Package: ttutils (via r-universe)

September 2, 2024

Version 1.0-1.1

Date 2009-06-18

Title Utility Functions

Author Thorn Thaler <thorn.thaler@thothal.com>

Maintainer Thorn Thaler <thorn.thaler@thothal.com>

Description Contains some auxiliary functions.

License GPL-2

Repository CRAN

Date/Publication 2022-04-04 09:25:58 UTC

NeedsCompilation no

Contents

`ttutils-package` *Utility Functions*

Description

The package **ttutils** contains some auxiliary functions.

See section 'Index' for a list of exported functions. Section 'Internals' lists the internal functions of the package, which are not exported but may be referenced by `ttutils:::.functionName`.

Details

Version:	1.0-1
Date:	2009-06-18
License:	GPL-2
Built:	R 2.8.1; ; 2009-06-22 15:18:40; unix

Index

Internals

Author(s)

Thorn Thaler <thorn.thaler@thothal.com>

Maintainer: Thorn Thaler <thorn.thaler@thothal.com>

---

check *Check Objects*

Description

check is a generic function that tests the validity of a given object.

Usage

```
check(object, ...)
```

Arguments

`object`	an object to be tested for validity.
`...`	further arguments to be passed to the particular dispatched function.

Details

`check` tests if a given object meets the formal requirements of being a valid object of its class. If the test fails, additional warnings should be provided, giving some information why the test failed.

Value

returns `TRUE` if `object` passes the validity test for the specific class and `FALSE` otherwise.

Note

R's dispatching mechanism determines the class of a given object and then calls the function `check.<class-name>`. If no specific `check` function is found, `check.default` is called. The function `check.default` does not make much sense, for the purpose of `check` is to test the validity for a *specific class*. Hence, `check.default` simply returns `FALSE` together with a warning message that no specific `check.<class-name>` function was found.

The dispatching mechanism has immediately two consequences:

1. a class specific `check` routine need not to check whether the object belongs to the class itself, because if it would not, the function would not have been called.
2. if no specific `check` routine is found, the result for a call of `check` will be `FALSE`, since in this case the default function is called which will return `FALSE` in any case.

Author(s)

Thorn Thaler

`interval` *Interval Class*

Description

`interval` constructs an object of class `interval` representing an interval.

`liesWithin` checks if a number lies within a given interval.

Usage

```
interval(lower, upper, left=c(">=", ">"), right=c("<=", "<"))

liesWithin(x, int)
```

Arguments

lower	the lower boundary of the interval. Can be set to -Inf.
upper	the upper boundary of the interval. Can be set to Inf.
left, right	a comparison symbol. Must be one of (">=", ">") for left and ("<=", "<") for right, respectively. Determines whether the boundary values are included in the interval or not. The default is ">=" and "<=", respectively.
x	a numeric vector or array giving the numbers to be checked.
int	an interval object.

Value

interval returns an object of class interval containing the following components:

lower	the lower boundary of the interval
upper	the upper boundary of the interval
left	the left comparison operator
right	the right comparison operator

liesWithin returns TRUE if the given number lies within the interval and FALSE otherwise.

Author(s)

Thorn Thaler

Examples

```
i <- interval(-3, 3, left=">")

liesWithin(-3:5, i)
```

---

isInteger *Test For Integrity*

Description

isInteger tests if a given number is an integer.

Usage

```
isInteger(n, tol = .Machine$double.eps)
```

Arguments

n	a vector or an array of values to be tested.
tol	a numeric value giving the tolerance level.

Details

As opposed to `is.integer` this function tests for integrity of a given value, rather than being of *type* `integer`.

In R integers are specified by the suffix `L` (e.g. `1L`), whereas all other numbers are of class `numeric` independent of their value. The function `is.integer` does not test whether a given variable has an integer value, but whether it belongs to the class `integer`.

In contrast, the function `isInteger` compares the difference between its argument and its rounded argument. If it is smaller than some predefined tolerance level, the variable is regarded as integer.

Value

`TRUE` if the argument `n` has an integer value, `FALSE` otherwise.

Note

The R function `c` concatenates its argument and forms a vector. In doing so, it coerces the values to a common type. Hence, attention has to be paid, because `isInteger` may give some unexpected results in this case. The R command `list`, however, does not coerce its arguments (see the example).

Author(s)

Thorn Thaler

See Also

`is.integer`

Examples

```
# isInteger tests if the _value_ of a variable is an integer
# 'c' as opposed to 'list' coerces its arguments!
isInteger(c("test", 1, 2, 2.1))     # FALSE FALSE FALSE FALSE
isInteger(list("test", 1, 2, 2.1))  # FALSE TRUE TRUE FALSE

class(1L)  # integer
typeof(1L) # integer
class(1)   # numeric
typeof(1)  # double

# is.integer tests if the _class_ of a variable is 'integer'
is.integer(c("test", 1, 2))    # FALSE
is.integer(list("test", 1, 2)) # FALSE
is.integer(1)                  # FALSE
is.integer(1L)                 # TRUE
```

`merge.list` *Merge Two Lists*

Description

`merge.list` merges two lists. If there are identical names in both lists, only the elements of the first list are considered.

