

Package ‘CatMisc’

December 14, 2018

Title Collection of miscellaneous utility methods

Version 1.0.0

Date 2018-12-14

Author Charles Tilford [aut, cre]

Maintainer Charles Tilford <cran@agent.fastmail.com>

Description Some helper methods, primarily testing for ``defined" objects and capturing fields from regular expressions.

URL <https://github.com/maptracker/CatMisc>

BugReports <https://github.com/maptracker/CatMisc/issues>

Depends R (>= 3.1), crayon

Imports methods

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat

NeedsCompilation no

R topics documented:

allRefClasses	2
CatMisc	3
colorize	3
colorMap	4
colorNameToFunc	5
file.rename2	5
is.def	7
is.empty.field	8
is.something	9
methodHelp	10

myRefClassStuff-class	11
myRefClassThing-class	11
parenRegExp	12
RefClassHelper-class	13
relativePath	14
textBlockToVector	15
thingProduct	16
useCol	16
useColor	17
Index	18

allRefClasses	<i>All Reference Classes</i>
---------------	------------------------------

Description

Return class representations for object and all inherited classes

Usage

allRefClasses(obj, standardClasses = FALSE, struct = NA)

Arguments

- obj Required, the object to be tested. This can be either a class name, a class generator structure, or a RefClass object itself.
- standardClasses Default FALSE. If TRUE then the R-internal utility classes (eg envRefClass) will be included in the output.
- struct Default NA. Used internally for recursion

Details

Recursively follows 'contains' (inherited) reference classes to provide an exhaustive list of all reference classes associated with the provided object

CatMisc

*CAT Miscellaneous Functions***Description**

A collection of utility methods frequently used in packages by Charles Tilford (CAT)

Details

- `is.empty.field` - Test for empty RefClass fields
- `is.def` - Tests a scalar against a variety of "nothing" values
- `is.something` - Tests if object is defined, and not zero, "" or FALSE
- `parenRegExp` - Simplifies capture from regular expression parentheses
- `textBlockToVector` - Splits a block of text into lines
- `.flexFilehandle` - Automatically handles .gz files
- `methodHelp` - Used inside RefClass methods for "self-help"
- `file.rename2` - File rename that can cross device boundaries

colorize

*Colorize***Description**

RefClassHelper object method to colorize a string

Arguments

<code>msg</code>	Default "". A character vector of text to colorize
<code>color</code>	Default NULL. Optional text (eg "yellow") corresponding to the foreground color of the text
<code>bgcolor</code>	Default NULL. Optional text (eg "silver") corresponding to the background color of the text
<code>help</code>	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
```

```
myObject$colorize( help=TRUE )
```

```
myObject$colorize( msg="", color=NULL, bgcolor=NULL )
```

Takes a character vector and applies foreground and/or background color to it.

Value

A character vector with ANSI color codes injected by [crayon](#)

See Also

[crayon](#), [colorNameToFunc](#), https://en.wikipedia.org/wiki/ANSI_escape_code#Colors

Examples

```
el <- RefClassHelper()
x <- el$colorize(c("This", "That"), "green")
x
message(paste(x, collapse=" ... and ... "))
```

colorMap

Map Color

Description

RefClassHelper object method to turn a string into a crayon color function

Arguments

color	Required, a string describing a color, eg "magenta". Can also be a function reference
bg	Default FALSE. If TRUE, will pick the background color corresponding to the name.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$colorMap( help=TRUE )
```

```
myObject$colorMap( color, bg=FALSE )
```

Takes a color name as input, returns a [crayon](#) function. This is primarily a utility function.

