Package ‘pkgmaker’

January 22, 2020

Type Package
Title Development Utilities for R Packages
Version 0.31
Date 2020-01-21
Description 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
LazyLoad yes
Encoding UTF-8
Depends R (>= 3.0.0), methods, stats, registry
Imports tools, grDevices, utils, graphics, codetools, digest, stringi, magrittr, stringr, xtable, withr, bibtex (>= 0.4), assertthat
Suggests devtools (>= 0.8), roxygen2, RUnit, testthat, knitr, markdown, yaml, Biobase, datasets
Collate 'bibtex.R' 'colors.R' 'data.R' 'namespace.R' 'utils.R'
'devutils.R' 'files.R' 'graphics.R' 'options.R' 'logging.R'
'unitTests.R' 'is.R' 'knitr.R' 'registry.R' 'package-extra.R'
'package.R' 'packages.R' 'project.R' 'rd.R' 'user.R'
'ventige.R' 'with.r'
RoxygenNote 7.0.2
NeedsCompilation no
Author Renaud Gaujoux [aut, cre]
Maintainer Renaud Gaujoux <renozao@protonmail.com>
Repository CRAN
Date/Publication 2020-01-22 12:50:02 UTC

R topics documented:

.silenceF .................................................. 4
addnames .................................................. 5
addToLogger ............................................... 6
add_lib ..................................................... 6
alphacol ..................................................... 7
askUser ....................................................... 8
bibtex ........................................................ 9
cgetAnywhere ............................................. 10
charmap ....................................................... 10
checkWarning ............................................. 11
citecmd ..................................................... 11
compile_src ............................................... 12
CRAN .......................................................... 12
digest_function .......................................... 13
exitCheck ................................................... 13
expand_list ............................................... 14
ExposeAttribute .......................................... 16
extractLocalFun .......................................... 17
factor2character .......................................... 17
file_extension ............................................ 18
find_devpackage ......................................... 18
getLoadingNamespace .................................... 19
gfile ........................................................ 20
graphics-utils ............................................ 20
hasArg2 ...................................................... 21
hasEnvvar ................................................... 22
inSweave .................................................... 22
irequire ...................................................... 23
isCRANcheck ............................................... 24
is_package_path .......................................... 25
is_something ............................................... 25
iterCount .................................................... 27
knit_ex ....................................................... 27
latex_preamble ............................................ 30
list.data .................................................... 32
list.libs .................................................... 33
load_all_file ............................................. 34
load_project .............................................. 34
lverbose ..................................................... 35
makeFakeVignette ......................................... 36
makeUnitVignette ......................................... 37
R topics documented:

make_vignette_auxfiles .......................................................... 37
messagef ............................................................................. 38
mkoptions .......................................................... 39
new2 .......................................................... 40
oneoffVariable .......................................................... 41
onLoad .......................................................... 41
option_symlink .......................................................... 42
orderVersion .......................................................... 43
packageData .......................................................... 44
packageEnv .......................................................... 45
packageReference .......................................................... 47
packageRegistry .......................................................... 48
packageTestEnv .......................................................... 50
parsePackageCitation .......................................................... 50
pkgmaker-deprecated .......................................................... 50
postponeAction .......................................................... 51
quickinstall .......................................................... 52
R.exec .......................................................... 52
RdSection2latex .......................................................... 53
read.yaml_section .......................................................... 54
regfetch .......................................................... 54
render_notes .......................................................... 55
reorder_columns .......................................................... 56
require.quiet .......................................................... 56
requireRUnit .......................................................... 58
rnw .......................................................... 58
R.version .......................................................... 60
setBiocMirror .......................................................... 60
setClassRegistry .......................................................... 61
setPackageExtraHandler .......................................................... 62
setupPackageOptions .......................................................... 63
simpleRegistry .......................................................... 64
source_files .......................................................... 64
str_diff .......................................................... 65
str_out .......................................................... 66
sVariable .......................................................... 68
Sys.getenv_value .......................................................... 69
sys_call_stack .......................................................... 70
testRversion .......................................................... 70
unit.test .......................................................... 71
unlist_ .......................................................... 72
unlist_with_sep .......................................................... 73
userData .......................................................... 74
userIs .......................................................... 74
using_something .......................................................... 75
utest .......................................................... 76
utestFramework .......................................................... 77
utestPath .......................................................... 78
Description

Generates a wrapper function that silences the output, messages, and/or warnings of a given function.

