Package 'memuse'

January 24, 2023

Title Memory Estimation Utilities

Version 4.2-3

Description How much ram do you need to store a 100,000 by 100,000 matrix?
How much ram is your current R session using? How much ram do you even have?
Learn the scintillating answer to these and many more such questions with
the 'memuse' package.

License BSD 2-clause License + file LICENSE

Depends R (>= 3.0.0)

Imports methods, utils

NeedsCompilation yes

ByteCompile yes

Maintainer Drew Schmidt <wrathematics@gmail.com>

URL https://github.com/shinra-dev/memuse

BugReports https://github.com/shinra-dev/memuse/issues

RoxygenNote 7.1.2

Author Drew Schmidt [aut, cre],
Christian Heckendorf [ctb] (FreeBSD improvements to meminfo),
Wei-Chen Chen [ctb] (Windows build fixes),
Dan Burgess [ctb] (donation of a Mac for development and testing)

Repository CRAN

Date/Publication 2023-01-24 15:40:15 UTC

R topics documented:

`memuse-package` *Core memuse Classes and Methods*

Description

Originally an amusing fancy printing system, this package contains that, as well as some functionality for showing. As of version 2.0, the package contains some helpful utilities for accessing hardware ram information, ram usage of the current R process, as well as some other niceties.

Details

If you do a lot of benchmarking, this is the package you've been waiting for. Maybe. I mean, I don't want to be too pushy about it.

Author(s)

Drew Schmidt

References

Project home page: `https://shinra-dev.github.io`

`Accessors` *Accessors*

Description

Accessor methods for slots of objects of class `memuse`.

Usage

```
mu.size(x, as.is = TRUE)

mu.unit(x)

mu.prefix(x)

mu.names(x)

## S4 method for signature 'memuse'
mu.size(x, as.is = TRUE)

## S4 method for signature 'memuse'
mu.unit(x)

## S4 method for signature 'memuse'
mu.prefix(x)

## S4 method for signature 'memuse'
mu.names(x)
```

Arguments

`x`	memuse object
`as.is`	logical; should the size be "as-is", or converted to bytes first.

Details

These methods are mostly just syntactic sugar for ordinary S4 slot accessing. So for example, `size(x)` is no different semantically from calling `x@size`.

There are two differences, however. The `size()` method has a parameter `as.is` which controls whether the return should be the raw value or the raw value converted to bytes first. For the latter, you should really use `as.numeric` instead, which is equivalent to calling `size(x, as.is=FALSE)`.

Value

Returns a numeric value in the case of `size()`, and `as.numeric()`, otherwise a string is returned.

See Also

memuse-class

Examples

```
## Not run:
x <- mu(1e6)

size(x)
as.numeric(x)
mu.unit(x)
mu.prefix(x)
mu.names(x)

## End(Not run)
```

Arithmetic *Binary Arithmetic*

Description

Binary arithmetic operations for memuse objects.

Usage

```
## S4 method for signature 'memuse,memuse'
e1 + e2

## S4 method for signature 'memuse,numeric'
e1 + e2

## S4 method for signature 'numeric,memuse'
e1 + e2

## S4 method for signature 'memuse,object_size'
e1 + e2

## S4 method for signature 'object_size,memuse'
e1 + e2

## S4 method for signature 'memuse,memuse'
e1 - e2

## S4 method for signature 'memuse,numeric'
e1 - e2

## S4 method for signature 'numeric,memuse'
```

```
e1 - e2

## S4 method for signature 'memuse,missing'
e1 - e2

## S4 method for signature 'memuse,object_size'
e1 - e2

## S4 method for signature 'object_size,memuse'
e1 - e2

## S4 method for signature 'memuse,memuse'
e1 * e2

## S4 method for signature 'memuse,numeric'
e1 * e2

## S4 method for signature 'numeric,memuse'
e1 * e2

## S4 method for signature 'memuse,object_size'
e1 * e2

## S4 method for signature 'object_size,memuse'
e1 * e2

## S4 method for signature 'memuse,memuse'
e1 / e2

## S4 method for signature 'memuse,numeric'
e1 / e2

## S4 method for signature 'numeric,memuse'
e1 / e2

## S4 method for signature 'memuse,object_size'
e1 / e2

## S4 method for signature 'object_size,memuse'
e1 / e2

## S4 method for signature 'memuse,memuse'
e1 ^ e2

## S4 method for signature 'memuse,numeric'
e1 ^ e2
```

Arguments

`e1, e2`	`memuse`, `numeric`, or `object_size` objects.

