# Package ‘CatMisc’

January 26, 2018

**Title** Collection of miscellaneous utility methods

**Version** 0.8.0

**Date** 2017-11-20

**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

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1

**Suggests** testthat

**NeedsCompilation** no

## R topics documented:

## `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 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/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 *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)
```

## 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` | The inherited packages, taken from .refClassDef@contains |

## 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 )
```

---

`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

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

## See Also

methodHelp

## Examples

```
# Help at the class level
myRefClassThing( help=TRUE )

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 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 )
```

---

## 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 |

### 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)
```

# Index