Package ‘pkgmaker’

February 20, 2015

Type Package

Title Package development utilities

Version 0.22

Date 2013-09-17

Author Renaud Gaujoux

Maintainer Renaud Gaujoux <renaud@tx.technion.ac.il>

Description This package provides some low-level utilities to use for package
development. It currently provides managers for multiple package specific
options and registries, vignette, unit test and bibtex related utilities.
It serves as a base package for packages like NMF, RcppOctave, doRNG, and
as an incubator package for other general purposes utilities, that will
eventually be packaged separately.
It is still under heavy development and changes in the interface(s) are
more than likely to happen.

License GPL (>= 2)

URL https://renozao.github.io/pkgmaker

BugReports http://github.com/renozao/pkgmaker/issues

SCM github:renozao, r-forge

LazyLoad yes

Depends R (>= 3.0.0), stats, registry

Imports methods, tools, codetools, digest, stringr, xtable, grDevices

Suggests devtools (>= 0.8), bibtex, RUnit, testthat, knitr,
ReportingTools, hwriter, argparse

‘CLI.R’ ‘knitr.R’ ‘repositories.R’

VignetteBuilder knitr

NeedsCompilation no

1
Repository  CRAN  
Date/Publication  2014-05-14 16:18:51

R topics documented:


topics documented:
  addnames .................................................. 3  
  addToLogger ............................................. 4  
  add_lib .................................................. 5  
  alphacol ............................................... 6  
  cgetAnywhere ........................................... 6  
  checkWarning ........................................... 7  
  citecmd ................................................ 7  
  CLIArgumentParser ..................................... 8  
  compile_src .......................................... 9  
  exitCheck ............................................. 9  
  expand_list ........................................... 10  
  ExposeAttribute ....................................... 12  
  extractLocalFun ....................................... 13  
  file_extension ....................................... 13  
  getLoadingNamespace .................................. 14  
  graphics-utils ........................................ 15  
  hasArg2 ................................................ 15  
  hasEnvar ............................................. 16  
  install.dependencies .................................. 16  
  inSweave .............................................. 17  
  isCRANcheck ........................................... 18  
  isManualVignette ...................................... 19  
  is_something .......................................... 21  
  knit_ex ................................................. 22  
  latex_preamble ........................................ 25  
  list.libs ............................................... 26  
  makeFakeVignette ...................................... 27  
  makeUnitVignette ...................................... 27  
  mkoptions ............................................. 28  
  new2 .................................................. 29  
  oneoffVariable ......................................... 29  
  onLoad ................................................. 30  
  option_symlink ....................................... 31  
  orderVersion .......................................... 32  
  packageCLI ............................................ 33  
  packageData .......................................... 34  
  packageDependencies .................................. 35  
  packageEnv ............................................. 35  
  packageReference ..................................... 37  
  packageReferenceFile ................................... 38  
  packageRegistry ....................................... 38  
  packageTestEnv ........................................ 40
addnames

Generating Names

Description

Generates names or dimnames for objects.

Usage

addnames(x, ...)

## Default S3 method:
addnames(x, ...)

## S3 method for class 'vector'
addnames(x, prefix = "x", sep = ",
...)

## S3 method for class 'array'
addnames(x,
  prefix = letters[1:length(dim(x))], sep = ",
...)

## S3 method for class 'matrix'
addnames(x, prefix = c("row", "col"),
...)

Arguments

x object whose names are generated.
prefix prefix string to use. A vector can be used to specify a prefix for each dimension of x. Names are build as <prefix><sep><index>.
sep separator used between the prefix and the numeric index.
... extra arguments to allow extension and passed to the next method.

addToLogger

Enhancing RUnit Logger

Description

Adds a function or a local variable to RUnit global logger.

Usage

addToLogger(name, value, logger = NULL)

Arguments

name name of the function or variable to add
value object to append to the logger. If value is a function it is added to the list and is accessible via .testLogger$name. If value is a variable it is added to the local environment and is therefore accessible in all logging functions.
logger an optional RUnit logger object. If missing or NULL, the object .testLogger is searched in .GlobalEnv – and an error is thrown if it does not exist.

Value

the modified logger object. Note that the global object is also modified if logger is NULL.
Description
Prepend/append paths to the library path list, using `.libPaths`.

Usage
```
add_lib(..., append = FALSE)
```

Arguments
- `...`: paths to add to `.libPath`
- `append`: logical that indicates that the paths should be appended rather than prepended.

Details
This function is meant to be more convenient than `.libPaths`, which requires more writing if one wants to:
- sequentially add libraries;
- append and not prepend new path(s);
- keep the standard user library in the search path.

Examples
```
ol <- .libPaths()
# called sequentially, .libPaths only add the last library
show(.libPaths('.'))
show(.libPaths(tempdir()))
# restore
.libPaths(ol)

# .libPaths does not keep the standard user library
show(.libPaths())
show(.libPaths('.'))
# restore
.libPaths(ol)

# with add_lib
show( add_lib('.'))
show( add_lib(tempdir()))
show( add_lib('..', append=TRUE))

# restore
.libPaths(ol)
```
alphacol  

Colour utilities

Description

alphacol adds an alpha value to a colour specification and convert to a hexadecimal colour string.

Usage

alphacol(col, alpha = FALSE)

Arguments

col vector of any of the three kinds of \( \mathbb{R} \) color specifications, i.e., either a color name (as listed by \texttt{colors()}) , a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see \texttt{rgb}) , or a positive integer \( i \) meaning \texttt{palette()}[i].

alpha logical value indicating whether the alpha channel (opacity) values should be returned.

Examples

```r
# Alphas
alphacol('red') # do nothing
alphacol('red', 10)
alphacol('#aabbcc', 5)
alphacol(4, 5)
```

cgetAnywhere  

Get Anywhere

Description

Similar to \texttt{getAnywhere}, but looks for the value of its argument.

Usage

cgetAnywhere(x)

Arguments

x a single character string
checkWarning

Extra Check Functions for RUnit

Description

checkWarning checks if a warning is generated by an expression, and optionally follows an expected regular expression pattern.

Usage

checkWarning(expr, expected = TRUE, msg = NULL)

Arguments

- `expr`: an R expression
- `expected`: expected value as regular expression pattern. If a logical, then it specifies if a warning is expected or not. For backward compatibility, a NULL value is equivalent to TRUE.
- `msg`: informative message to add to the error in case of failure

Examples

```r
# check warnings
checkWarning({ warning('ah ah'); 3})
checkWarning({ warning('ah oh ah'); 3}, 'oh')
try( checkWarning(3) )
try( checkWarning({ warning('ah ah'); 3}, 'warn you') )
```

citecmd

Citing Package References

Description

Create a citation command from package specific BibTex entries, suitable to be used in Rd files or Latex documents. The entries are looked in a file named REFERENCES.bib in the package’s root directory (i.e. inst/ in development mode).

Usage

citecmd(key, ..., REFERENCES = NULL)
Arguments

key character vector of BibTex keys
... extra arguments passed to format.bibentry.
REFERENCES package or bibentry specification

Value

a character string containing the text formatted BibTex entries

CLIArgumentParser Enhanced Command Line Argument Parser

Description

Extends the capabilities of package argparse, e.g., in defining sub commands.

Usage

CLIArgumentParser(prog = CLIfile(), description = "",
..., epilog = "", show.defaults = TRUE)

parseCMD(parser, ARGS = commandArgs(TRUE), debug = FALSE,
envir = parent.frame())

Arguments

prog program name
description program description
... extra arguments passed to ArgumentParser.
epilog epilog messages to display at the end of the man pages
show.defaults logical that indicates if default argument values should be displayed.
parser parser object as returned by CLIArgumentParser.
ARGS command line argument to parse, as a named list or a character string.
debug logical that indicate if debugging information should be printed.
envir environment that contains where the sub-command functions are looked for.
**compile_src**

*Compile Source Files from a Development Package*

**Description**

Compile Source Files from a Development Package

**Usage**

```
compile_src(pkg = NULL, load = TRUE)
```

**Arguments**

- `pkg` : the name of the package to compile
- `load` : a logical indicating whether the compiled library should be loaded after the compilation (default) or not.

**Value**

None

---

**exitCheck**

*Exit Error Checks*

**Description**

`exitCheck` provides a mechanism to distinguish the exit status in `on.exit` expressions.