Details

Simple binary arithmetic for `memuse` objects. Options include any combination of `memuse`, `object_size` (output from the `object.size()` function), and `numeric` objects.

Value

Returns a `memuse` class object.

See Also

`Constructor`, `memuse-class`

Examples

```
## Not run:
x <- mu(200)
y <- mu(100)

x+y
x-y
x*y
x/y
x^2

## End(Not run)
```

`cachelinesize` *Cache Sizes and Linesize*

Description

Shows the size of the cache line.

Usage

```
Sys.cachelinesize()
```

Details

`Sys.cachelinesize()` will return the cache linesize. It's almost certainly 32 or 64 bytes.

Value

Returns a list, whose values are platform dependent.

See Also

meminfo

Examples

```
## Not run:
library(memuse)

Sys.cachelinesize()

## End(Not run)
```

---

cachesize *Cache Sizes*

Description

Shows the sizes of the CPU caches.

Usage

```
Sys.cachesize()
```

Details

`Sys.cachesize()` will check the various levels of cache and return all available cache information in a list. If you don't have some kind of level-1 cache, then it will return an error. If you have some kind of future space computer with more than 3 levels of cache, levels higher than 3 will not be displayed.

Value

Returns a list, whose values are platform dependent.

See Also

meminfo

Examples

```
## Not run:
library(memuse)

Sys.cachesize()

## End(Not run)
```

`Comparators` *Comparators*

Description

Binary comparators for `memuse` objects.

Usage

```
## S4 method for signature 'memuse,memuse'
e1 == e2

## S4 method for signature 'memuse,memuse'
e1 < e2

## S4 method for signature 'memuse,memuse'
e1 <= e2

## S4 method for signature 'memuse,memuse'
e1 > e2

## S4 method for signature 'memuse,memuse'
e1 >= e2

## S4 method for signature 'memuse,memuse'
e1 != e2

## S4 method for signature 'memuse,numeric'
e1 == e2

## S4 method for signature 'memuse,numeric'
e1 < e2

## S4 method for signature 'memuse,numeric'
e1 <= e2

## S4 method for signature 'memuse,numeric'
e1 > e2

## S4 method for signature 'memuse,numeric'
e1 >= e2

## S4 method for signature 'memuse,numeric'
e1 != e2

## S4 method for signature 'numeric,memuse'
e1 == e2
```

```
## S4 method for signature 'numeric,memuse'
e1 < e2

## S4 method for signature 'numeric,memuse'
e1 <= e2

## S4 method for signature 'numeric,memuse'
e1 > e2

## S4 method for signature 'numeric,memuse'
e1 >= e2

## S4 method for signature 'numeric,memuse'
e1 != e2
```

Arguments

`e1, e2`	`memuse`, `numeric`, or `object_size` objects.

Details

Comparisons to numeric values are done at the byte level.

Value

Returns a `memuse` class object.

See Also

`Constructor`, `memuse-class`

Examples

```
## Not run:
x <- mu(2000)
y <- mu(3000)

x < y
x <= y
x > y
x >= y
x == y
x != y

## End(Not run)
```

Constructor *memuse Constructor*

Description

Constructor for objects of class memuse.

Usage

```
mu(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'ANY'
mu(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature '`NULL`'
mu(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'numeric'
mu(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'object_size'
mu(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'missing'
mu(size, unit = "best", prefix = "IEC", names = "short")

memuse(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'ANY'
memuse(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature '`NULL`'
memuse(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'missing'
memuse(size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'numeric'
memuse(size = size, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'object_size'
memuse(size, unit = "best", prefix = "IEC", names = "short")
```

Arguments

size	numeric; indicates the unit-multiple number of bytes used by the object.

`unit`	`string`; the unit of storage, such as "MiB" or "MB", depending on prefix. Case is ignored.
`prefix`	`string`; the unit prefix, namely IEC or SI. Case is ignored.
`names`	`string`; control for whether the unit names should be printed out or their abbreviation should be used. Options are "long" and "short", respectively. Case is ignored.