Value

A function from the [crayon](#)

See Also

[colorize](#), [crayon](#), [colorNameToFunc](#)

Examples

```
el <- RefClassHelper()
f <- el$colorMap("red")
base::message(f("Quick Brown Fox"))
f <- el$colorMap("cyan", TRUE)
base::message(f("Lazy Dog"))
```

colorNameToFunc	<i>Color Name to Crayon Function</i>
-----------------	--------------------------------------

Description

Function to generate a map of color names to crayon functions

Usage

```
colorNameToFunc()
```

Details

The list generated by this method is simply a lookup list-of-lists used to turn names ("cyan", "black") into [crayon](#) colorizing functions. It is utilized by [colorize](#) to convert the user's color parameters into the appropriate colorizing functions.

The function is run once, the first time the map is requested. Afterwards the resulting list is stored in a local variable for cached retrieval.

Value

A list of lists

See Also

[colorize](#)

file.rename2	<i>File Rename II</i>
--------------	-----------------------

Description

Attempts to move/rename files over device boundaries

Usage

```
file.rename2(from, to)
```

Arguments

from	Required, the current path to the file
to	Required, the path you wish to rename/move the file to

Details

At least some implementations of R apparently use a low-level rename library that will refuse to move files across device boundaries. That is, if the file resides at a path that is on a device different from the destination path, the rename will fail with an error message similar to:

```
In file.rename(fromPath, toPath) : cannot rename file '/mnt/deviceX/foobar.txt' to '/mnt/deviceY/fo
```

Some (many? all?) systems disallow (or simply cannot?) make *hard links* between devices. However, a move/rename is not a link (at least in outcome), so the error is a bit perplexing. The behavior is not limited to R, problems are also seen in Python where the `os.rename` method fails, but `shutil.move` will work (<https://stackoverflow.com/a/15300474>)

Anyhoo. If renames are failing with cross-device link messages, you're probably running into this issue. This method detects the issue (based on a failure to copy and a grepl to "Invalid cross-device link"), and attempts to solve the problem by a copy-then-delete-source mechanism. Doing this task carefully was more complex than initially anticipated. Source on GitHub has more commentary (which get removed if you just evaluate `file.rename2`) on why the code is the way it is.

Value

A single logical value, TRUE for success, FALSE for failure.

See Also

[file.rename](#)

Examples

```
from <- "/mnt/device1/foo.txt"
to   <- "/mnt/device2/foo.txt"
file.rename(from, to) # Fails if /mnt/device1 and /mnt/device2 differ
file.rename2(from, to) # Yay! Works.
```

`is.def`*Is Defined*

Description

Broad test returning false for a variety of "empty" values

Usage

```
is.def(x)
```

Arguments

x	The object to be tested
---	-------------------------

Details

Born out of frustration with diverse possibilities for "not present". The function is NOT vectorized, instead expecting the passed object to be "a single thing".

Value

TRUE if "defined", otherwise FALSE. The following objects are considered "not defined":

- NULL
- Objects that ONLY contain NA
- Matrices with zero columns and zero rows
- Empty fields (TRUE value from [is.empty.field](#))

Examples

```
## TRUE (defined) objects
is.def(FALSE)
is.def(c(""))
noData <- matrix(numeric(), 0, 2,
                 dimnames=list(sample=character(),attr=c("width","weight")))
is.def(noData) # No information, but it has two columns
is.def(c(NA, NA, "b"))

## FALSE (not defined)
is.def(NULL)
is.def(numeric())
is.def(c(NA,NA,NA,NA))
is.def(matrix(numeric(),0,0)) # 0x0 matrix
```

is.empty.field	<i>Is Empty Field</i>
----------------	-----------------------

Description

Tests for ReferenceClass fields that have not been set.

Usage

```
is.empty.field(x, zero.length.empty = FALSE)
```

Arguments

x	The object to be tested
zero.length.empty	Default FALSE, if true then count a zero-length vector as an empty field (returns TRUE)

Details

Some RefClass fields will be automatically set, even if you try not to. For example, simple fields (numeric, character, logical, etc) will be unavaoidably (?) set to a zero-length vector of the appropriate type.