**Usage**

```
exitCheck()
```

**Details**

It generates a function that is used within a function’s body to “flag” normal exits and in its `on.exit` expression to check the exit status of a function. Note that it will correctly detect errors only if all normal exit are wrapped into a call to it.
Examples

```r
# define some function
f <- function(err){

  # initialise an error checker
  success <- exitCheck()

  # do something on exit that depends on the error status
  on.exit({
    if(success()) cat("Exit with no error: do nothing\n")
    else cat("Exit with error: cleaning up the mess ...\n")
  })

  # throw an error here
  if( err ) stop('There is an error')

  success(1+1)
}

  # without error
  f(FALSE)
  # with error
  try( f(TRUE) )
```

---

**Description**

`expand_list` expands a named list with a given set of default items, if these are not already in the list, partially matching their names.

`expand_dots` expands the `...` arguments of the function in which it is called with default values, using `expand_list`. It can only be called from inside a function.

**Usage**

```r
expand_list(x, ..., .exact = TRUE, .names = !.exact)
```

```r
expand_dots(..., .exclude = NULL)
```

**Arguments**

- `x` : input list
- `...` : extra named arguments defining the default items. A list of default values can also be passed as a single unnamed argument.
**expand_list**

.logical that indicates if the names in `x` should be partially matched against the
defaults.

.logical that only used when `.exact` = `FALSE` and indicates that the names of items
in `x` that partially match some defaults should be expanded in the returned list.

optional character vector of argument names to exclude from expansion.

**Value**

a list

**Examples**

```r
#-------
# expand_list
#-------
expand_list(list(a=1, b=2), c=3)
expand_list(list(a=1, b=2, c=4), c=3)
# with a list
expand_list(list(a=1, b=2), list(c=3, d=10))
# no partial match
expand_list(list(a=1, b=2, c=5), cd=3)
# partial match with names expanded
expand_list(list(a=1, b=2, c=5), cd=3, .exact=FALSE)
# partial match without expanding names
expand_list(list(a=1, b=2, c=5), cd=3, .exact=FALSE, .names=FALSE)

# works also inside a function to expand a call with default arguments
f <- function(...){
  cl <- match.call()
  expand_list(cl, list(a=3, b=4), .exact=FALSE)
}
f()
f(c=1)
f(a=2)
f(c=1, a=2)

#-------
# expand_dots
#-------
# expanding dot arguments

f <- function(...){
  expand_dots(list(a=2, bcd='a', xxx=20), .exclude='xxx')
}

# add default value for all arguments
f()
# add default value for `bcd` only
f(a=10)
# expand names
```
Exposing Object Attributes

Description

The function `exposeattribute` creates an S3 object that exposes all attributes of any R object, by making them accessible via methods `$` and/or `$<-$`. attr_mode and attr_mode<- get and sets the access mode of ExposeAttribute objects.

Usage

```r
exposeattribute(object, ..., .MODE = "rw", .VALUE = FALSE)
attr_mode(x)
attr_mode(x)<-value
```

Arguments

- **object**: any R object whose attributes need to be exposed
- **...**: attributes, and optionally their respective values or access permissions. See argument value of attr_mode for details on the way of specifying these.
- **.MODE**: access mode:
  - "r": (read-only) only method `$` is defined
  - "w": (write-only) only method `$<-$` is defined
  - "rw": (read-write) both methods `$` and `$<-$` are defined
- **.VALUE**: logical that indicates if the values of named arguments in ... should be considered as attribute assignments, i.e. that the result object has these attributes set with the specified values. In this case all these attributes will have the access permission as defined by argument .MODE.
- **x**: an ExposeAttribute object
- **value**: replacement value for mode. It can be NULL to remove the ExposeAttribute wrapper, a single character string to define a permission for all attributes (e.g., 'rw' or 'r'), or a list specifying access permission for specific attributes or classes of attributes defined by regular expressions. For example, list(a='r', b='w', 'blabla.*'='rw') set attribute 'a' as read-only, attribute 'b' as write-only, all attributes that start with 'blabla' in read-write access.
extractLocalFun  

**Extracting Local Function Definition**

**Description**

extractLocalFun Extracts local function from wrapper functions of the following type, typically used in S4 methods: `function(a, b, ...){ .local <- function(a, b, c, d, ...){} .local(a, b, ...) }`

Works for methods that are created (setMethod) as a wrapper function to an internal function named .local.

**Usage**

```r
extractLocalFun(f)
allFormals(f)
```

**Arguments**

- `f` definition of the wrapper function

**Value**

- a function
  - a paired list like the one returned by `formals`.

---

**file_extension  

Extract File Extension**

**Description**

Extract File Extension

**Usage**

```r
file_extension(x, ext = NULL)
```

**Arguments**

- `x` path as a character vector.
- `ext` extension to append instead of the original extension.
Examples

```r
file_extension('alpha.txt')
file_extension(paste('aa.tt', 1:5, sep=''))
# change extension
file_extension(paste('aa.tt', 1:5, sep=''), 'pdf')
file_extension(paste('aatt', 1:5, sep=''), 'pdf')
```

Description

getLoadingNamespace returns information about the loading namespace. It is a wrapper to `loadingNamespaceInfo`, that does not throw an error.

Tests if a namespace is being loaded.

- `isNamespaceLoaded` tests if a given namespace is loaded, without loading it, contrary to `isNamespace`.
- `isDevNamespace` tests the – current – namespace is a devtools namespace.

Dynamically adds exported objects into the loading namespace.

`ns_get` gets an object from a given namespace.

Usage

```r
getLoadingNamespace(env = FALSE, info = FALSE, nodev = FALSE)

isLoadingNamespace(ns, nodev = FALSE)

isNamespaceLoaded(ns)

isDevNamespace(ns)

addNamespaceExport(x)

ns_get(x, ns)
```

Arguments

- `env` logical that indicates that the namespace’s environment (i.e. the namespace itself) should be returned.
- `info` logical that indicates that the complete information list should be returned
- `ns` the name of a namespace or a namespace whose loading state is tested. If missing `isLoadingNamespace` test if any namespace is being loaded.
- `nodev` logical that indicates if loading devtools namespace should be discarded.
Graphics-utils

character vector containing the names of R objects to export in the loading namespace.

Value

the name of the loading namespace if env and info are FALSE, an environment if env=TRUE, a list with elements pkgname and libname if info=TRUE.

Description

Utility Functions for Graphics

mfrow returns a 2-long numeric vector suitable to use in par(mfrow=x), that will arrange n panels in a single plot.

Usage

mfrow(n)

Arguments

n number of plots to be arranged.

Examples

mfrow(1)
mfrow(2)
mfrow(3)
mfrow(4)
mfrow(10)

hasArg2

Checking for Missing Arguments

Description

This function is identical to hasArg, except that it accepts the argument name as a character string. This avoids to have a check NOTE about invisible binding variable.

Usage

hasArg2(name)
Arguments

name the name of an argument as a character string.

Examples

```r
f <- function(...){ hasArg2('abc') }
f(a=1)
f(abc=1)
f(b=1)
```

hasEnvvar Check Environment Variables

Description

Tells if some environment variable(s) are defined.

Usage

```r
hasEnvvar(x)
```

Arguments

x environment variable name, as a character vector.

Examples

```r
hasEnvvar('_R_CHECK_TIMINGS_')
hasEnvvar('ABC_D')
```

install.dependencies Installing All Package Dependencies

Description

Install all dependencies from a package source directory or package source file.

Usage

```r
install.dependencies(pkg = NULL, all = FALSE, ..., dryrun = FALSE)
```
**Arguments**

- **pkg**  
  package path or source file
- **all**  
  logical that indicates if 'Suggests' packages should be installed.
- **...**  
  extra arguments passed to `install.packages`.
- **dryrun**  
  logical that indicates if the packages should be effectively installed or only shown.

**Examples**

```r
try(installdependencies('Matrix', dryrun=TRUE))
## Not run:
installdependencies("mypackage_1.0.tar.gz", dryrun=TRUE)
## End(Not run)
```

---

**inSweave**  
**Identifying Sweave Run**

**Description**

Tells if the current code is being executed within a Sweave document.

**Usage**

`inSweave()`

**Value**

TRUE or FALSE

**Examples**

```r
# Not in a Sweave document
inSweave()

# Within a Sweave document
```
Description

isCRANcheck tries to identify if one is running CRAN-like checks.

isCRAN_timing tells if one is running CRAN check with flag `--timing`.

Currently, isCHECK checks both CRAN expected flags, the value of environment variable `_R_CHECK_RUNNING_UTESTS_`, and the value of option `_R_CHECK_RUNNING_EXAMPLES_`. It will return TRUE if any of these environment variables is set to anything not equivalent to FALSE, or if the option is TRUE. For example, the function `utest` sets it to the name of the package being checked (`_R_CHECK_RUNNING_UTESTS_=<pkname>`), but unit tests run as part of unit tests vignettes are run with `_R_CHECK_RUNNING_UTESTS_=FALSE`, so that all tests are run and reported when generating them.