Usage

```
## S3 method for class 'list'
merge(x, y = NULL, mergeUnnamed = TRUE, ...)
```

Arguments

`x`	a list of possibly named elements. All of these are in the merged list.
`y`	a list of possibly named elements or any object, which can be coerced to `list`. If an element has a name occuring also in the argument `x`, it will not be included in the merged list to avoid duplicate names. If `NULL`, `x` is returned.
`mergeUnnamed`	logical. If `TRUE` (the default) unnamed elements in the second list are always included.
`...`	arguments to be passed to or from methods.

Details

The purpose of this function is to merge two lists (e.g. argument lists). If a named element is found as well in the first list as in the second, only the value of the element in the first list is considered. One can think of the second list as a list of default values, which should be considered only if they are not set explicitly in the first list.

Unnamed elements in `y` are included in the merged list only if `mergeUnnamed` is `TRUE`.

Value

a list containing all elements of the argument `x` and those of `y` having names not occuring in `x`.

Author(s)

Thorn Thaler

Examples

```
merge(list(a=1, b="test"), list(3, b=2)) # list(a=1, b="test", 3)
merge(list(1), "test")                   # list(1, "test")
merge(list(1), "test", FALSE)            # list(1)
merge(list(1))                           # list(1)
merge(list(1, a=2, b=3), list(2, b=4))   # list(1, a=2, b=3, 2)
merge(list(1), list(2, b=3), FALSE)      # list(1, b=3)
```

```
a <- list(1, 2, 3)
b <- list("a", "b", "c")
names(a)[2] <- names(b)[2] <- "z"
all.equal(merge(a, b), list(1, z=2, 3, "a", "c")) # TRUE
```

---

plotAndSave *Display And Save A Plot*

Description

plotAndSave saves a plot as "pdf", "(e)ps", "jp(e)g", "png", "bmp", "tiff", "emf" and/or "wmf" and additionally displays the plot.

Usage

```
plotAndSave(plot.func, plot.name, ..., folder=getwd(),
         format=c("eps", "pdf"),
         options=list(eps = list(onefile=TRUE, horizontal=FALSE,
                                 paper="special",
                                 width=7, height=7),
                      ps  = list(onefile=TRUE, horizontal=FALSE,
                                 paper="special",
                                 width=7, height=7),
                      pdf = list(onefile=TRUE)),
         do.plot=TRUE, do.return=do.plot)
```

Arguments

plot.func	either a function or a non-empty character string naming the plotting function to be called.
plot.name	a character string (without any suffix such as ".pdf" or ".eps") giving the name of the file where the plot should be saved to.
...	additional arguments to be passed to the plotting function.
folder	a character string giving the name of the folder to which the plot should be saved. The default is the current directory.
format	output format. Must be a subset of ("pdf", "(e)ps", "jp(e)g", "png", "bmp", "tiff", "emf", "wmf"). The latter two can be used only on with a Windows OS. The default is to produce both an eps-file and a pdf-file. Can be abbreviated.
options	named list of options to be passed to the respective device driver. Each entry of the list is an option list for the device corresponding to the name of the list item.
do.plot	logical. If TRUE (the default) the plot is displayed.
do.return	logical. If TRUE the return value of the plotting function is returned. Defaults to the value of the parameter do.plot.

Details

The purpose of this function is to produce a plot on the monitor and to save it to a file simultaneously.

The file name must be given without any file-suffix. Depending on the argument `format` the function then generates the respective file with the appropriate suffix. The path should not be included in the file name, since the location where the files should be saved to is controlled by the parameter `folder`.

The function needs a plotting function to be defined, which actually does the plotting itself. Additional arguments (e.g. further graphical parameters) can be passed to `plotAndSave`, which in turn, passes these arguments down to the plotting function,

The parameters of devices are controlled by the arguments `options`.

Value

the return value of the plotting function.

Note

When using Trellis plots from package **lattice** one has to assure that the plotting function actually *does* the plotting. Since the default behaviour of Trellis plots is just to return the Trellis object, one should wrap the call to the particular **lattice** function in a call of the function `print`. The generic function `print` ensures that the plot is displayed and not just returned as an object.

Author(s)

Thorn Thaler

See Also

`pdf`, `postscript`, `jpeg`, `png`, `bmp`, `tiff`

Examples

```
## Not run:
## Plotting Function
# For 'lattice' graphics:
# WRONG:
# f <- function(x, ...) xyplot(x~sin(x), ...)
# CORRECT:
# f <- function(x, ...) print(xyplot(x~sin(x), ...))

f <- function(x, ...) plot(x, sin(x), col=2, type="l", ...)

# Save the plot as "Sine_Function.pdf" in the current folder
# and add a title to the plot

plotAndSave(f, "Sine_Function", x=seq(-pi, pi, length=100),
            main="Sine-Function", format="pd")


## End(Not run)
```

Index