Package 'CatMisc'

March 11, 2019

Title Collection of miscellaneous utility methods

Version 1.1.2

Date 2019-01-11

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)

Imports methods, crayon

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests testthat

NeedsCompilation no

R topics documented:

`allRefClasses` *All Reference Classes*

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

msg	Default "". A character vector of text to colorize
color	Default NULL. Optional text (eg "yellow") corresponding to the foreground color of the text
bgcolor	Default NULL. Optional text (eg "silver") corresponding to the background color of the text
help	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 foregrand 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 *Color Name to Crayon Function*

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

dotFlexFilehandle *Flexible Filehandle*

Description

Utility method to transparently handle 'normal' and compressed files

Usage

```
.flexFilehandle(file)
```

Arguments

`file`	Required, the path to the file to read

Details

If the file ends with 'gz', then `gzfile` will be used to open a file handle. Otherwise, `file` will be used.

Value

A list with filehandle ("fh"), basename ("name"), suffix ("sfx") and gzip flag ("gz")

`dotSelfVarName` *Get Name of Current Variable*

Description

For an object, get the variable name being used by R to 'hold' it

Arguments

`def`	Default `"myObj"`, a default name that can be used if the actual name can not be found.
`fallbackVar`	Default `""`. Another default name, will be used in preference to `def` if the real name can not be found. There are ... reasons ... for this. I can't remember what they were and am too timid to simplify the function at this point.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$.selfVarName( def="myObj", fallbackVar="" )
```

Returns the variable name of an object. That is, if you have an object `foo`, calling `foo$.selfVarName` should return `"foo"`.

This functionality is used to generate self-referential help text that can be copied-and-pasted to allow for easy exploration of the object.

Value

A single character string

See Also

varName, help, varCheck, getFieldDescriptions

Examples

```
rch <- RefClassHelper()
rch$.selfVarName()
identical("rch", rch$.selfVarName())
```

fieldDescriptions *Field Descriptions*

Description

Get brief descriptions for all RefClass fields used by the object

Arguments

help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$fieldDescriptions( )
```

This method simply returns a list of descriptive text for each Reference Class field defined for the object. It is intended to provide self-documenting information to the help function.

If the object inherits other RefClass classes, they will be recursively queried for descriptive text associated in their own `fieldDescriptions` structures. This recursion is managed by getFieldDescriptions.

Value

A list of character vectors, each a single entry long, each (ideally) being a brief description of the field.

See Also

getFieldDescriptions, help, helpSections

Examples

```
el <- RefClassHelper()
el$fieldDescriptions()
el$help() # Descriptive text will be at bottom
```

file.rename2 *File Rename II*

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
```

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.
```

`getFieldDescriptions` *Get Field Descriptions*

Description

Get brief descriptions for all object fields

Arguments

help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$getFieldDescriptions( )
```

Returns a list, with names representing object fields and values being descriptions of each one. The descriptions need to be defined in the object's fieldDescriptions method. If inheritance is being used, they will be recovered recursively for each parent class. The function is used by the help method.

Value

A list of character strings

See Also

fieldDescriptions, getHelpSections, help

Examples

```
rch <- RefClassHelper()
rch$getFieldDescriptions()
rch$help() # Will be shown at the end of the help
```

---

getHelpSections *Get Help Sections*

Description

Get a list of all notable object methods, organized into sections

Arguments

help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$getHelpSections( )
```

Returns a list, with names representing conceptual sections and values being object methods assigned to those sections. This structure needs to be defined in the object's helpSections method. If inheritance is being used, sections will be recovered recursively for each parent class. The function is used by the help method.

Value

A list of character strings

See Also

helpSections, getFieldDescriptions, help

Examples

```
rch <- RefClassHelper()
rch$getHelpSections()
rch$help() # Will be used near the top to organize methods
```

`help` *Object Help*

Description

Summarize object methods, fields and commands to get additional help

Arguments

`genericName`	Default 'myObject'. Used if R can not determine the actual variable name of the object (via .selfVarName)
`color`	Default `NULL`, which will use the value defined by useColor. If TRUE, then output will be colorized.
`generic`	Default `FALSE`. If TRUE, then generic Reference Class methods common to all RefClass objects will be included.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$help( genericName='myObject', color=NULL, generic=FALSE )
```

Shows help for the object. This method attempts to both be compact and exhaustive. Each object method will be shown, as well as a one-line description of its function (if it has been defined in the code). Additionally, the method call using the help flag will be shown explicitly with the actual object variable. That is, if `$help()` is called on an object named `foo` that has a method named `soak`, then the reported text will include:

```
foo$soak( help=TRUE )
```

This is to aid rapid copy-paste exploration of the object.

Object fields will also be reported, along with their descriptive text.

Value

NULL, invisibly

See Also

getFieldDescriptions, getHelpSections, show

Examples

```
rch <- RefClassHelper()
## Report help for the object
rch$help()
rch$help( generic=TRUE ) # Include generic methods
```

`helpSections` *Help Sections*

Description

Get a list that organizes methods into conceptually-related sections

Arguments

`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$helpSections( )
```

This method simply returns a list that organizes the methods used by the Reference Class object into related sections. It is intended to provide self-documenting information to the help function.

If the object inherits other RefClass classes, they will be recursively queried for their own sections. This recursion is managed by getHelpSections.

Value

A list of character vectors, each containing one or more methods within the section defined by that entry's name.

See Also

getHelpSections, help, fieldDescriptions

Examples

```
el <- RefClassHelper()
el$helpSections()
el$help() # Methods will be broken out into the above sections
```

initialize *Initialize RefClass Object*

Description

R internal method to create a new Reference Class instance

Details

The `$initialize()` object method is one of the reserved Reference Class functions. It executes whenever you generate a new object (that is, if a class 'foo' is defined, when you run `x <- foo()`). You should not call `$initialize()` directly.