Usage

```r
isCRANcheck(...)  
isCRAN_timing()  
isCHECK()
```

Arguments

... each argument specifies a set of tests to do using an AND operator. The final result tests if any of the test set is true. Possible values are:

'`timing`' Check if the environment variable `_R_CHECK_TIMINGS_` is set, as with the flag `--timing` was set.

'`cran`' Check if the environment variable `_R_CHECK_CRAN_INCOMING_` is set, as with the flag `--as-cran` was set.

Details

Currently isCRANcheck returns TRUE if the check is run with either environment variable `_R_CHECK_TIMINGS_` (as set by flag `--timings`) or `_R_CHECK_CRAN_INCOMING_` (as set by flag `--as-cran`).

Important: the checks performed on CRAN check machines are – on purpose – not always run with such flags, so there is no guarantee this function effectively identifies such runs. CRAN recommends users rely on custom dedicated environment variables to enable specific tests or examples.

References

Adapted from the function CRAN in the `fda` package.

https://github.com/renozao/roxygen2
Examples

isCHECK()

isManualVignette Identifies Manually Run Vignettes

Description

isManualVignette tells if a vignette is being run through the function runVignette of pkgmaker, allowing disabling behaviours not allowed in package vignettes that are checked via CMD check.

rnw provides a unified interface to run vignettes that detects the type of vignette (Sweave or knitr), and which Sweave driver to use (either automatically or from an embedded command \VignetteDriver command).

as.rnw creates a S3 rnw object that contains information about a vignette, e.g., source filename, driver, fixed included files, etc..

rnwCompiler tries to detect the vignette compiler to use on a vignette source file, e.g., Sweave or knitr.

rnwWrapper tries to detect the type of vignette and if it is meant to be wrapped into another main file.

rnwDriver tries to detect Sweave driver to use on a vignette source file, e.g., SweaveCache, highlight, etc..

rnwIncludes detects fixed includes, e.g., image or pdf files, that are required to build the final document.

rnwChildren detects included vignette documents and return them as a list of vignette objects.

vignetteMakefile returns the path to a generic makefile used to make vignettes.

Compact PDFs using either gs_quality='none' or 'ebook', depending on which compacts best (as per CRAN check criteria).

Usage

isManualVignette()

rnw(x, file = NULL, ..., raw = FALSE)

as.rnw(x, ..., load = TRUE)

rnwCompiler(x, verbose = TRUE)

rnwWrapper(x, verbose = TRUE)

rnwDriver(x)
rnwIncludes(x)
	nrwChildren(x)

vignetteMakefile(package = NULL, skip = NULL,
    print = TRUE, template = NULL, temp = FALSE,
    checkMode = isCHECK() || vignetteCheckMode(),
    user = NULL, tests = TRUE)

compactVignettes(paths, ...)

**Arguments**

- **x**: vignette source file specification as a path or a `rnw` object.
- **file**: output file
- **...**: extra arguments passed to `as.rnw` that can be used to force certain building parameters.
- **raw**: a logical that indicates if the raw result for the compilation should be returned, instead of the result file path.
- **load**: logical to indicate if all the object's properties should loaded, which is done by parsing the file and look up for specific tags.
- **verbose**: logical that toggles verbosity
- **package**: package name. If `NULL`, a DESCRIPTION file is looked for one directory up: this meant to work when building a vignette directly from a package's 'vignettes' sub-directory.
- **skip**: Vignette files to skip (basename).
- **print**: logical that specifies if the path should be printed or only returned.
- **template**: template Makefile to use. The default is to use the file “vignette.mk” shipped with the package `pkgmaker` and can be found in its install root directory.
- **temp**: logical that indicates if the generated makefile should using a temporary filename (TRUE), or simply named “vignette.mk”
- **checkMode**: logical that indicates if the vignettes should be generated as in a CRAN check (TRUE) or in development mode, in which case `pdflatex`, `bibtex`, and, optionally, `qpdf` are required.
- **user**: character vector containing usernames that enforce `checkMode=TRUE`, if the function is called from within their session.
- **tests**: logical that enables the compilation of a vignette that gathers all unit test results. Note that this means that all unit tests are run before generating the vignette. However, unit tests are not (re)-run at this stage when the vignettes are built when checking the package with `R CMD check`.
- **paths**: A character vector of paths to PDF files, or a length-one character vector naming a directory, when all ‘.pdf’ files in that directory will be used.
is_something

Testing Object Type

Description
Testing Object Type

is NA tests if a variable is exactly NA (logical, character, numeric or integer)
is FALSE Tests if a variable is exactly FALSE.
is Number tests if a variable is a single number
is Real tests if a variable is a single real number
is Integer tests if an object is a single integer
is String tests if an object is a character string.
is dir tests if a filename is a directory.
is file tests if a filename is a file.
has Names tests if an object has names.

Usage

is NA(x)
is FALSE(x)
is Number(x)
is Real(x)
is Integer(x)
is String(x, y, ignore.case = FALSE)
is dir(x)
is file(x)
has Names(x, all = FALSE)

Arguments

x an R object
y character string to compare with.
ignore case logical that indicates if the comparison should be case sensitive.
all logical that indicates if the object needs all names non empty
Value

TRUE or FALSE

See Also

isTRUE

knit_ex

Knitr Extensions

Description

knit_ex is a utility function for running small knitr examples, e.g., to illustrate functionalities or issues.

hook_try simply defines a function try in envir that prints the error message if any, and is called instead of base try.

hook_backspace is a chunk hook that enables the use of backspace characters in the output (e.g., as used in progress bars), and still obtain a final output as in the console.

hook_toggle is a chunk hook that adds clickable elements to toggle individual code chunks in HTML documents generated from .Rmd files.

Usage

knit_ex(x, ..., quiet = TRUE, open = FALSE)

hook_try(before, options, envir)

hook_backspace()

hook_toggle()

Arguments

x

... text to knit as a character vector

arguments passed to knit2html or knit

quiet logical that indicates if knitting should be quiet (no progress bars etc..).

open logical, only used when x is in .Rmd format, that indicates if the generated document result should be open in a browse, instead of being printed on screen. Not that a browser will not open in non-interactive sessions, and the result will be returned invisibly.

before logical that indicates when the hook is being called: before or after the chunk is processed.

options list of current knitr chunk options

envir environment where the chunk is evaluated
Value

knit_ex returns the generated code, although invisibly when open=TRUE.

Examples

```r
#-------
# knit_ex
#-------
library(knitr)
knit_ex("1 + 1")

#-------
# hook_try
#-------
library(knitr)

# standard error message is caught
knti_ex("stop('ah ah')")

# with try the error is output on stderr but not caughted by knitr
knti_ex("try( stop('ah ah') )")

# no message caught
knti_ex(""
^^^{r, include = FALSE}
kntiHooks$set(try = pkgmaker::hook_try)
^^^

^^^{r, try=TRUE}
try( stop('ah ah') )
^^^")

#-------
# hook_backspace
#-------
# Correctly formatting backspaces in chunk outputs
tmp <- tempfile(fileext = ".Rmd")
cafile = tmp,
^^^{r, include = FALSE}
library(knitr)
kntiHooks$set(backspace = pkgmaker::hook_backspace())
^^^
Default knitr does not handle backspace and adds a special character:
^^^{r}
cafile(abc\bd)
^^^

Using the hook backspace solves the issue:
^^^{r, backspace=TRUE}
cafile(abc\bd)
```
```r
# knit
out <- knitr::knit2html(tmp, fragment.only = TRUE)
# look at output
## Not run:
browseURL(out)
edit(file = out)

## End(Not run)
# cleanup
unlink(c(tmp, out))

#----------------
# hook_toggle
#----------------
knit_ex("

Declare chunk hook:
```r
\(\text{\{r, setup}\)}
library(knitr)
knot_hooks$set(toggle = hook_toggle())
```

The R code of this chunk can be toggled on/off, and starts visible:
```r
\(\text{\{r, toggle=TRUE}\)}
print(1:10)
```

The R code of this chunk can be toggled on/off, and starts hidden:
```r
\(\text{\{r, toggle=FALSE}\)}
print(1:10)
```

This is a plain chunk that cannot be toggled on/off:
```r
\(\text{\{r}\)}
print(1:10)
```

Now all chunks can be toggled and start visible:
```r
\(\text{\{r, toggle_all}\)}
opts_chunk$set(toggle = TRUE)
```

```r
\(\text{\{r}\)}
sample(S)
```

To disable the toggle link, one can pass anything except TRUE/FALSE:
```r
\(\text{\{r, toggle = NA}\)}
sample(S)
```
```
Description

latex_preamble outputs/returns command definition LaTeX commands to be put in the preamble of vignettes.

latex_bibliography prints or return a LaTeX command that includes a package bibliography file if it exists.

