

MemUse

July 11, 2013

memuse-package

Core memuse Classes and Methods

Description

The package gives the user the size (memory usage) of an in-core, dense matrix. The output is an S4 class object that can be manipulated to be presented in different ways (different units, short/long versions of those units, etc.). For more information, see the package vignette and the manual.

Details

Package: memuse
Type: Package
License: GPL
LazyLoad: yes

Author(s)

wrathematics

References

<http://wrathematics.github.io>

memuse-class

Class memuse

Description

Memory usage class object.

Creating Objects

```
new('memuse', size = ..., unit = ..., unit.prefix = ..., unit.names = ...)
```

Slots

size: Object of class `numeric`
unit: Object of class `character`
unit.prefix: Object of class `character`
unit.names: Object of class `character`

Prototype

numeric size 0
character unit "B"
character unit.prefix "IEC"
character unit.names "short"

Details

`memuse` is the container for memory usage data for an unallocated, dense, in-core R object. The `size` slot contains the memory usage in some unit of bytes. The `unit` slot contains the unit of bytes that `size` is stored in (e.g., kb, mb, gb, ...). The `unit.prefix` slot contains the unit prefix, either IEC or SI. The `unit.names` slot contains the unit names, either short (e.g., kb) or long (e.g., kilobyte).

See the `memuse` guide vignette for more details.

See Also

[Control](#)

Description

A set of controls which provides default values for many functions in this package.

Details

.UNIT defaults to "best". The default choice will scale size values to the nearest (by scaling factor — 1024 or 1000 depending on unit prefix). Other acceptable choices are, for example, "kb" or "kib". If the user requests the wrong unit by prefix (e.g., "kb" instead of "kib" when the unit prefix is IEC), then the correct one will be chosen for the user.

.PREFIX defaults to "IEC". Acceptable values are "IEC" and "SI".

.NAMES defaults to "short". Acceptable values are "short" and "long".

.PRECEDENCE defaults to "prefix". Acceptable values are "unit" and "prefix".

All values are case insensitive. The correct case will be determined for the user if the incorrect case is supplied. For an explanation of what these values do, see [memuse-class](#) or the package user guide vignette.

See Also

[memuse-class](#), [Constructor](#)

Environment	<i>Environment for the memuse Package</i>
-------------	---

Description

The environment for the memuse package.

Constructor	<i>memuse Constructor</i>
-------------	---------------------------

Description

Constructor for objects of class memuse.

Usage

```
memuse(size=0, unit=.UNIT, unit.prefix=.PREFIX, unit.names=.NAMES)
mu(size=0, unit=.UNIT, unit.prefix=.PREFIX, unit.names=.NAMES)
```

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.
unit.prefix	string; the unit prefix, namely IEC or SI. Case is ignored.
unit.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

Simple constructor for memuse objects.

Value

Returns a memuse class object.

See Also

[Constructor](#), [memuse-class](#)

Examples

```
x <- mu(100, unit="kb")
x

y <- mu(100, unit="kb", unit.prefix="SI")
y
```

Accessors	<i>Accessors</i>
-----------	------------------

Description

Accessor methods for slots of objects of class memuse.

Usage

```
## S4 method for signature 'memuse'
size(x, as.is=TRUE)
## S4 method for signature 'memuse'
as.numeric(x, ...)
## S4 method for signature 'memuse'
unit(x)
## S4 method for signature 'memuse'
unit.prefix(x)
## S4 method for signature 'memuse'
unit.names(x)
```

Arguments

- x memuse object
- as.is logical; should the size be "as-is", or converted to bytes first.
- ... Additional arguments; in this case, they are ignored.

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.

Methods

```
signature(x = "memuse")
```

See Also

[Replacers](#), [memuse-class](#)

Examples

```
x <- mu(1e6)

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

Print

Printing

Description

Print and show methods for memuse class objects.

Usage

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

Arguments

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

Value

Returns a string.

Methods

`signature(x = "memuse")`

See Also

[Constructor](#), [memuse-class](#)

Examples

```
x <- mu(1e6)

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

Replacers

Replacers

Description

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

Usage

```
size(x) <- value
unit(x) <- value
unit.prefix(x) <- value
unit.names(x) <- value
```

Arguments

<code>x</code>	memuse object
<code>value</code>	replacement value

Details

These methods are syntactic sugar for assignment using ordinary S4 accessors. So for example, `size(x) <- 10` is semantically no different from calling `x@size <- 10`

These methods are strict replacement methods; if you need to swap the units of a memuse class object, you should probably be using the [Swaps](#) methods. See example below for further details.