In many cases you may not need to define such a method. I have found, however, that if you are attempting RefClass inheritance that the method can be useful. Some suggestions when creating a method:

- Make use of ... in the parameters.
- Be sure to call `callSuper(...)` to invoke the `initialize` for parent classes
- This is a reasonable place to define default field values, expose them for parameterized customization, and put in sanity-checking
- Using the same variable name for parameters and fields works, but can be confusing.
- Do not forget to set fields with <<-

Value

The newly created RefClass object

Examples

```
## Using one of the built-in toy objects:
rct <- myRefClassThing() # There. That was it. It ran transparently.
```

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 *Is Empty Field*

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.

Value

A character vector

Fields

`txt` An interesting text value

Methods

`fieldDescriptions(help = FALSE)` A static list of brief descriptions for each field in this object

`helpSections(help = FALSE)` Static list organizing object methods into conceptual sections

`initialize(useColor = TRUE, ...)` Create a new RefClassHelper Reference Class object

`show(help = FALSE)` Pretty-print the object

`thingText(prefix = "[Thing Text]", help = FALSE)` Reports the text held by the thing

See Also

myRefClassThing, allRefClasses

Examples

```
mrcs <- myRefClassStuff(txt="Hello World")
mrcs
mrcs$help()
```

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

`fieldDescriptions(help = FALSE)` A static list of brief descriptions for each field in this object

`helpSections(help = FALSE)` Static list organizing object methods into conceptual sections

`initialize(txt = "Initial text", ...)` Initialize a new myRefClassStuff object

`show(...)` Provide pretty-printed summary for RefClassStuff objects

`thingProduct(y = 7, help = FALSE)` Multiplies the x field by parameter y

See Also

methodHelp

Examples

```
mrct <- myRefClassThing( x=17, txt="Text is managed by Stuff" )
# Help at the object level:
mrct$help()
# Help at the method level:
mrct$thingProduct( help=TRUE )

# This class's method:
mrct$thingProduct(3)
# Method from the parent class (myRefClassStuff):
mrct$thingText(prefix="[Foo]")
# Method from the grandparent class (RefClassHelper)
message(mrct$colorize("This text is red", color='red'))
```

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 that 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 )
```

print.rchField *RefClassHelper Field Pretty Printer*

Description

S3 print class for annotated RefClassHelper fields

Usage

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

Arguments

x	Required, the object to be printed
...	To make the build shut up about "S3 generic/method consistency" : https://stackoverflow.com/a/14237

Details

If a RefClassHelper object has annotated its fields with descriptive metadata (via the $annotate-Fields() method, which is generally automatically called by the $help() method), the fields will be classed to allow this function to run. It just makes the description and help guidance a little nicer

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

`getHelpSections(help = FALSE)` Recursively process an object's helpSections() return value

`help(genericName = "myObject", color = NULL, generic = FALSE, rm.skip = TRUE, help = FALSE)` Construct text for the help() method

`helpSections(help = FALSE)` Static list organizing object methods into conceptual sections

`initialize(useColor = TRUE, ...)` Create a new RefClassHelper Reference Class object

`useColor(newval = NULL, help = FALSE)` Get or set the flag determining if messages are colorized

Examples

```
rch <- RefClassHelper()
rch        # Summary text
rch$help() # Detailed help text
message(rch$colorize("This is blue", color="blue"))
rch$useColor(FALSE)
message(rch$colorize("This is no longer blue", color="blue"))
```

---

relativePath *Relative Path*

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 normalizePath, 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, normalizePath will convert links to their ultimate target path. If you wish to leave links as-is, set normChild 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 existance, or if they are an appropriate object. Both will have their paths normalized via `normalizePath()`. If you wish to force existance 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 *Text Block to Vector*

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` *Thing Product*

Description

A toy object method used to illustrate methodHelp

Arguments

`y`	The second number
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$thingProduct( y=7 )
```

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)
```

thingText *Thing Text*

Description

Get text for toy (demonstration) Reference Class object

Arguments

prefix	Default '[Thing Text]'. A prefix to include before the actual Thing text.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

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

myObject$thingText( prefix="[Thing Text]" )
```

Returns the thing's text, with a prefix

Value

A character vector

Examples

```
mrcs <- myRefClassStuff(txt="Some text for the thing")
## Query the text field directly
mrcs$txt
## Access the text by this method
mrcs$thingText(prefix="[Hooray!]")
# Method from the parent class (RefClassHelper)
message(mrcs$colorize("This text is blue", color='blue'))
```

useCol *RefClassHelper Use Color Flag*

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
```

varCheck *RefClassHelper Variable Name Last Checked*

Description

Internal RefClass field tracking the last time the object name was checked

Details

The varName field caches the variable name of the object. Finding this value sometimes fails (not sure why). It is generally possible to eventually recover the name. The varCheck field tracks the last attempt to get the name, and restricts retries to once every 10 seconds. function.

See Also

varName, .selfVarName, help, getFieldDescriptions

Examples

```
rch <- RefClassHelper( )
rch$varCheck # Not attempted yet
rch # Pretty-print the object, which will attempt to discover its name
Sys.time()
rch$varCheck # Should be close to the time above
```

varName *RefClassHelper Variable Name*

Description

Internal RefClass field holding its own variable name

Details

A character string containing the name of the object. That is, an object invoked with the R variable x should have "x" as the value for this field. It is set using the .selfVarName function.

See Also

.selfVarName, help, varCheck, getFieldDescriptions

Examples

```
rch <- RefClassHelper( )
rch$varName # Should initially be unset
rch # Pretty-print the object, which will attempt to discover its name
rch$varName # Should now be "rch"
```

Index