Usage

latex_preamble(PACKAGE, R = TRUE, CRAN = TRUE,
               Bioconductor = TRUE, GEO = TRUE, ArrayExpress = TRUE,
               biblatex = FALSE, only = FALSE, file = "")

latex_bibliography(PACKAGE, file = "")

Arguments

- `R` logical that indicate if general R commands should be added (e.g. package names, inline R code format commands)
- `CRAN` logical that indicate if general CRAN commands should be added (e.g. CRAN package citations)
- `Bioconductor` logical that indicate if general Bioconductor commands should be added (e.g. Bioconductor package citations)
- `GEO` logical that indicate if general GEOmnibus commands should be added (e.g. GEO datasets)
- `ArrayExpress` logical that indicate if general ArrayExpress commands should be added (e.g. URLs to ArrayExpress datasets)
- `biblatex` logical that indicates if a \bibliography command should be added to include references from the package’s REFERENCES.bib file.
- `only` a logical that indicates if the only the commands whose dedicated argument is not missing should be considered.
- `file` connection where to print. If NULL the result is returned silently.
- `PACKAGE` package name

Details

Argument PACKAGE is not required for latex_preamble, but must be correctly specified to ensure biblatex=TRUE generates the correct bibliography command.
Examples

latex_preamble()
latex_preamble(R=TRUE, only=TRUE)
latex_preamble(R=FALSE, CRAN=FALSE, GEO=FALSE)
latex_preamble(GEO=TRUE, only=TRUE)

list.libs

Library Files Utilities

Description

Lists binary library files in a directory

libname extracts library names from a path, removing the directory part of the path, as well as the
platform specific library extension.

Usage

list.libs(dir, ..., all.platforms = FALSE)

libname(x)

Arguments

dir directory

all.platforms a logical that indicates whether to list library files for the current platform only
(default) or all platforms (Unix, Windows, Mac).

... extra arguments passed to list.files.

x a filename

Value

a character vector

Examples

libname('mylib.so')
libname('/some/path/somewhere/mylib.dll')
makeFakeVignette  Generate a Fake Vignette

Description
Generate a Fake Vignette

Usage
makeFakeVignette(src, out, PACKAGE = NULL)

Arguments
src  original Sweave file
out  output file
PACKAGE  package name where to look the source vignette

makeUnitVignette  Make Vignette for Unit Tests

Description
Builds a vignette for unit tests in a package using the utest and a template vignette file.

Usage
makeUnitVignette(pkg,
    file = paste(pkg, "-unitTests.pdf", sep = ""), ...,
    check = FALSE)

Arguments
pkg  Package name
file  Output file (.Rnw, .tex, or .pdf)
...  extra arguments passed to utest.
check  logical that indicates the call was made from R CMD check, in which case the vignette is updated only if results of unit tests can be found in the unit test output directory, where they would have been generated by utest.

Value
Result of running unit test suite
**Description**

`mkoptions` is a function that returns a function that behaves like `options`, with an attached internal/local list of key-value pairs.

`.options` is a low-level function that mimics the behaviour of the base function `options`, given a set of key-value pairs. It is the workhorse function used in `mkoptions` and package-specific option sets (see `setupPackageOptions`)

**Usage**

```r
mkoptions(...)

.options(..., .DATA)
```

**Arguments**

- `...` list of keys or key-value pairs. For `mkoptions` these define initial/default key-value pairs.
- `.DATA` a list or an environment with an element `.options`.

**See Also**

`setupPackageOptions`

**Examples**

```r
f <- mkoptions(a=3, b=list(1,2,3))
str(f())
f('a')
f('b')
str(old <- f(a = 10))
str(f())
f(old)
str(f())
```
Alternative S4 Constructor

Description

An alternative version of `new` to create objects based on a list of values.

Usage

```r
new2(class, ...)
```

Arguments

- `class`: Class name to instantiate
- `...`: extra arguments from which slot values are extracted by exact matching of names.

Examples

```r
setClass('A', contain='character', representation(x='numeric', y='character'))

# identical behaviour with standard calls
identical(new('A'), new2('A'))
identical(new('A', x=1), new2('A', x=1))

# but if passing that are names not slots
identical(new('A'), new2('A', b=1))
identical(new('A', x=1), new2('A', x=1, b=3))
identical(new('A', x=1), new2('A', x=1, b=3))

# standard `new` would coerce first unnamed argument into parent of 'A' (i.e. 'character')
new('A', list(x=1))
new('A', list(x=1, y='other'))

# `new2` rather use it to initialise the slots it can find in the list
identical(new('A', x=1), new2('A', list(x=1)))
identical(new('A', x=1, y='other'), new2('A', list(x=1, y='other')))```

One-off Global Variables

Description

Defines a function that allow to get/assign a global variable whose value is ensured to be reset after each access.
Usage

oneoffVariable(default = NULL)

Arguments

default  
default value to which the global variable is reset after each access. Default is NULL.

Value

a function with one argument (value) that provides get/set access to a global variable. If called with a value, it assigns this value to the global variable. If called with no argument, it returns the current value of the global variable and reset it to its default value – as defined at its creation.

Examples

x <- oneoffVariable(0)  
# returns default value  
x()  
# assign a value  
x(3)  
# get the value  
x()  
# second call returns default value again  
x()

onLoad  

Default Load/Unload Functions

Description

Default Load/Unload Functions

Usage

onLoad(libname = NULL, pkgname, chname = packageName())

onUnload(libpath)

Arguments

libname  
a character string giving the library directory where the package defining the namespace was found.

pkgname  
a character string giving the name of the package.

libpath  
a character string giving the complete path to the package.

cname  
a character string naming a DLL (also known as a dynamic shared object or library) to load.
option_symlink

Examples

```r
# ----------
# onLoad
# ----------
# in a package namespace:
.onLoad <- function(libname=NULL, pkgname){

pkgmaker::onLoad(libname, pkgname)

}

# ----------
# onUnload
# ----------
# in a package namespace:
.onUnload <- function(libpath){

pkgmaker::onUnload(libpath)

}
```

---

**option_symlink**  
option_symlink creates a symbolic link to option x.

**Description**

- `option_symlink` creates a symbolic link to option x.
- `is_option_symlink` tests if x is a symbolic link option.
- `option_symlink_target` returns the end target option of a symbolic link option x.
- `as.package_options` creates an object such as the ones used to store package specific options.
- The method `[[` is equivalent to `options()` or `getOption(...)`; e.g. `obj[['']]` returns the list of options defined in `obj`, and `obj[['abc']]` returns the value of option `'abc'`.
- `packageOptions` provides access to package specific options from a given package that were defined with `setupPackageOptions`, and behaves as the base function `options`.
- `listPackageOptions` returns the names of all option currently defined with `setupPackageOptions`.

**Usage**

```r
option_symlink(x)

is_option_symlink(x, opts)

option_symlink_target(x, opts)
```
## orderVersion

as.package_options(..., defaults = NULL)

```r
## S3 method for class 'package_options'
x[[...]]
```

```r
packageOptions(..., PACKAGE = packageName())
```

listPackageOptions()

### Arguments

- **opts**: a list of options
- **x**: a character string, a list or an object of class `package_options`
- **defaults**: `NULL` or a list of default options with their values.
- **...**: arguments passed to `getOption` (only first one is used).
- **PACKAGE**: a package name

### Value

a character vector (possibly empty).

### Examples

```r
listPackageOptions()
```

---

### orderVersion

**Ordering Version Numbers**

Orders a vector of version numbers, in natural order.

#### Usage

```r
orderVersion(x, decreasing = FALSE)
```

```r
sortVersion(x, ...)
```

#### Arguments

- **x**: a character vector of version numbers
- **decreasing**: a logical that indicates if the ordering should be decreasing
- **...**: extra parameters passed to `orderVersion`
packageCLI

Examples

```r
#--------
# orderVersion
#--------
v <- c('1.0', '1.03', '1.2')
order(v)
orderVersion(v)

#--------
# sortVersion
#--------
sort(v)
sortVersion(v)
```

**packageCLI**  
*Package Specific Command Line Interface*

**Description**

Package Specific Command Line Interface

**Usage**

