memuse manual

December 6, 2013

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

Description

The package gives the user the size (memory usage) of an unallocated, dense, in-core numeric object (matrix/vector). 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.). Details about these various options are outlined in the rest of this manual, as well as the package vignette.

Details

Package:	memuse
Type:	Package
License:	GPL (>= 2)
LazyData:	yes
LazyLoad:	yes
NeedsCompilation:	no

Author(s)

Drew Schmidt

References

Project home page: http://wrathematics.github.io

CRAN listing: http://cran.r-project.org/web/packages/memuse/index.html

memuse-class *Class memuse*

Description

Memory usage class object.

Creating Objects

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

There is almost certainly no compelling reason to work with this class directly. Instead, you should use the host of methods for this class.

See Also

`Control` `Constructor`

`Control Variables` *Control Variables for the memuse Package.*

Description

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

Details

Currently, there are 4 control data objects:

- `.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, in that the correct case will be determined for the user if the incorrect case is supplied. For example, if the user sets `.PREFIX <- "si"`, then the correct case ("SI") will be determined as needed.

For a "human readable" explanation of what these values do, see the package vignette.

See Also

`memuse-class` `Constructor`

`Environment` *Environment for the memuse Package*

Description

The environment for the memuse package. The package control values are stored here.

Constructor *memuse Constructor*

Description

Constructor for objects of class memuse.

Usage

```
## S4 method for signature 'numeric'
memuse(size=0, unit=.UNIT, unit.prefix=.PREFIX, unit.names=.NAMES)
## S4 method for signature 'numeric'
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

memuse-class Accessors Converters

Examples

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

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

Converters *Converters*

Description

Converter methods.

Usage

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

Arguments

x	Numeric or object_size data.
...	Additional arguments; currently ignored.

Details

These methods convert numeric or `object_size` objects into `memuse` objects.

Value

Returns a `memuse` object.

Methods

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

See Also

`memuse-class` `Accessors`

Examples

```
as.memuse(10)
```

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

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

Value

Returns a string.

Methods

signature(x = "memuse")

See Also

\ link{Constructor} memuse-class

Examples

```
x <- mu(1e6)

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

Accessors *Accessors*

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

memuse-class Replacers

Examples

```
x <- mu(1e6)

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

`x`	memuse object
`value`	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`, `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.

Methods

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

See Also

`Constructor` `memuse-class`

Examples

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

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

Swaps *Swaps*

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=1, par="balanced", unit=.UNIT,
unit.prefix=.PREFIX, unit.names=.NAMES, ..., type="double",
intsize=4, ICTXT=0, bldim=c(4, 4))
```

Arguments

nrow, ncol	Number of (global) rows/columns of the matrix.
cores	The number of cores/processors
par	Type of data distribution. Choices are "dmat" or "balanced". The former is for **pbdDMAT** objects, the latter is the simple, locally load-balanced block partitioning.
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.
ICTXT	BLACS context number; only used with howbig.par() with par="dmat".
bldim	Blocking factor for block-cyclically decomposed (dmat) data.

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 *howmany*

Description

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

Usage

```
## S4 method for signature 'memuse'
howmany(x, nrow, ncol, out.type="full", ...,  type="double", intsize=4)
## S4 method for signature 'memuse'
howmany.par(x, nrow, ncol, out.type="full", cores=1, par="row", ..., type="double",
            intsize=4, ICTXT=0, bldim=c(4, 4))
```

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).
cores	The number of cores/processors
par	Type of data distribution. Choices are "row", "column", and "dmat". The first is for row-contiguous (distribyted by row) distributions, such as the "GBD" format (see the pbdDEMO package vignette). Next, "column" is for the transpose, column-contiguous distributions. Finally, "dmat" is for **pbdDMAT** objects.
...	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.
ICTXT	BLACS context number; only used with howbig.par() with par="dmat".
bldim	Blocking factor for block-cyclically decomposed (dmat) data.

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

howmany() returns a numeric pair, the dimensions of a matrix.

howmany.par() returns a list (the global and local dimensions), each of which is a numeric pair.

Methods

signature(x = "memuse")

See Also

howbig

Examples

```
x <- mu(1, "gib")

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

Index