Usage

\mcode{.silenceF(f, level = 7L)}

Arguments

\begin{itemize}
\item \textbf{f} \hfill function to silence
\item \textbf{level} \hfill a single numeric (integer) that indicates the silencing level, which encodes the set of output to be silenced.
\end{itemize}

It is interpreted like unix permission bit system, where each bit of the binary expression of the silencing level corresponds to a given type of output:

\begin{itemize}
\item 0: nothing silenced;
\item 1: stdout;
\item 2: stderr messages;
\item 4: stderr warnings.
\end{itemize}

For example, level 3 = 2 + 1 means silencing stdout and stderr, while 5 = 3 + 2 means silencing stderr messages and warnings, but not outputs to stdout. The default value is 7 = 4 + 2 + 1, which silences all output.

Negative values are supported and mean "silence everything except the corresponding type", e.g., \texttt{level = -1} silences all except stdout (computed as the binary complementary of 7, i.e. \texttt{7 -1 = 5 = 3 + 2}). See examples.

Value

a function
Examples

```r
f <- function()
  cat("stdout message\n")
  message("stderr message")
  warning("stderr warning", immediate. = TRUE)
}

# example of generated wrapper
f <- .silenceF(f)

# use of silencing level
for(l in 7:-7){ message("\nLevel: ", l); .silenceF(f, l)() }

# inline functions
ifun <- .silenceF(function(){ f(); invisible(1) })
ifun()
ifun <- .silenceF(function(){ f(); 1 })
ifun()
ifun <- .silenceF(function(){ f(); 1 }, 2L)
ifun()
```

addnames

Generating Names

Description

Generates names or dimnames for objects.

Usage

```r
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.
- **...**: extra arguments to allow extension and passed to the next method.
- **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.

---

**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`.

---

**add_lib**

*Adding Package Libraries*

**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

```r
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

```r
alphacol(col, alpha = FALSE)
```
Arguments

- **col**: vector of any of the three kinds of R color specifications, i.e., either a color name (as listed by `colors()`), a hexadecimal string of the form "#rrggbbaa" or "#rrggbbaa" (see `rgb`), or a positive integer i meaning `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)
```

Description

This function is an improved version of `userQuery` from Bioconductor `Biobase` package, which asks the user about some task that needs her intervention to proceed, e.g., ask if one should perform a computation, install a package, etc..

Usage

```r
askUser(msg, allowed = c("y", "n"), idefault = "n", default = "n",
case.sensitive = FALSE)
```

Arguments

- **msg**: The output message
- **allowed**: Allowed input from the user
- **idefault**: default response in interactive mode. This answer will be in upper case in the question and will be the one returned if the user simply hits return.
- **default**: default response in non-interactive mode. If NA, then the user is forced to provide an answer, even in non-interactive mode (e.g., when run through Rscript).
- **case.sensitive**: Is the response case sensitive? Defaults to FALSE

Value

the character string typed/agreed by the user or directly the default answer in non-interactive mode.
bibtex  

**Bibtex Utilities**

**Description**

Utility functions to work with BiBTeX files.

**Usage**

- `packageReferenceFile(PACKAGE = NULL, check = FALSE)`
- `package_bibliography(PACKAGE = NULL, action = c("path", "copy", "load"))`

**Arguments**

- `PACKAGE`  
  package name. If `NULL`, then the name of the calling package is used.

- `check`  
  logical that indicates if the result should be an empty string if the bibliography file (or package) does not exist.

- `action`  
  single character string that specifies the action to be performed:
  - 'path': return the path to the bibliography file. It returns an empty character string if the file does not exist.
  - 'copy': copy the bibliography file to the current directory, overwriting any existing REFERENCES.bib file.
  - 'load': load the bibliography file and return a list of `utils::bibentry` objects. It returns `NULL` if the file does not exist.

**Functions**

- `packageReferenceFile`: returns the path to a package REFERENCES.bib file.
- `package_bibliography`: returns the bibliography associated with a package. This can

**Examples**