```r
packageCLI(package, altfile = NULL, local = TRUE,
ARGS = commandArgs(TRUE), ...)
```

**Arguments**

- **package**: package name
- **altfile**: alternative file that defines the main CLI entry point. That is a function named `cli`, which takes the list of parsed command line arguments as its first argument.
- **local**: logical that indicates if the main CLI function should be defined and evaluated in a local environment, or in the user’s Global environment.
- **ARGS**: list of parsed arguments passed to the main CLI function.
- **...**: extra arguments passed to the package’s CLI function.
packageData

Loading Package Data

Description

Loads package data using \texttt{data}, but allows the user to avoid \texttt{NOTE}s for a ‘non visible binding variable’ to be thrown when checking a package. This is possible because this function returns the loaded data.

\texttt{ldata} loads a package data in the parent frame. It is a shortcut for \texttt{packageData(list, \ldots, envir=parent.frame())}.

Usage

\begin{verbatim}
packageData(list, envir = .GlobalEnv, \ldots)
\end{verbatim}

\begin{verbatim}
lData(list, \ldots)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{list} character vector containing the names of the data to load.
  \item \ldots other arguments eventually passed to \texttt{data}.
  \item \texttt{envir} the \texttt{environment} where the data should be loaded.
\end{itemize}

Value

the loaded data.

Examples

\begin{verbatim}
#--------
# packageData
#--------
## Not run: mydata <- packageData('mydata')

#--------
# ldata
#--------
## Not run: # in a package' source => won't issue a NOTE
myfunction function(){
  mydata <- lData('mydata')
}

## End(Not run)
\end{verbatim}
packageDependencies

List Package Dependencies

Description
List Package Dependencies

Usage
packageDependencies(x, all = TRUE, as.list = FALSE, available = NULL)

Arguments
x path to package source directory or file.
all logical that indicates if all dependencies should be returned, or only the required ones.
as.list logical that indicates if the result should be a list with one element per type of dependency.
available a matrix of available packages (as returned by available.packages), from which the dependencies are retrieved. This means that there must be a row for the package x.

packageEnv

Package Development Utilities

Description
packageEnv is a slight modification from topenv, which returns the top environment, which in the case of development packages is the environment into which the source files are loaded by load_all.

topns_name: the top namespace is is not necessarily the namespace where topns_name is effectively called. This is useful for packages that define functions that need to access the calling namespace, even from calls nested into calls to another function from the same package – in which case topenv would not give the desired environment.
topns returns the runtime top namespace, i.e. the namespace of the top calling package, possibly skipping the namespace where topns is effectively called. This is useful for packages that define functions that need to access the calling namespace, even from calls nested into calls to another function from the same package – in which case topenv would not give the desired environment.
packageName returns the current package’s name. It was made internal from version 0.16, since the package utils exported its own packageName function in R-3.0.0.
str_ns formats a package environment/namespace for log/info messages.
packagePath returns the current package’s root directory, which is its installation/loading directory in the case of an installed package, or its source directory served by devtools.

isPackageInstalled checks if a package is installed.

as.package is enhanced version of as.package, that is not exported not to mask the original function. It could eventually be incorporated into devtools itself. Extra arguments in ... are passed to find.package.

Usage

packageEnv(pkg, skip = FALSE, verbose = FALSE)

topns_name(n = 1L, strict = TRUE, unique = TRUE)

topns(strict = TRUE)

packageName(envir = packageEnv(), .Global = FALSE, rm.prefix = TRUE)

str_ns(envir = packageEnv())

packagePath(..., package = NULL, lib.loc = NULL)

isPackageInstalled(..., lib.loc = NULL)

as.package(x, ..., quiet = FALSE, extract = FALSE)

Arguments

pkg package name. If missing the environment of the runtime caller package is returned.

skip a logical that indicates if the calling namespace should be skipped.

verbose logical that toggles verbosity

n number of namespaces to return

strict a logical that indicates if the global environment should be considered as a valid namespace.

unique logical that indicates if the result should be reduced to contain only one occurrence of each namespace.

envir environment where to start looking for a package name. The default is to use the runtime calling package environment.

.Global a logical that indicates if calls from the global environment should throw an error (FALSE: default) or the string 'R_GlobalEnv'.

rm.prefix logical that indicates if an eventual prefix 'package:' should be removed from the returned string.

package optional name of an installed package

lib.loc path to a library of R packages where to search the package
Package References

Description

Create a citation string from package specific BibTex entries, suitable to be used in Rd files. The entries are looked in a file named REFERENCES.bib in the package’s root directory (i.e. inst/ in development mode).

Usage

packageReference(key, short = FALSE)

Arguments

key character vector of BibTex keys
short logical that indicates if the reference should be shorten as First Author et al. if it has more than one author.

Value

a character string containing the text formatted BibTex entries
packageReferenceFile  BibTex Utilities

Description
packageReferenceFile returns the path to a package REFERENCES.bib file.

Usage
packageReferenceFile(PACKAGE = NULL)

Arguments
PACKAGE package name

packageRegistry  Package Registry

Description
packageRegistry provides ways to create query package specific registries.
packageRegistries lists registries from loaded packages.
hasPackageRegistry tells if a given package has a meta-registry or a given registry.
Each package sub-registry has its own set of fields. Sub-registries defined by passing a character string in argument regobj of setPackageRegistry have the following fields: 'key' and 'object'
setPackageRegistryEntry adds an entry in a package registry.

Usage
packageRegistry(regname = NULL, quiet = FALSE,
entry = FALSE, update = !entry,
package = topenv(parent.frame()))

packageRegistries(regname = NULL, package = NULL,
primary = FALSE)

hasPackageRegistry(regname = NULL, package)

setPackageRegistry(regname, regobj, description = "",
entrydesc = NA, ..., package = topenv(parent.frame()),
overwrite = FALSE)

setPackageRegistryEntry(regname, key, ...,
overwrite = FALSE, verbose = FALSE,
where = topenv(parent.frame()), msg = NULL)
Arguments

**regname** Name of a sub-registry, used as its identifier.

**quiet** a logical that indicates that one should return the (meta-)registry if it exists, or `NULL` otherwise, without throwing any error.

**entry** logical that indicates if the corresponding meta registry entry should be directly returned, without any other processing.

**update** logical that indicates if the package registry should be updated, by adding/removing entries from other loaded/unloaded packages.

**package** package where to store or look for the registry.

**primary** logical that indicates if only primary registries should be listed.

**regobj** a `registry` object or a single character string that indicates the class of the objects that are stored in the sub-registry. See details for the list of the sub-registry's fields in this latter case.

**description** short description line about the registry. It is recommended to provide such description as it makes clearer the purpose of the registry. This description is shown when the registry object is printed/formated/listed.

**entrydesc** human readable description that is used in log messages when registering/removing entries.

**overwrite** a logical that indicate if an existing registry with the same should be overwritten if it exists.

**key** entry identifier.

**where** package name or namespace that owns the registry.

**verbose** a logical that indicates if verbosity should be toggle on.

**msg** addon message to print at the end of the output log line, when `verbose=True`.

Details

Package registries are organised in a meta-registry (a registry of registries) within a package’s namespace. Each registry can be used to store sets of built-in or user-defined objects in an organised way, e.g. algorithms or datasets.

A package meta-registry is a `registry` object, whose entries are `registry` objects themselves. A sub-registry entry is defined by the following fields:

**key** The sub-registry's accession key/identifier (a character string).

**regobj** The sub-registry itself (a `registry` object)

**description** Human readable description of the purpose of the registry (a character string)

**description** Short human readable description of the type of entries (a character string)

**package** owner package, which is forced to be the package in which the meta registry is defined.
**parent**  The name of the package that holds the parent registry, which we call the primary package. This field is non empty for cross-package registries, i.e. registries that derive from primary package’s own registry. Their entries are defined when (lazy-)loading the dependent package’s namespace.

Note that this function cannot be called from the global environment, but from a package namespace, e.g., when a package is lazy-loaded on installation or loaded via the function `load_all` from the `devtools` package.

**Value**

a `registry` object or `NULL` (see argument `quiet`).

---

**packageTestEnv**  
*Returns the package internal environment where unit tests are stored.*

**Description**

Returns the package internal environment where unit tests are stored.

**Usage**

`packageTestEnv(pkg)`

**Arguments**

`pkg`  
package name. If missing the caller’s package is assumed.

---

**parsePackageCitation**  
*Formatting Package Citations in Sweave/knitr Documents*

**Description**

Formatting Package Citations in Sweave/knitr Documents

**Usage**

`parsePackageCitation(x)`

**Arguments**

`x`  
output document, as a single string.
Description

These functions have been defunct or superseded by other functions.

Usage

\texttt{write.bib(...)}

Arguments

\texttt{...} \quad \texttt{extra arguments}

Description

This function implements a mechanism to postpone actions, which can be executed at a later stage. This is useful when developing packages, where actions that need to be run in the \texttt{.onLoad} function but can be defined close to their context.

Usage

\texttt{postponeAction(expr, key = digest(tempfile()), group = NULL, envir = topns(strict = FALSE), verbose = getOption("verbose"))}

\texttt{runPostponedAction(group = NULL, verbose = getOption("verbose"))}

Arguments

- \texttt{expr} \quad \text{expression that define the action to postpone. Currently only functions are supported.}
- \texttt{key} \quad \text{identifier for this specific action. It should be unique across the postponed actions from the same group.}
- \texttt{group} \quad \text{optional parent action group. This enables to define meaningful sets of actions that can be run all at once.}
- \texttt{envir} \quad \text{environment in which the action should be executed. Currently not used.}
- \texttt{verbose} \quad \text{logical that toggles verbose messages.}
Examples

