, and can be changed by passing `sharedSubset = TRUE` to the function

At the time of writing, The shared subset feature will cause an unnecessary memory duplication in R studio. Therefore, for the performance consideration, it is recommended to keep the feature off in R studio.

`sharedCopy` determines whether the object is still a shared object after the duplication. Note that it must be used with `copyOnWrite = TRUE`. Otherwise, the shared object will never be duplicated. The default value is `FALSE`.

Value

A shared object

Examples

```
## For vector
x <- runif(10)
so <- share(x)
```

```
x
so

## For matrix
x <- matrix(runif(10), 2, 5)
so <- share(x)
x
so

## For data frame
x <- as.data.frame(matrix(runif(10), 2, 5))
so <- share(x)
x
so

## export the object
library(parallel)
cl <- makeCluster(1)
clusterExport(cl, "so")
## check the exported object in the other process
clusterEvalQ(cl, so)

## close the connection
stopCluster(cl)

## Copy on write
x <- runif(10)
so1 <- share(x, copyOnWrite = TRUE)
so2 <- so1
so2[1] <- 10
## so1 is unchanged since copy-on-write feature is on.
so1
so2

## No copy on write
so1 <- share(x, copyOnWrite = FALSE)
so2 <- so1
so2[1] <- 10
#so1 is changed
so1
so2

## Flaw of no-copy-on-write
## The following code changes the value of so1,
## highly unexpected! Please use with caution!
-so1
so1
## The reason is that the minus function tries to
## duplicate so1 object, but the duplication function
## will return so1 itself, so the values in so1 get changed.
```

`SharedObject` *Create an empty shared object*

Description

Create an empty shared object with a specific length and attributes.

Usage

```
SharedObject(
  mode = c("raw", "logical", "integer", "numeric", "complex"),
  length,
  attrib = list(),
  ...
)
```

Arguments

`mode`	the type of the shared object
`length`	the length of the shared object
`attrib`	the attributes of the shared object
`...`	Parameters that is used to create the shared object, please refer to `?share` for details.

Value

An R vector

Examples

```
## Create an empty shared vector
x1 <- SharedObject(mode = "numeric", length = 10)
x1
## Create an empty shared matrix
x2 <- SharedObject(mode = "numeric", length = 6,
                   attrib = list(dim = c(2L,3L)))
x2
```

`sharedObjectPkgOptions` *Get or set the global options for the SharedObject package*

Description

Get or set the global options for the SharedObject package

Usage

```
sharedObjectPkgOptions(..., literal = TRUE)
```

Arguments

`...`	The name of the option(s), it can be either symbols or characters. if the argument is missing, it means getting all option. See examples.
`literal`	Whether the parameters in `...` are always treated as characters.

Value

`set`: The old package options

`get`: A list of the package options or a single value

Examples

```
## Get all options
sharedObjectPkgOptions()

## Get copyOnWrite only
sharedObjectPkgOptions(copyOnWrite)
sharedObjectPkgOptions("copyOnWrite")
opt <- "copyOnWrite"
sharedObjectPkgOptions(opt, literal = FALSE)


## Set options
sharedObjectPkgOptions(copyOnWrite = FALSE)
## Check if we have changed the option
sharedObjectPkgOptions(copyOnWrite)

## Restore the default
sharedObjectPkgOptions(copyOnWrite = TRUE)
```

sharedObjectProperties
Get/Set the properties of a shared object.

Description

Get/Set the properties of a shared object.

Usage

```
sharedObjectProperties(x, ..., literal = TRUE)

## S4 method for signature 'ANY'
sharedObjectProperties(x, ..., literal = TRUE)

## S4 method for signature 'list'
sharedObjectProperties(x, ..., literal = TRUE)

getCopyOnWrite(x)

getSharedSubset(x)

getSharedCopy(x)

setCopyOnWrite(x, value)

setSharedSubset(x, value)

setSharedCopy(x, value)
```

Arguments

x	A shared object
...	The name of the property(s), it can be either symbols or characters. if the argument is missing, it means getting all properties. See examples.
literal	Whether the parameters in ... are always treated as characters.
value	The value of the property

Details

For numeric objects, the properties are dataId, length, totalSize, dataType, ownData, copyOnWrite, sharedSubset, sharedCopy.

For character objects, the properties are length, unitSize,totalSize,dataType,uniqueChar,copyOnWrite.

Note that only copyOnWrite, sharedSubset and sharedCopy are mutable. The other attributes are read-only.

Value

get: The property(s) of a shared object

set: The old property(s)

Examples

```
## For numeric objects
x1 <- share(1:10)

## Get attributes
sharedObjectProperties(x1)
sharedObjectProperties(x1, copyOnWrite)
sharedObjectProperties(x1, "copyOnWrite")
props <- "copyOnWrite"
sharedObjectProperties(x1, props, literal = FALSE)
getCopyOnWrite(x1)

## Set attributes
sharedObjectProperties(x1, copyOnWrite = FALSE)
setCopyOnWrite(x1, FALSE)

## For character objects
x2 <- share(letters)
sharedObjectProperties(x2)
```

---

unshare *Unshare a shared object*

Description

Unshare a shared object. There will be no effect if the object is not shared.

Usage

```
unshare(x)
```

Arguments

x	a shared object, or an object that contains a shared object.

Value

An unshared object

Examples

```
x1 <- share(1:10)
x2 <- unshare(x1)
is.shared(x1)
is.shared(x2)
```

Index

allocateSharedMemory, *6*
allocateSharedMemory *(*getLastIndex*)*, 2

freeSharedMemory, *6*
freeSharedMemory *(*getLastIndex*)*, 2

getCopyOnWrite
        *(*sharedObjectProperties*)*, 12
getLastIndex, 2, *6*
getSharedCopy *(*sharedObjectProperties*)*,
        12
getSharedMemorySize, *6*
getSharedMemorySize *(*getLastIndex*)*, 2
getSharedSubset
        *(*sharedObjectProperties*)*, 12

hasSharedMemory, *6*
hasSharedMemory *(*getLastIndex*)*, 2

initialSharedObjectPackageData
        *(*getLastIndex*)*, 2
is.altrep, 4
is.shared, 4
is.shared,ANY-method *(*is.shared*)*, 4

listSharedObjects, *3*, 5

mapSharedMemory, *6*
mapSharedMemory *(*getLastIndex*)*, 2

pkgconfig, 6

releaseSharedObjectPackageData
        *(*getLastIndex*)*, 2

setCopyOnWrite
        *(*sharedObjectProperties*)*, 12
setSharedCopy *(*sharedObjectProperties*)*,
        12
setSharedSubset
        *(*sharedObjectProperties*)*, 12

share, 7
share,ANY-method *(*share*)*, 7
share,data.frame-method *(*share*)*, 7
share,list-method *(*share*)*, 7
share,matrix-method *(*share*)*, 7
share,vector-method *(*share*)*, 7
SharedObject, 10
sharedObjectPkgOptions, 11
sharedObjectProperties, 12
sharedObjectProperties,ANY-method
        *(*sharedObjectProperties*)*, 12
sharedObjectProperties,list-method
        *(*sharedObjectProperties*)*, 12

unmapSharedMemory, *6*
unmapSharedMemory *(*getLastIndex*)*, 2
unshare, 13
unshare,ANY-method *(*unshare*)*, 13
unshare,list-method *(*unshare*)*, 13
unshare,vector-method *(*unshare*)*, 13