```r
packageReferenceFile('pkgmaker')
packageReferenceFile('pkgmaker', check = TRUE)
```
### cgetAnywhere

**Get Anywhere**

**Description**

Similar to `getAnywhere`, but looks for the value of its argument.

**Usage**

```r
cgetAnywhere(x)
```

**Arguments**

- `x` a single character string

### charmap

**Substituting Strings Against a Mapping Table**

**Description**

Match the elements of a character vectors against a mapping table, that can specify multiple exact or partial matches.

**Usage**

```r
charmap(x, maps, nomatch = NULL, partial = FALSE, rev = FALSE)
```

**Arguments**

- `x` character vector to match
- `maps` mapping tables. May be a character vector, a list of character vectors or a function.
- `nomatch` character string to be used for non-matched elements of `x`. If NULL, these elements are left unchanged.
- `partial` logical that indicates if partial matches are allowed, in which case mappings are used as regular expressions.
- `rev` logical that indicates if the mapping should be interpreted in the reverse way.
**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**

```r
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**

```r
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

---

**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

---

**CRAN**  
*Main CRAN Mirror URL*

Description

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

Usage

CRAN

Format

An object of class character of length 1.
Examples

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

---

**digest_function**  
*Compute Function Digest Hash*

**Description**

Computes a digest hash of the body and signature of a function. Note that the hash is not affected by attributes or the function’s environment.

**Usage**

```r
digest_function(fun, n = Inf)
```

**Arguments**

- `fun`: a function
- `n`: a single numeric that indicates the length of the hash.

**Details**

The hash itself is computed using `digest::digest`.

**Value**

A character string

---

**exitCheck**  
*Exit Error Checks*

**Description**

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

**Usage**

```r
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 exits 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) )
```

---

**expand_list**

**Expanding Lists**

**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.

**Usage**

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

Arguments

- **x**: input list
- **...**: extra named arguments defining the default items. A list of default values can also be passed as a a single unnamed argument.
- **.exact**: logical that indicates if the names in **x** should be partially matched against the defaults.
- **.names**: 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.
- **.exclude**: optional character vector of argument names to exclude from expansion.

Value

- a list

Functions

- 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.

Examples

```r
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)

# 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
f(a=10, b=4)

---

**ExposeAttribute**

*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 `$` and/or `$<-$`.

### 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., 'r' or 'w'), 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**

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

---

**factor2character**  

*Converting Factors to Character Vectors*

**Description**

Converts all factor variables to character vectors in a `data.frame` or phenotypic data.

**Usage**

```r
factor2character(x)
```

**Arguments**

- `x` `data.frame` or `ExpressionSet` object

**Value**

- an object of the same class as `x`. 

---
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')
```

---

find_devpackage  

**Find Path to Development Package Root Directory**

**Description**

Development packages are looked-up according to rules defined in a file `.Rpackages` in the user’s home directory.

**Usage**