Value

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

Methods

```
signature(x = "memuse")
```

See Also

[Accessors](#), [memuse-class](#)

Examples

```
x <- mu(2000, unit="bytes")
x

size(x) <- 1000
x
```

Arithmetic

memuse Arithmetic

Description

Binary arithmetic operations for memuse objects.

Usage

```
x + y
x - y
x * y
x / y
x ^ y
```

Arguments

`x`, `y` memuse class objects

Details

Simple binary arithmetic for memuse objects. Options are memuse-memuse, memuse-numeric, and numeric-memuse.

Value

Returns a memuse class object.

Methods

```
signature(x = "memuse", y = "memuse")
signature(x = "numeric", y = "memuse")
signature(x = "memuse", y = "numeric")
```

See Also

[Constructor](#), [memuse-class](#)

Examples

```
x <- mu(200)
y <- mu(100)

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

Swaps	<i>Swaps</i>
-------	--------------

Description

Methods for swapping between different memuse formats.

Usage

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

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.

Methods

```
signature(x = "memuse")
```

See Also

[Constructor](#), [memuse-class](#)

Examples

```
x <- mu(1e6)

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

howbig

howbig

Description

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

Usage

```
howbig(nrow, ncol, unit=.UNIT, unit.prefix=.PREFIX, unit.names=.NAMES,
       ..., type="double", intsize=4)
howbig.par(nrow, ncol, cores, par="mpi", unit=.UNIT, unit.prefix=.PREFIX,
           unit.names=.NAMES, ..., type="double", intsize=4, ICTXT=0)
```

Arguments

<code>nrow, ncol</code>	Number of (global) rows/columns of the matrix.
<code>cores</code>	The number of cores/processors
<code>par</code>	Type of data distribution. Choices are "dmat" or "mpi". The former is for pbd-DMAT objects, the latter is the simple, locally load-balanced block partitioning.
<code>unit</code>	string; the unit of storage, such as "MiB" or "MB", depending on prefix. Case is ignored.
<code>unit.prefix</code>	string; the unit prefix, namely IEC or SI. Case is ignored.
<code>unit.names</code>	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.
<code>...</code>	Additional arguments.
<code>type</code>	"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.
<code>intsize</code>	The size (in bytes) of an integer. Default is 4, but this is platform dependent.
<code>ICTXT</code>	BLACS context number; only used with <code>howbig.par()</code> with <code>par="dmat"</code> .

Details

These functions provide the memory usage of an unallocated, dense, in-core, numeric matrix. As the names suggest, `howbig()` simply returns the size (as a `memuse` object), while `howbig.par()` is the parallel, distributed analogue. The latter returns the memory usage of a *distributed*, object

Value

`howbig()` returns a `memuse` class object.

`howbig.par()` returns a list of 2 elements, each of class `memuse`. One is the total memory usage, the other is the local memory usage.

See Also

[howmany](#)

Examples

```
# size of a 1000x1000 matrix
howbig(1000, 1000)
```

howmany

How Many Rows/Cols of a Matrix for a Memory Size

Description

Binary arithmetic operations for memuse objects.

Usage

```
## S4 method for signature 'memuse'
howmany(x, nrow, ncol, unit=.UNIT, unit.prefix=.PREFIX,
        unit.names=.NAMES, ..., type="double", intsize=4)
```

Arguments

x	The size of a matrix stored as a memuse class object.
nrow, ncol	Number of (global) rows/columns of the matrix.
unit	string; the unit of storage, such as "MiB" or "MB", depending on prefix. Case is ignored.
unit.prefix	string; the unit prefix, namely IEC or SI. Case is ignored.
unit.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.
...	Additional arguments.
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.

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().