A field defined as 'ANY', however, can not auto-populate. If left unset, it will be represented as a little Slot object of class "uninitializedField". This function simply tests if the provided object inherits that class

Value

TRUE if x inherits class "uninitializedField", otherwise FALSE

Examples

```
## Simple object with two fields, one numeric, the other ANY
foo <- setRefClass("foo", fields = list( x = 'numeric', y = 'ANY' ))
foo$methods( initialize = function(...) {
  ## Don't do anything. This should override (?) the default
  ## $initFields() call.
});

fooObj <- foo()

## $x will be an zero-element numeric vector.
str(fooObj$x)
## It is not considered an empty field:
is.empty.field(fooObj$x)
## ... unless you ask for such cases to test positive:
```



```
is.empty.field(fooObj$x, zero.length.empty=TRUE)

## $y will be a small object:
str(fooObj$y)
## And will report as being an empty field:
is.empty.field(fooObj$y)
```

is.something

Is Something

Description

Check if something is defined, and not zero or an empty string

Usage

```
is.something(x)
```

Arguments

x	The object to be tested
---	-------------------------

Details

Primarily designed to deal with parameter checking for user-supplied arguments

Value

TRUE if x is "defined" ([is.def](#)) and is neither (numeric) zero nor an empty string.

Examples

```
is.something("")
is.something(0)
is.something(c(0,0)) # -> TRUE
```

methodHelp

*Method Help***Description**

Mechanism to identify relevant help topics from calling context

Usage

```
methodHelp(mc, cl, inh = NULL)
```

Arguments

mc	Required, the result of match.call, called just before entering this function. This should allow automatic determination of the method name, as well as the variable name holding the object
cl	Required, the class() of the object
inh	Ignored. Included for compatibility with old calls.

Details

This function is part of an attempt to better document ReferenceClass objects, here focusing on method documentation. The function is designed to be called generically from within a method; See Examples for a fleshed-out illustration. This allows richer documentation to be maintained in Roxygen, and allows objects to be self-documenting if the user passes a help=TRUE flag, eg:

```
myRefClass$methods(
  cube = function( x, help=FALSE ) {
    if (help) return( methodHelp(match.call(), class(.self),
                                names(.refClassDef@contains)) )
    x ^ 3
  })
```

Value

The mysterious 'help_files_with_topic' object R uses for managing internal help (which will be rendered if not captured), or NA if no topic could be found

Examples

```
mrct <- myRefClassThing(5)

# General information on the class as a whole:
?myRefClassThing

# Specific information on the $thingProduct() function
mrct$thingProduct( help=TRUE )
```

myRefClassStuff-class *Example Inherited RefClass Object*

Description

A tiny object designed to be inherited by another tiny object

Details

This is a toy ReferenceClass (aka 'R5') object that is used by another toy class ([myRefClassThing](#)), and is used for tests of [allRefClasses](#).

Fields

txt An interesting text value

Methods

initialize(useColor = TRUE, ...) Create a new RefClassHelper Reference Class object
thingText(prefix = "[Thing Text]", help = FALSE) Reports the text held by the thing

See Also

[myRefClassThing](#), [allRefClasses](#)

myRefClassThing-class *Example Reference Class Object*

Description

A tiny object that multiplies things

Details

This is a toy ReferenceClass (aka 'R5') object used to illustrate the [methodHelp](#) function's use in documenting object methods. It is also utilized by tests for [methodHelp](#).

Fields

x A numeric value

Methods

initialize(txt = "Initial text") Initialize a new myRefClassStuff object
thingProduct(y = 7, help = FALSE) Multiplies the x field by parameter y

See Also[methodHelp](#)**Examples**

```
x <- myRefClassThing( x=17 )
# Help at the method level
x$thingProduct( help=TRUE )
```

parenRegExp

Parenthetical Regular Expression

Description

Extract values from parenthetical capture blocks in regular expressions

Usage

```
parenRegExp(RegExp, text, ignore.case = TRUE, unlist = TRUE)
```

Arguments

RegExp	The regular expression to use
text	The string(s) to test
ignore.case	Default TRUE, which will perform matches case insensitively.
unlist	Default TRUE, which will unlist the results. A string that does not match will return a single NA. If more than one capture field is defined, and at least one string fails to match, then the returned, unlisted vector will be ragged. If you are matching multiple strings, set unlist=FALSE.