Details

For numeric objects, if the length is 1, then its value is used as the number of bytes. Otherwise, the object's memory usage in R is taken for the size parameter.

Value

Returns a `memuse` class object.

See Also

`memuse-class` `Accessors` `Converters`

Examples

```
## Not run:
### The value passed as 'size' is the number of bytes
x <- memuse(100, unit="kb")
x

y <- memuse(100, unit="kb", prefix="SI")
y

### Use the memory usage of object 'size'
memuse(rnorm(1e4))

## End(Not run)
```

`Converters` *Converters*

Description

Converter methods between memuse and base R objects.

Usage

```
as.memuse(x, ...)

## S4 method for signature 'numeric'
as.memuse(x, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'object_size'
as.memuse(x, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'character'
as.memuse(x, unit = "best", prefix = "IEC", names = "short")

## S4 method for signature 'memuse'
as.character(x, ...)

## S4 method for signature 'memuse'
as.numeric(x, ...)
```

Arguments

`x`	Numeric value, object_size data, or appropriate string (see details section for more information).
`...`	Additional arguments.
`unit`	`string`; the unit of storage, such as "MiB" or "MB", depending on prefix. Case is ignored.
`prefix`	`string`; the unit prefix, namely IEC or SI. Case is ignored.
`names`	`string`; control for whether the unit names should be printed out or their abbreviation should be used. Options are "long" and "short", respectively. Case is ignored.

Details

These methods convert numeric, `object_size`, and string (character) objects to/from `memuse` objects.

`as.numeric(x)` for a memuse object `x` is just sugar for `mu.size(x, as.is=FALSE)`

Strings must be of the same form as the printed output of a a memuse object. For example, "100 KiB" is valid, but "100 (KiB)" is not. As always, case of the unit is ignored, and so "100 kib" would be valid as well.

Value

Returns a character, numeric, or `memuse` object, depending on the call.

See Also

`memuse-class` `Accessors`

Examples

```
## Not run:
as.memuse(10)

## End(Not run)
```

---

filesize *filesize*

Description

Returns size of a file as reported by the file system (the file is not scanned).

Usage

```
Sys.filesize(filename)

Sys.dirsize(dirname)
```

Arguments

`filename, dirname`	Location of the file/directory (as a string).

Details

All of the C-level source code for these methods (in src/meminfo of the root directory of the memuse source tree) is licensed under the permissive 2-Clause BSD license.

Value

A memuse class object.

Examples

```
## Not run:
library(memuse)

x <- rnorm(1e5)
memuse(x) ### size in ram

tmp <- tempfile()
saveRDS(x, file=tmp)
Sys.filesize(tmp) ### size on disk
unlink(tmp)

## End(Not run)
```

howbig *howbig*

Description

Determines the memory usage for a dense, in-core, numeric matrix of specified rows/columns.

Usage

```
howbig(
  nrow = 1,
  ncol = 1,
  representation = "dense",
  unit = "best",
  prefix = "IEC",
  names = "short",
  sparsity = 0.05,
  type = "double",
  intsize = 4
)
```

Arguments

`nrow, ncol`	Number of (global) rows/columns of the matrix.
`representation`	The kind of storage the object would be in, i.e. "dense" or "sparse".
`unit`	string; the unit of storage, such as "MiB" or "MB", depending on prefix. Case is ignored.
`prefix`	string; the unit prefix, namely IEC or SI. Case is ignored.
`names`	string; control for whether the unit names should be printed out or their abbreviation should be used. Options are "long" and "short", respectively. Case is ignored.
`sparsity`	The proportion of sparsity of the matrix if `representation="sparse"`
`type`	"float", "double", or "int"; the storage type of the data matrix. If you don't know the type, it is probably stored as a double, so the default value will suffice.
`intsize`	The size (in bytes) of an integer. Default is 4, but this is platform dependent.

Details

These functions provide the memory usage of an unallocated, dense, in-core, numeric matrix. As the name suggests, `howbig()` simply returns the size (as a `memuse` object).

Value

returns a `memuse` class object.

See Also

howmany

Examples

```
## Not run:
# size of a 1000x1000 matrix
howbig(1000, 1000)

## End(Not run)
```

howmany *howmany*

Description

How many rows/columns of a matrix can be stored for a given memory size.