```r
opt <- options(verbose=2)

# define actions
postponeAction(function(){print(1:10), "print")
postponeAction(function(){print(1:10), "more")
postponeAction()
# execute actions
runPostponedAction()
runPostponedAction()

# restore options
options(opt)
```

---

**quickinstall**  
*Quick Installation of a Source Package*

**Description**
Builds and installs a minimal version of a package from its source directory.

**Usage**

```r
quickinstall(path, destdir = NULL, vignettes = FALSE,
force = TRUE, ..., 
lib.loc = if (!is.null(destdir)) TRUE)
```

**Arguments**
- **path**: path to the package source directory
- **destdir**: installation directory. If NULL, the package is installed in the default installation library. If NA, the package is installed in a temporary directory, whose path is returned as a value.
- **vignettes**: logical that indicates if the vignettes should be rebuilt and installed.
- **force**: logical that indicates if the package should be installed even if a previous installation exists in the installation library.
- **...**: extra arguments passed to `R CMD`
- **lib.loc**: library specification. If TRUE then the installation directory destdir is added to the default library paths. This can be useful if dependencies are installed in this directory. If NULL, then the default library path is left unchanged.

**Value**
The path of the library where the package was installed.
Description

R.exec executes a single R command via system2.
R.CMD executes R CMD commands.
R.SHLIB executes R CMD SHLIB commands.

Usage

R.exec(..., lib.loc = NULL)
R.CMD(cmd, ...)
R.SHLIB(libname, ...)

Arguments

... extra arguments that are concatenated and appended to the command.
lib.loc logical that indicates if the current library locations should be used. If a character vector, then it is used as the library path specification.
cmd command to run, e.g. ‘check’ or ‘INSTALL’.
libname name of the output compiled library

Description

This function extract sections from Rd files and convert them into LaTeX code. This can be useful to include Rd text into vignettes, hence keeping them up to date.

Usage

RdSection2latex(topic, package, i = 1L, notitle = TRUE)

Arguments

topic Rd topic
package package in which to search the topic
i index of the section to format
notitle logical that indicates if the section’s title should be removed
Example section

This is a nice section, with a bullet list:

- tata
- toto

Examples

\begin{verbatim}
RdSection2latex('RdSection2latex', package = 'pkgmaker')
\end{verbatim}

---

regfetch | Finds an entry in a registry.
---

Description

This function provides extra control on how entries are queried from a registry object. `pkgreg_fetch` loads the requested package registry and uses `regfetch` to retrieve data from it. `pkgreg_remove` removes an entry from a package registry.

Usage

\begin{verbatim}
regfetch(regobj, ..., all = FALSE, error = TRUE,
          exact = FALSE, KEYS = NULL, verbose = FALSE,
          entry = FALSE, msg = NULL)

pkgreg_fetch(regname, ..., msg = NULL,
             where = topenv(parent.frame()))

pkgreg_remove(regname, ..., msg = NULL,
              where = topenv(parent.frame()), quiet = FALSE)
\end{verbatim}

Arguments

- `regobj` a registry object
- `...` key value(s) to look up. If multiple indexes are used, then the primary key should come first.
- `all` logical to indicate if hidden keys (starting with a '.') should be returned and output in message.
- `error` a logical that indicates if an error should be thrown if the key has no match or multiple matches
- `exact` a logical that indicates if matching should be exact or partial. Note that if exact matches exist then they are returned, independently of the value of `exact`. 
**require.quiet**

Silent Require

*Description*

Silently require a package.

*Usage*

```r
require.quiet(package, character.only = FALSE, ...)
```

*Arguments*

- `package` the name of a package, given as a name or literal character string, or a character string, depending on whether `character.only` is `FALSE` (default) or `TRUE`.
- `character.only` a logical indicating whether package or help can be assumed to be character strings.

**requirePackage**

Require a Package

*Description*

Require a package with a custom error message.

*Usage*

```r
requirePackage(pkg, ...)
```

*Arguments*

- `pkg` package name as a character string
- `...` extra arguments concatenated to for the header of the error message
requireRUnit  

Load RUnit Compatible Package

Description

Loads the package responsible for the implementation of the RUnit framework, choosing amongst 'RUnitX', 'svUnit' and 'RUnit'.

Usage

requireRUnit(...)

Arguments

... arguments passed to requirePackage.

Value

nothing

rversion

Complete R version

Description

Returns the complete R version, e.g. 2.15.0

Usage

rversion()

Examples

rversion()
### setBiocMirror

#### Setting Mirrors and Repositories

**Description**

`setBiocMirror` sets all Bioconductor repositories (software, data, annotation, etc.) so that they are directly available to `install.packages`. It differs from `chooseBiocmirror` in that it effectively enables the repositories.

`getBiocMirror` is a shortcut for `getOption('BioC_mirror')`, which returns the current Bioconductor mirror as used by biocLite.

`getBiocRepos` returns urls to all Bioconductor repositories on a given mirror.

`setCRANMirror` sets the preferred CRAN mirror.

CRAN simply contains the url of CRAN main mirror ([http://cran.r-project.org](http://cran.r-project.org)), and aims at simplifying its use, e.g., in calls to `install.packages`.

**Usage**

```r
setBiocMirror(url = "http://www.bioconductor.org", version = NULL, unique = TRUE)

goBiocMirror()

getBiocRepos(url = "http://www.bioconductor.org", version = NULL)

setCRANMirror(url = CRAN, unique = TRUE)
```

**Arguments**

- `url` or Bioconductor mirror url
- `version` version number
- `unique` logical that indicate if duplicated urls or names should be removed.

**Format**

```
chr "http://cran.r-project.org"
```

**Examples**

```r
## Not run:
install.packages('pkgmaker', repos=CRAN)

## End(Not run)
```
**setClassRegistry**  
*Automatic S4 Class for Registry Entries*

**Description**

Automatic S4 Class for Registry Entries

**Usage**

```
setClassRegistry(registry, Class, ...)  
```

**Arguments**

- `registry` a registry object
- `Class` name of the class to generate
- `...` extra arguments passed to `setClass`.

---

**setPackageExtraHandler**  
*Install/Run Extra Things After Standard Package Installation*

**Description**

These functions define a framework to register actions for which default sets of arguments can be defined when (lazy-)loading a package, and run later on, e.g., after the package is installed using dedicated commands.

`setPackageExtraHandler` defines main action handler functions, for which actions are defined as a set of arguments and registered using `setPackageExtra`.

`packageExtraHandler` retrieves a given handler from the registry. For example, calling `setPackageExtra('install', pkgs='non_CRAN_pkg', repos='http://non-standard-repo')` in a source file of package 'myPkg' registers the call `install.packages('non_CRAN_pkg', repos='http://non-standard-repo')` in a registry internal to the package. All calls to `setPackageExtra('install', ...)` can then be run by the user, as a post installation step via `install.extrapackages('myPkg', ...)`. `packageExtra` retrieve a given extra action, either as its registry entry, or as a function that would perform the given action.

`packageExtraRunner` defines a function to run all or some of the actions registered for a given handler in a given package. For example, the function `install.extrapackages` is the runner defined for the extra handler 'install' via `packageExtraRunner('install')`. `install.extrapackages` runs all extra actions registered for a given package.

`install.extrapackages` is defined as the extra handler for the extra action handler 'install.packages'. All arguments in ... are passed to `install.packages`. By default, packages that are already installed are not re-installed. An extra argument `force` allows to force their installation. The packages are loaded if their installation is successful.
Usage

setPackageExtraHandler(handler, fun, ...)

packageExtraHandler(handler = NULL, ...)

setPackageExtra(handler, extra, ...)

packageExtra(handler = NULL, extra = NULL,
  package = NULL, .wrap = FALSE)

packageExtraRunner(handler)

install.extras(package, extra = NULL, handler = NULL,
  ..., .verbose = getOption("verbose"))

install.extrapackages(package, extra = NULL,
  handler = NULL, ..., .verbose = getOption("verbose"))

Arguments

  handler  name of a handler, e.g., 'install'. It must be unique across all handlers registered by any other packages.

  fun      handler function that will be called with the arguments registered with packageExtra(name, ...)

  package  package name where to store/look for the internal registries. End users should not need to use this argument.

  ...  extra arguments passed to internal function calls. In packageExtraHandler, these are passed to pkgreg_fetch.

    In setPackageExtra, these define default arguments for the handler function. These are overwritten by arguments in the call to runner function if any.

  extra  name of the extra action.

  .wrap   logical that indicates if a function that runs the extra action should be returned or only the default arguments

  .verbose logical that indicates if verbose messages about the extra actions being run should be displayed.

Value

  the runner function associated with the newly registered handler, as built by packageExtraRunner.
Description

The following functions to access/set the options from the set are assigned in envir:

<subset>Options
<subset>GetOption

Usage

setupPackageOptions(..., NAME = NULL,
  ENVIR = topenv(parent.frame()),
  RESET = isLoadingNamespace())

Arguments

...  a single named list or named arguments that provide the default options and their values.
NAME  name of the set of options. This is used as a prefix for the name of the associated global option: package:<name>.
ENVIR  environment where the option wrapper functions will be defined. No function is defined if ENVIR=NULL
RESET  a logical that indicates whether the option set should overwrite one that already exists if necessary. The default is FALSE (i.e. no reset), except when loading a namespace, either from an installed package or a development package – with devtools. If FALSE, an error is thrown if trying to setup options with the same name.

---

simpleRegistry  Simple Package Registry

Description

Simple Package Registry

Usage

simpleRegistry(name, envir = topenv(parent.frame()),
  verbose = FALSE)

Arguments

name  name of the registry object, with which it will be assigned in envir.
envir  environment where to store the registry object. Defaults to the caller's top environment.
verbose  logical that toggle a verbose message when the object is first created.
source_files

Source Multiple Files

Description

Vectorised version of source.

Usage

source_files(x, pattern = NULL, ...)

Arguments

x character vector containing filenames
...
extra arguments passed to source.
pattern an optional regular expression. Only file names which match the regular expression will be returned.

str_diff

Finding Differences Between Strings

Description

Computes which characters differ between two strings.

Usage

str_diff(x, y)

Arguments

x a single string
y a single string

Value

an integer vector containing the index of all mis-matched characters in the first string.
Examples