Value

Returns a numeric pair, the dimensions of a matrix.

Methods

```
signature(x = "memuse", y = "memuse")  
signature(x = "numeric", y = "memuse")  
signature(x = "memuse", y = "numeric")
```

See Also

[howbig](#)

Examples

```
x <- mu(1, "gib")  
  
# largest square matrix that's 1 GiB  
howmany(x)
```

Index

*Topic **Classes**

memuse-class, 1

*Topic **Data**

Control Variables, 2

*Topic **Methods**

Accessors, 4

Arithmetic, 7

Constructor, 3

howbig, 9

howmany, 11

Print, 5

Replacers, 6

Swaps, 8

*Topic **Package**

memuse-package, 1

*(Arithmetic), 7

*, memuse, memuse-method (Arithmetic), 7

*, memuse, numeric-method (Arithmetic), 7

*, numeric, memuse-method (Arithmetic), 7

*-method (Arithmetic), 7

+(Arithmetic), 7

+, memuse, memuse-method (Arithmetic), 7

+, memuse, numeric-method (Arithmetic), 7

+, numeric, memuse-method (Arithmetic), 7

+-method (Arithmetic), 7

-(Arithmetic), 7

-, memuse, memuse-method (Arithmetic), 7

-, memuse, missing-method (Arithmetic), 7

-, memuse, numeric-method (Arithmetic), 7

-, numeric, memuse-method (Arithmetic), 7

--method (Arithmetic), 7

.NAMES (Control Variables), 2

.PRECEDENCE (Control Variables), 2

.PREFIX (Control Variables), 2

.UNIT (Control Variables), 2

.memuse_envir (Environment), 3

/(Arithmetic), 7

/, memuse, memuse-method (Arithmetic), 7

/, memuse, numeric-method (Arithmetic), 7

/, numeric, memuse-method (Arithmetic), 7

/-method (Arithmetic), 7

^ (Arithmetic), 7

^, memuse, memuse-method (Arithmetic), 7

^, memuse, numeric-method (Arithmetic), 7

^-method (Arithmetic), 7

Accessors, 4, 7

Arithmetic, 7

as.numeric (Accessors), 4

as.numeric, memuse-method (Accessors), 4

as.numeric-method (Accessors), 4

Constructor, 3, 3, 4, 6, 8, 9

Control, 2

Control (Control Variables), 2

Control Variables, 2

Environment, 3

howbig, 9, 12

howmany, 10, 11

howmany, memuse-method (howmany), 11

howmany-method (howmany), 11

memuse (Constructor), 3

memuse-class, 1

memuse-package, 1

mu (Constructor), 3

Print, 5

print (Print), 5

print, memuse-method (Print), 5

print-method (Print), 5

Replacers, 5, 6

show (Print), 5

show, memuse-method (Print), 5

show-method (Print), 5

size (Accessors), 4

size, memuse-method (Accessors), 4
size-method (Accessors), 4
size<- (Replacers), 6
size<-, memuse-method (Replacers), 6
size<--method (Replacers), 6
swap.names (Swaps), 8
swap.names, memuse-method (Swaps), 8
swap.names-method (Swaps), 8
swap.prefix (Swaps), 8
swap.prefix, memuse-method (Swaps), 8
swap.prefix-method (Swaps), 8
swap.unit (Swaps), 8
swap.unit, memuse-method (Swaps), 8
swap.unit-method (Swaps), 8
Swaps, 7, 8

unit (Accessors), 4
unit, memuse-method (Accessors), 4
unit-method (Accessors), 4
unit.names (Accessors), 4
unit.names, memuse-method (Accessors), 4
unit.names-method (Accessors), 4
unit.names<- (Replacers), 6
unit.names<-, memuse-method (Replacers),
6
unit.names<--method (Replacers), 6
unit.prefix (Accessors), 4
unit.prefix, memuse-method (Accessors), 4
unit.prefix-method (Accessors), 4
unit.prefix<- (Replacers), 6
unit.prefix<-, memuse-method
(Replacers), 6
unit.prefix<--method (Replacers), 6
unit<- (Replacers), 6
unit<-, memuse-method (Replacers), 6
unit<--method (Replacers), 6