Usage

```
howmany(
  x,
  nrow,
  ncol,
  out.type = "full",
  representation = "dense",
  ...,
  sparsity = 0.05,
  type = "double",
  intsize = 4,
  names = "short"
)
```

Arguments

`x`	The size of a matrix stored as a `memuse` class object.
`nrow, ncol`	Number of (global) rows/columns of the matrix.
`out.type`	If the full dimensions or a reduced representation should be returned (see Details section below). Options are "full" and "approximate" (with partial matching).
`representation`	The kind of storage the object would be in, i.e. "dense" or "sparse".
`...`	Additional arguments.
`sparsity`	The proportion of sparsity of the matrix if `representation="sparse"`
`type`	"double" or "int"; the storage type of the data matrix. If you don't know the type, it is probably stored as a double, so the default value will suffice.

`intsize`	The size (in bytes) of an integer. Default is 4, but this is platform dependent.
`names`	string; control for whether the unit names should be printed out or their abbreviation should be used. Options are "long" and "short", respectively. Case is ignored.

Details

This function provides the maximum dimension of an unallocated, dense, in-core, numeric matrix of known byte size. For example, it will show the largest possible square matrix which is 16 GiB (46340x46340).

If the both `nrow` and `ncol` are missing (blank inputs), then the largest square matrix will be returned. If one of `nrow` or `ncol` is supplied and the other is missing, then the non-supplied argument (`nrow` or `ncol`) will be determined according to the supplied one. If both arguments are supplied, an error is produced — you probably meant to use `howmany()`.

If `out.type="approximate"`, then a reduced representation of the dimensions will be returned. For example, the reduced representation of the number 1234567890 would be "1.2b", since this number is basically 1.2 billion. Not super useful, but kind of cute, and it arguably enhances readability when fishing for a ballpark figure.

Value

A numeric pair, the dimensions of a matrix.

See Also

`howbig`

Examples

```
## Not run:
x <- mu(1, "gib")

# largest square matrix that's 1 GiB
howmany(x)
# same, but ballpark figure
howmany(mu(1, "gib"), out.type="approx")

## End(Not run)
```

`hr` *hr*

Description

A rich man’s exponential notation.

Usage

```
hr(x, names = "comma", digits = 2)
```

Arguments

`x`	A number.
`names`	"long", "short", or "comma"; determines wheter the output reads like "10 million", "10m", or "10,000,000", respectively.
`digits`	The number of decimal digits to retain.

Details

Approximate size of an integer. Very useful when dealing with potentially large values, such as those from `howmany()`.

Value

An object of class `humanreadable`.

See Also

`howmany`

Examples

```
## Not run:
library(memuse)

hr(12345678)
hr(12345678, "long")

## End(Not run)
```

`meminfo` *meminfo*

Description

Platform memory information.

Usage

```
Sys.meminfo(compact.free = TRUE)
```

Arguments

`compact.free`	logical; determines whether various free memory values should be combined into a single value. See details section for more information.

Details

`Sys.meminfo()` returns some basic memory values, such as total ram, free ram, and ram used for buffers/cache (when applicable).

All of the C-level source code for these methods (in src/meminfo of the root directory of the memuse source tree) is licensed under the permissive 2-Clause BSD license.

Value

Returns a list, whose values are platform dependent in addition to being modified by input arguments.

See Also

`procmem`

Examples

```
## Not run:
library(memuse)

Sys.meminfo()

## End(Not run)
```

`memuse-class` *Class memuse*

Description

Memory usage class object.

Slots

`size` The actual size in some memuse units.

`unit` The mem unit (e.g., byte, kilobyte, etc.)

`unit.prefix` IEC or SI units

`unit.names` short (e.g., kb) or long (e.g., kilobyte)

See Also

`Control` `Constructor`

`print-hr` *Print* `humanreadable` *objects*

Description

Printing for `hr()`

Usage

```
## S3 method for class 'humanreadable'
print(x, ...)
```

Arguments

`x`	`humanreadable` object
`...`	unused

`print-memuse` *Printing*

Description

Print and show methods for `memuse` class objects.