```r
# strings to compare
x <- "once upon a time"
y <- "once upon a time there was"
z <- "once upon two times"

# diff: x - y
d <- str_diff(x, y)
d
str(d)

# other comparisons
str_diff(y, x)
str_diff(x, x)
str_diff(x, z)
str_diff(y, z)
```

---

**str_out** | **Formatting Utilities**
---

**Description**

str_out formats character vectors for use in show methods or error/warning messages.

str_desc builds formatted string from a list of complex values.

str_fun extracts and formats a function signature. It typically formats the output capture.output(args(object)).

str_bs substitutes backspace characters (\b) to produce a character string as it would be displayed in the console.

**Usage**

```r
str_out(x, max = 3L, quote = is.character(x),
        use.names = FALSE, sep = ", ", total = FALSE)

str_desc(object, exdent = 0L)

str_fun(object)

str_bs(x)
```

**Arguments**

- `x` character vector
- `max` maximum number of values to appear in the list. If `x` has more elements than `max`, a "..." suffix is appended.
str_out

quote a logical indicating whether the values should be quoted with single quotes (defaults) or not.

use.names a logical indicating whether names should be added to the list as NAME=VAL, ... or not (default).

sep separator character

total logical that indicates if the total number of elements should be appended to the formatted string as "'a', 'b', ... 'z' (<N> total)".

object an R object

exdent extra indentation passed to str_wrap, and used if the output should spread over more than one lines.

Value

a single character string

Author(s)

Renaud Gaujoux

str_bs was adapted from a proposal from Yihui Xie.

Examples

#--------
# str_out
#--------
x <- letters[1:10]
str_out(x)
str_out(x, 8)
str_out(x, Inf)
str_out(x, quote=FALSE)
str_out(x, total = TRUE)

#--------
# str_fun
#--------
str_fun(install.packages)

#--------
# str.bs
#--------
# Backspace substitution
str_bs("abc")
str_bs("abc\b")
str_bs("abc\b\b")
str_bs("abc\bd")
str_bs("abc\b\b\de\b")

# more complex example
sys.getenv_value

Description

sys.getenv_value defines a function that acts as a global static variable.

Usage

sys.getenv_value(name, raw = FALSE)

Arguments

default default value for the environment variable.

echo $HOME

Examples

Sys.getenv_value System Environment Variables

Description

System Environment Variables

Usage

Sys.getenv_value(name, raw = FALSE)
testRversion

Arguments

name
variable name as a character string.

raw
logical that indicates if one should return the raw value or the conversion of any false value to FALSE.

Value
the value of the environment variable as a character string or NA if the variable is not defined at all.

Examples

# undefined returns FALSE
Sys.getenv_value('TOTO')

# raw undefined returns NA
Sys.getenv_value('TOTO', raw = TRUE)

Sys.setenv(TOTO='bla')
Sys.getenv_value('TOTO')

# anything false-like returns FALSE
Sys.setenv(TOTO='false'); Sys.getenv_value('TOTO')
Sys.setenv(TOTO='0'); Sys.getenv_value('TOTO')

# cleanup
Sys.unsetenv('TOTO')

descRtest

Description
Compares current R version with a given target version, which may be useful for implementing version dependent code.

Usage
testRversion(x, test = 1L)

Arguments

x
target version to compare with.

test
numeric value that indicates the comparison to be carried out. The comparison is based on the result from utils::compareVersion(R.version, x):
• 1: is R.version > x?
• 0: is R.version = x?
• -1: is R.version < x?
Value

a logical

Examples

    testRversion("2.14")
    testRversion("2.15")
    testRversion("10")
    testRversion("10", test = -1)
    testRversion("< 10")
    testRversion(Rversion())
    testRversion(paste0(=', Rversion()))

Description

The function `unit.test` provides a way to write unit tests embedded within package source files. These tests are stored and organised in the package namespace, and can be run using the unified interface provided by the function `link{utest}`. Both Runit and testthat tests are supported – and automatically detected.

Usage

    unit.test(x, expr, framework = NULL,
             envir = parent.frame())

Arguments

x          single character string used as test identifier/label
expr       expression containing the actual test commands. It is not evaluated, but only stored in the package namespace.
framework  Unit test framework
envir      the definition environment of object x.

Value

a test function with no arguments that wrapping around expr
**userIs**

*Checking R User*

Description

Tests if the current R user is amongst a given set of users.

Usage

```
userIs(user)
```

Arguments

- `user` the usernames to check for, as a character vector.

---

**utest**

*Running Unit Tests*

Description

Run unit tests in a variety of settings. This is still very experimental.

Usage

```
utest(x, ...)

## S4 method for signature 'function'
utest(x, run = TRUE)

## S4 method for signature 'character'
utest(x,
    filter = "^runit.+\.[rR]$", fun = "^test\.$", ..., testdir = "tests", framework = c("RUnit", "testthat"),
    quiet = Sys.getenv("RCMDCHECK") != "FALSE",
    lib.loc = NULL)

## S4 method for signature 'RUnitTestSuite'
utest(x, ..., quiet = FALSE,
    outdir = NULL)
```
Arguments

- `x`: object to which a unit test is attached
- `...`: extra arguments to allow extensions and are passed to the unit framework running functions.
- `run`: a logical that indicates if the unit test should be run
- `filter`: pattern to match files that contain the definition of the unit tests functions to run.
- `fun`: pattern to match the test functions to run.
- `testdir`: directory where to look for the test files
- `framework`: unit test framework
- `quiet`: a logical that indicates if the tests should be run silently
- `lib.loc`: path to a library where installed packages are searched for. Used is of the form `x='package:*'`.
- `outdir`: output directory

Methods

- `utest signature(x = "function")`: Run the unit test associated to a function.
- `utest signature(x = "character")`: Run a package test suite
- `utest signature(x = "JUnitTestSuite")`: Runs a JUnit test suite

Description

Inferring Unit Test Framework

Usage

```r
utestFramework(x, eval = FALSE)
```

Arguments

- `x`: an filename, a function or the body of a function
- `eval`: a logical that indicates if the value of `x` should be used.

Value

the name of the framework as a character string or NULL if it could not be detected.
utestPath  

*Unit Tests Result Directory*

**Description**

Returns the path to the directory where the results of unit tests are stored. This path is used by `utest` to save unit test results, which are read by `makeUnitVignette` to update the unit test vignette when running R CMD check.

**Usage**

```
utestPath(...)```

**Arguments**

- `...` extra arguments passed to `packagePath`, e.g., `package`.

---

`write.pkgbib`  

*Generate a Bibtx File from Package Citations*

**Description**

Generates a Bibtx file from a list of packages or all the installed packages. It is useful for adding relevant citations in Sweave documents.