Details

RegExp in R is not fully fleshed out. This implements a hack suggested in the internal documentation to allow recovery of text from multiple parenthetical captures

Value

A character vector representing matched values, or NA if no match was found. If unlist is set to FALSE, then a list of character vectors, one list element for each value in the submitted text.

Examples

```

codes <- c("Launch code: 0000", "Bro code", "Locker code = 321203")
extractor <- '([A-Z]+).+?(\\d+)'
parenRegExp( extractor, codes, unlist = FALSE )

header <- ">PK139-beta  Alien infection mediator (San Antonio serotype) "
fastaRE <- '^>(\\S+)\\s*(.*?)\\s*$'
parenRegExp( fastaRE, header )

```

RefClassHelper-class *Reference Class Helper Class*

Description

A utility class that provides help functions when inherited

Details

This class is not designed to be used on its own, but rather to be inherited by another Reference Class.

Fields

useCol Default TRUE. Logical flag to indicate if color should be used in messaging
varName Extracted variable name of the object
varCheck Last time an attempt was made to extract varName

Methods

annotateFields(help = FALSE) Update object fields to include attributes with brief descriptions
colorize(msg = "", color = NULL, bgcolor = NULL, help = FALSE) Use crayon to add ANSI color codes to text
colorMap(color, bg = FALSE, help = FALSE) Convert a color name into a crayon colorizing function
fieldDescriptions(help = FALSE) A static list of brief descriptions for each field in this object
getFieldDescriptions(help = FALSE) Recursively process an object's fieldDescriptions() return value
help(color = NULL, help = FALSE) Display high-level help about all object methods
initialize(useColor = TRUE, ...) Create a new RefClassHelper Reference Class object
showHelp(sections = list(), genericName = "myObject", color = NULL, generic = FALSE, help = FALSE) Construct text for the help() method
useColor(newval = NULL, help = FALSE) Get or set the flag determining if messages are colorized

relativePath	<i>Relative Path</i>
--------------	----------------------

Description

Reports the relative file path from a parent directory to a child object

Usage

```
relativePath(parent, child, mustWork = FALSE, normChild = TRUE)
```

Arguments

parent	Required, the file path that presumably is an ancestor of the child in the directory structure. To return a non-NA value, this object presumably needs to resolve to a directory.
child	Required, the file path of the "deeper" object (can be any component of the file system - file, directory, link, etc.
mustWork	Default FALSE. Passed to <code>normalizePath</code> , set to TRUE if you wish to assure that both child and parent exist.
normChild	Default TRUE, which will cause the child path to be normalized as well. This is not always desirable; For example, <code>normalizePath</code> will convert links to their ultimate target path. If you wish to leave links as-is, set <code>normChild</code> to FALSE.

Details

Given 'child' and 'parent' file paths, return the relative path needed to reach the child from the parent, or NA if the child is not a descendant of the parent.

By default, neither child nor parent will be checked for existence, or if they are an appropriate object. Both will have their paths normalized via `normalizePath()`. If you wish to force existence of both, set `mustWork=TRUE`.

Value

If either child or parent are any of NULL, NA or an empty string, then NA. If child is the same as parent (after normalization), an empty string. If child is not a descendant of the parent, NA. In all other cases, a single string representing the relative path.