```r
find_devpackage(x, error = TRUE)
```

**Arguments**

- `x`  
  name of the development package to lookup.
- `error`  
  logical that indicates if an error is thrown when the project root directory could not be found.
**Specification of package path**

Package paths are specified in a list with:

- unnamed elements: character strings give path to directories to lookup for sub-directories that match exactly the package’s name;
- named element containing character strings: these are paths that are looked up only for packages that match the element name. If the element name contains any of the characters `*?()$\[]`, then it is matched using regular expression.

---

**getLoadingNamespace**

*Namespace Development Functions*

**Description**

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

**Usage**

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

isLoadingNamespace(ns, nodev = FALSE)

isNamespaceLoaded2(ns)

isDevNamespace(ns)

addNamespaceExport(x)

ns_get(x, ns = NULL, ...)

**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
- `nodev` logical that indicates if loading devtools namespace should be discarded.
- `ns` the name of a namespace or a namespace whose loading state is tested. If missing `isLoadingNamespace` test if any namespace is being loaded.
- `x` character vector containing the names of R objects to export in the loading namespace.
- `...` extra arguments passed to `get0`.

**Value**

getLoadingNamespace: 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.
Functions

- `isLoadingNamespace`: Tests if a namespace is being loaded.
- `isNamespaceLoaded2`: Tests if a given namespace is loaded, without loading it, contrary to `isNamespace`. It is similar to `isNamespaceLoaded` – which it uses – but also accepts environments.
- `isDevNamespace`: Tests the – current – namespace is a devtools namespace.
- `addNamespaceExport`: Dynamically adds exported objects into the loading namespace.
- `ns_get`: Gets an object from a given namespace.

---

### gfile

**Open a File Graphic Device**

**Description**

Opens a graphic device depending on the file extension.

**Usage**

```r
gfile(filename, width, height, ...)
```

**Arguments**

- `filename`: Path to the image file to create.
- `width`: Output width.
- `height`: Output height.
- `...`: Other arguments passed to the relevant device function such as `png` or `pdf`. ImportFrom grDevices bmp jpeg pdf png svg tiff.

---

### graphics-utils

**Utility Functions for Graphics**

**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**

```r
mfrow(n)
```
hasArg2

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

ame the name of an argument as a character string.

Examples

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

**Check Environment Variables**

**Description**

Tells if some environment variable(s) are defined.

**Usage**

`hasEnvar(x)`

**Arguments**

- `x`: environment variable name, as a character vector.

**Examples**

```r
hasEnvar("_R_CHECK_TIMINGS_")
hasEnvar("ABCD")
```

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
```
irequire

Description

Like base `require`, `irequire` tries to find and load a package, but in an interactive way, i.e. offering the user to install it if not found.

Usage

`irequire(package, lib = NULL, ..., load = TRUE, msg = NULL,
quiet = TRUE, prependLF = FALSE, ptype = c("CRAN-like", "BioC",
"BioCsoft", "BioCann"), autoinstall = !interactive())`

Arguments

- `package` name of the package
- `lib` path to the directory (library) where the package is to be looked for and installed if agreed by the user.
- `...` extra arguments passed to `install.packages`
- `load` a logical that indicates if the package should be loaded, possibly after installation.
- `msg` message to display in case the package is not found when first trying to load/find it. This message is appended to the string “Package `<packagename>` is required”.
- `quiet` logical that indicates if loading a package should be done quietly with `require.quiet` or normally with `require`.
- `prependLF` logical that indicates if the message should start at a new line.
- `autoinstall` logical that indicates if missing packages should just be installed without asking with the user, which is the default in non-interactive sessions.

Value

TRUE if the package was successfully loaded/found (installed), FALSE otherwise.

See Also

Other require: `require.quiet`
isCRANcheck

Package Check Utils

Description

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

Usage

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').

**Warning:** the checks performed on CRAN check machines are on purpose not always run with such flags, so that users cannot effectively "trick" the checks. As a result, there is no guarantee this function effectively identifies such checks. If really needed for honest reasons, CRAN recommends users rely on custom dedicated environment variables to enable specific tests or examples.

Functions

- **isCRAN_timing**: tells if one is running CRAN check with flag '--timing'.
- **isCHECK**: tries harder to test if running under R CMD check. It will definitely identifies check runs for:  
  - unit tests that use the unified unit test framework defined by **pkgmaker** (see **utest**);  
  - examples that are run with option _R_CHECK_RUNNING_EXAMPLES_ = TRUE, which is automatically set for man pages generated with a fork of **roxygen2** (see **References**).