Usage

```
## S4 method for signature 'memuse'
print(
  x,
  ...,
  unit = mu.unit(x),
  prefix = mu.prefix(x),
  names = mu.names(x),
  digits = 3
)

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

Arguments

`x, object`	memuse class object
`...`	extra arguments
`unit`	the unit to be used in printing; defaults to `x`'s unit
`prefix`	the unit prefix to be used in printing; defaults to `x`'s prefix
`names`	the unit names (short or long) to be used in printing; defaults to `x`'s names
`digits`	the number of decimal digits to print; default is 3

See Also

\ link{Constructor} memuse-class

Examples

```
## Not run:
x <- mu(1e6)

print(x)
x # same as show(x)

## End(Not run)
```

print-sysinfo *Print* sysinfo *objects*

Description

Printing for sysinfo objects.

Usage

```
## S3 method for class 'sysinfo'
print(x, ...)
```

Arguments

x	sysinfo object
...	unused

procmem *procmem*

Description

Shows the amount of ram used by the current R process.

Usage

```
Sys.procmem(gcFirst = TRUE)
```

Arguments

gcFirst	logical; determines if garbage collection should be called before getting process memory usage.

Details

`Sys.procmem()` returns the total memory usage of the current R process, and (if supported), the maximum memory usage as well.

All of the C-level source code for these methods (in src/meminfo of the root directory of the memuse source tree) is licensed under the permissive 2-Clause BSD license.

Value

Returns a list, whose values are platform dependent in addition to being modified by input arguments.

See Also

`meminfo`

Examples

```
## Not run:
library(memuse)

### How much is being used?
Sys.procmem()

### Use more.
x <- rnorm(1e7)
Sys.procmem()

### Use less.
rm(x)
gc(FALSE)
Sys.procmem()

## End(Not run)
```

`sum,memuse-method` *memuse Arithmetic*

Description

Binary arithmetic operations for `memuse` objects.

Usage

```
## S4 method for signature 'memuse'
sum(x, ..., na.rm = FALSE)
```

Arguments

`x`	A memuse object.
`...`	Additional arguments
`na.rm`	Whether NA's should be ignored.

Details

Simple arithmetic reductions.

Value

Returns a memuse class object.

See Also

Constructor memuse-class

Examples

```
## Not run:
x = mu(2000)
y = mu(5000)

sum(x, y)

### Mixing numeric and memuse objects will work, but the first one must be a
### memuse object.
sum(mu(10), 10) # This will work
sum(10, mu(10)) # This will not

## End(Not run)
```

`swapinfo` *swapinfo*

Description

Platform swap information.

Usage

```
Sys.swapinfo()

Sys.pageinfo()
```

Details

`Sys.swapinfo()` returns basic swap/page (virtual memory) information. `Sys.pageinfo()` is identical to `swapinfo()` in every way but name (provided for Windows users who may be more comfortable/familiar with the 'pagefile' naming convention).

All of the C-level source code for these methods (in src/meminfo of the root directory of the memuse source tree) is licensed under the permissive 2-Clause BSD license.

Value

Returns a list, whose values are platform dependent in addition to being modified by input arguments.

See Also

`procmem`

Examples

```
## Not run:
library(memuse)

Sys.swapinfo()

## End(Not run)
```

`Swaps` *Swaps*

Description

Methods for swapping between different memuse formats.

Usage

```
swap.prefix(x)

## S4 method for signature 'memuse'
swap.prefix(x)

swap.names(x)

## S4 method for signature 'memuse'
swap.names(x)

swap.unit(x, unit)

## S4 method for signature 'memuse'
swap.unit(x, unit)
```

Arguments

`x`	memuse object
`unit`	new unit for the `memuse` object after the swap occurs

Details

These methods allow simple (coherent) swaps between the different `memuse` formats.

`swap.unit()` will switch an object to another, supplied unit. If the unit is from another prefix, then the prefix too will change. In this case, the size will change appropriately.

`swap.prefix()` will change an object from one unit.prefix to the other. In this case, the size will change appropriately.

`swap.names` will change from short to long, or long to short printing. The size and prefix of the object are unchanged.

Value

Returns a `memuse` class object.

See Also

`Constructor` `memuse-class`

Examples

```
## Not run:
x <- mu(1e6)

x
swap.prefix(x)
swap.names(x)
swap.unit(x, "bytes")

## End(Not run)
```

Index