See Also

[normalizePath](#)

Examples

```
relativePath("/tmp/RtmpaacRRB", "/tmp/RtmpaacRRB/output.txt")
relativePath(file.path(Sys.getenv('HOME'), "data"), "~/data/plots/x.png")
relativePath("/bin/bang/boom", "/bin/etc/etc/etc.txt")
relativePath("/usr/bin", "")
```

textBlockToVector	<i>Text Block to Vector</i>
-------------------	-----------------------------

Description

Convert a multi-line string to a vector of lines

Usage

```
textBlockToVector(x, split = "[\\n\\r]", trim.white = TRUE,
  skip.empty = TRUE)
```

Arguments

x	The block of text to parse
split	Default "[\\n\\r]+", the regular expression used to split out lines
trim.white	Default TRUE, which will cause leading and trailing whitespace to be removed from each line
skip.empty	Default TRUE, which will remove elements that are the empty string ('')

Details

When embedding static lists of text in code I find it sometimes easier to read and maintain the list if it is encoded as a simple block of text. This function will break such a block into a vector

Value

A character vector, each element being a line

Examples

```
myBlock <- "
A quick brown fox
  Five golden rings
Klaatu barada nikto
"

textBlockToVector(myBlock)
```

thingProduct	<i>Thing Product</i>
--------------	----------------------

Description

A toy object method used to illustrate [methodHelp](#)

Arguments

y The second number

Details

Multiplies the internally-stored value of x by a supplied second number.

Value

A numeric product of x * y

Examples

```
mrct <- myRefClassThing(5)
mrct$thingProduct(11)
```

useCol	<i>RefClassHelper Use Color Flag</i>
--------	--------------------------------------

Description

Internal RefClass field indicating if output should be colorized

Details

A logical flag, if TRUE it indicates that messages (from compliant methods) should be colorized with the [crayon](#) package.

```
## NORMALLY YOU WILL NOT WANT TO ACCESS THIS FIELD DIRECTLY
## Instead, use the \link{useColor} method to check and alter the value
```

Value

A single logical value

See Also

[useColor](#), [message](#)

Examples

```
rch <- RefClassHelper( )
message(rch$colorize("I'm feeling blue", "blue"))
rch$useColor(FALSE)
message(rch$colorize("I'm still down, I'm just not showing it", "blue"))
```

useColor

Use Color

Description

RefClassHelper object method to get/set colorization flag

Arguments

newval	Default NULL. If provided and can be made logical, will set the flag. Inability to cast as logical will emit a non-fatal error.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
#' ## Method Usage:
myObject$useColor( help=TRUE )

myObject$useColor( newval=NULL )
```

Gets or sets the flag determining if messages should be colorized

Value

A single logical value, invisibly

See Also

[message](#), [colorize](#), [useCol](#)

Examples

```
rch <- RefClassHelper()
## Show the object:
rch
## Turn off colorization, show the object again:
rch$useColor(FALSE)
rch
```

Index

`allRefClasses`, [2](#), [11](#)

`CatMisc`, [3](#)

`colorize`, [3](#), [4](#), [5](#), [17](#)

`colorMap`, [4](#)

`colorNameToFunc`, [4](#), [5](#)

`crayon`, [4](#), [5](#), [16](#)

`file.rename`, [6](#)

`file.rename2`, [5](#)

`is.def`, [7](#), [9](#)

`is.empty.field`, [7](#), [8](#)

`is.something`, [9](#)

`message`, [17](#)

`methodHelp`, [10](#), [11](#), [12](#), [16](#)

`myRefClassStuff`
 (`myRefClassStuff-class`), [11](#)

`myRefClassStuff-class`, [11](#)

`myRefClassThing`, [11](#)

`myRefClassThing`
 (`myRefClassThing-class`), [11](#)

`myRefClassThing-class`, [11](#)

`normalizePath`, [14](#)

`parenRegExp`, [12](#)

`RefClassHelper` (`RefClassHelper-class`),
 [13](#)

`RefClassHelper-class`, [13](#)

`relativePath`, [14](#)

`textBlockToVector`, [15](#)

`thingProduct`, [16](#)

`useCol`, [16](#), [17](#)

`useColor`, [17](#), [17](#)