**Usage**

```
write.pkgbib(entry = NULL, file = "Rpackages.bib",
             prefix = "", append = FALSE, verbose = TRUE)
```

**Arguments**

- `entry` a `bibentry` object or a character vector of package names. If NULL, then the list of all installed packages is used.
- `file` output Bibtx file. It can be specified as a filename (as a single character string), NULL for stdout, or a `link` object. If `file` is a character string, an extension `.bib` is appended if not already present.
- `prefix` character string to prepend to the generated packages’ Bibtx key.
- `append` a logical that indicates that the Bibtx entries should be added to the file. If FALSE (default), the file is overwritten.
- `verbose` a logical to toggle verbosity. If `file=NULL`, verbosity is forced off.
Details

Multiple citations are handled by adding a numeric suffix to the Bibtex key (other than the first/main citation) as "<pkname>\%i" (e.g. pkg, pkg2, pkg3).

This function has now been integrated by Romain François in the bibtex package.

Value

the list of Bibtex objects – invisibly.

Author(s)

Renaud Gaujoux, based on the function rpackagesNbib from Achim Zeileis (see References).

References


See Also

link{connection}, link{bibentry}

Examples

```r
write.pkbib(c('bibtex', 'utils', 'tools'), file='references')
bibs <- bibtex::read.bib('references.bib')
write.pkbib(bibs, 'references2.bib')
md5 <- tools::md5sum(c('references.bib', 'references2.bib'))
```

# write to stdout()
write.pkbib(c('bibtex', 'utils', 'tools'), file=NULL)

# clean up
unlink(c('references.bib', 'references2.bib'))
```

writeUnitVignette    Writes Unit Tests Vignette

Description

Writes a vignette that contains the results from running unit test suites.
write_PACKAGES_index

Usage

```r
writeUnitVignette(pkg, file, results = NULL, check = FALSE)
```

Arguments

- **pkg**: Package name
- **file**: Output Sweave (.Rnw) file
- **results**: result file or output character vector
- **check**: logical that indicates the call was made from R CMD check, in which case the vignette is updated only if results of unit tests can be found in the unit test output directory, where they would have been generated by `utest`.

---

write_PACKAGES_index  Generate CRAN-like Repository Index

Description

Generate CRAN-like Repository Index

Usage

```r
write_PACKAGES_index(path = ".", output = "index.html", pattern = NULL, title = "Packages", robots.file = TRUE)
```

Arguments

- **path**: path to the repository's root directory
- **output**: output filename – relative to the repository root path.
- **pattern**: regular expression used to filter the names of the packages that will appear in the index.
- **title**: title of the index page
- **robots.file**: logical that indicates if a file `robots.txt` that hides the repository from search engine robots should be created.
Index

*Topic datasets
  setBiocMirror, 47
*Topic methods
  utest, 57
.libPaths, 5
.options (mkoptions), 28
[[.package_options (option_symlink), 31
 $, 12
add_lib, 5
addnames, 3
addNamespaceExport
  (getLoadingNamespace), 14
addToLogger, 4
allFormals (extractLocalFun), 13
alphacol, 6
ArgumentParser, 8
as.package, 36
as.package (packageEnv), 35
as.package_options (option_symlink), 31
as.rnw/isManualVignette, 19
attr_mode (ExposeAttribute), 12
attr_mode<- (ExposeAttribute), 12
available.packages, 35
bibentry, 59
cgetAnywhere, 6
checkWarning, 7
chooseBiocMirror, 47
citecmd, 7
CLIArgumentParser, 8
colors, 6
compactVignettes/isManualVignette, 19
compile_src, 9
CRAN/setBiocMirror, 47
data, 34
environment, 34
exitCheck, 9
expand_dots (expand_list), 10
expand_list, 10
ExposeAttribute, 12
extractLocalFun, 13
file.path, 37
file_extension, 13
find.package, 36, 37
formals, 13
getAnywhere, 6
getBiocMirror (setBiocMirror), 47
getBiocRepos (setBiocMirror), 47
getLoadingNamespace, 14
graphics-utils, 15
hasArg, 15
hasArg2, 15
hasEnvvar, 16
hasNames/is_something), 21
hasPackageRegistry (packageRegistry), 38
hook_backspace (knit_ex), 22
hook_toggle (knit_ex), 22
hook_tries (knit_ex), 22
install.packages, 16
install.extras
  (setPackageExtraHandler), 48
install.packages, 17
install.packages, 17, 47, 48
inSwave, 17
is.dir (is_something), 21
is.file (is_something), 21
is_NA (is_something), 21
is_option_symlink (option_symlink), 31
is_something, 21
isCRAN/isCRANcheck, 18
isCRAN/isCRANcheck, 18
isCRANcheck, 18
INDEX

isDevNamespace (getLoadingNamespace), 14
isFALSE (is_something), 21
isInteger (is_something), 21
isLoadingNamespace
  (getLoadingNamespace), 14
isManualVignette, 19
isNamespace, 14
isNamespaceLoaded
  (getLoadingNamespace), 14
isNumber (is_something), 21
isPackageInstalled (packageEnv), 35
isReal (is_something), 21
isString (is_something), 21
isTRUE, 22
knit, 22
knit2html, 22
knit_ex, 22
knitr, 19
latex_bibliography (latex_preamble), 25
latex_preamble, 25
ldata (packageData), 34
libname (list.libs), 26
list.files, 26
list.libs, 26
listPackageOptions (option_symlink), 31
load_all, 35, 40
loadingNamespaceInfo, 14
makeFakeVignette, 27
makeUnitVignette, 27, 59
mfrow (graphics-utils), 15
mkoptions, 28
name, 45
new, 29
new2, 29
ns_get (getLoadingNamespace), 14
on.exit, 9
oneoffVariable, 29
onLoad, 30
onUnload (onLoad), 30
option_symlink, 31
option_symlink_target (option_symlink), 31
options, 28, 31
orderVersion, 32
packageCLI, 33
packageData, 34
packageDependencies, 35
packageEnv, 35
packageExtra (setPackageExtraHandler), 48
packageExtraHandler
  (setPackageExtraHandler), 48
packageExtraRunner
  (setPackageExtraHandler), 48
packageName, 35
packageName (packageEnv), 35
packageOptions (option_symlink), 31
packagePath, 59
packagePath (packageEnv), 35
packageReference, 37
packageReferenceFile, 38
packageRegistries (packageRegistry), 38
packageRegistry, 38
packageTestEnv, 40
palette, 6
par, 15
parseCMD (CLIArgumentParser), 8
parsePackageCitation, 40
pkgmaker-defunct, 41
pkgreg_fetch, 49
pkgreg_fetch (regfetch), 44
pkgreg_remove (regfetch), 44
postponeAction, 41
quickinstall, 42
R.CMD, 42
R.CMD (R.exec), 43
R.exec, 43
R.SHLIB (R.exec), 43
RdSection2latex, 43
regfetch, 44
registry, 39, 40, 44
regular expression, 51
require, 45
requirequiet, 45
requirePackage, 45, 46
requireRUNit, 46
rgb, 6
rnw (isManualVignette), 19
rnwChildren (isManualVignette), 19
rnwCompiler (isManualVignette), 19
rnwDriver (isManualVignette), 19
rnwIncludes (isManualVignette), 19
rnwWrapper (isManualVignette), 19
runPostponedAction (postponeAction), 41
Rversion, 46
setBiocMirror, 47
setClass, 48
setClassRegistry, 48
setCRANMirror (setBiocMirror), 47
setPackageExtra
  (setPackageExtraHandler), 48
setPackageExtraHandler, 48
setPackageRegistry (packageRegistry), 38
setPackageRegistryEntry
  (packageRegistry), 38
setupPackageOptions, 28, 49
simpleRegistry, 50
sortVersion (orderVersion), 32
source, 51
source_files, 51
str_bs (str_out), 52
str_desc (str_out), 52
str_diff, 51
str_fun (str_out), 52
str_ns (packageEnv), 35
str_out, 52
sVariable, 54
Sweave, 19
Sys.getenv_value, 54
system2, 43
testRversion, 55
topenv, 35
topns (packageEnv), 35
topns_name (packageEnv), 35
try, 22
unit.test, 56
userIs, 57
utest, 18, 27, 57, 59, 61
utest.character-method (utest), 57
utest.function-method (utest), 57
utest.RUnitTestSuite-method (utest), 57
utest-methods (utest), 57
utestFramework, 58
utestPath, 59
vignetteMakefile (isManualVignette), 19
write.bib (pkgmaker-defunct), 41
write_pkgbib, 59
write_PACKAGES_index, 61
writeUnitVignette, 60