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_=<pkgname>), 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.

References

Adapted from the function CRAN in the fda package.

https://github.com/renozao/roxygen

Examples

```
isCHECK()
```

<table>
<thead>
<tr>
<th>is_package_path</th>
<th>Test for Package Root Directory</th>
</tr>
</thead>
</table>

Description

Tells if a directory is a package directory, i.e. that it contains a DESCRIPTION file.

Usage

```
is_package_path(x, error = FALSE)
```

Arguments

- `x`: path to the directory to test
- `error`: logical that indicates if an error should be raised if the directory is not a package directory.

<table>
<thead>
<tr>
<th>is_something</th>
<th>Testing Object Type</th>
</tr>
</thead>
</table>

```
Description

Testing Object Type

is_NA tests if a variable is exactly NA (logical, character, numeric or integer)

isFALSE Tests if a variable is exactly FALSE.

isNumber tests if a variable is a single number

isReal tests if a variable is a single real number

isInteger tests if an object is a single integer

isString 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.

hasNames tests if an object has names.

Usage

is_NA(x)

isFALSE(x)

isNumber(x)

isReal(x)

isInteger(x)

isString(x, y, ignore.case = FALSE)

is.dir(x)

is.file(x)

hasNames(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
**iterCount**

**Simple Text Iteration Counter**

**Description**

Simple Text Iteration Counter

**Usage**

```r
iterCount(n = 100, i0 = 0L, title = "Iterations", extra = NULL, verbose = TRUE)
```

**Arguments**

- `n` number of total steps
- `i0` starting step
- `title` character string to use as title
- `extra` character vector providing extra text to add at each step
- `verbose` logical that toggles the counter

**Examples**

```r
progress <- iterCount(LETTERS)
res <- sapply(LETTERS, function(x){
  Sys.sleep(.1)
  progress()
})
# terminate counter
i_end <- progress(NULL)
i_end
```

---

**knit_ex**

**Knitr Extensions**

**Description**

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

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.
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 browser, 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.

Functions

- hook_try: is a knitr hook to enable showing error messages thrown by try. The function is
  not meant to be called directly, but only registered using knitr::knit_hooks (see details on this
  dedicated man page).
  This simply defines a function try in envir that prints the error message if any, and is called
  instead of base try.
- hook_toggle: is a chunk hook that adds clickable elements to toggle individual code chunks
  in HTML documents generated from .Rmd files.

Examples

library(knitr)
knit_ex("1 + 1")

library(knitr)

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

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

# no message caught
knit_ex()

##(r, include = FALSE)
knit_hooks$set(try = pkgmaker::hook_try)
##

##(r, try=TRUE)
try( stop('ah ah') )
##

# Correctly formatting backspaces in chunk outputs
tmp <- tempfile(fileext = '.Rmd')
cat(file = tmp, "
##(r, include = FALSE)
library(knitr)
knit_hooks$set(backspace = pkgmaker::hook_backspace())
##
Default knitr does not handle backspace and adds a special character:
##(r)
cat('abc\bd')
##
Using the hook backspace solves the issue:
##(r, backspace=TRUE)
cat('abc\bd')
##
")

# knit
out <- knitr::knit2html(tmp, fragment.only = TRUE)
# look at output
## Not run:
browseURL(out)
edit( file = out)

## End(Not run)
# cleanup
out_files <- list.files(dirname(out), full.names = TRUE,
   pattern = paste0("^", tools::file_path_sans_ext(out)))
unlink(c(tmp, out_files))

knit_ex(""

Declare chunk hook:
##(r, setup)
library(knitr)
knit_hooks$set(toggle = hook_toggle())

The R code of this chunk can be toggled on/off, and starts visible:
\r\n\r
\r^{\text{(r, toggle=TRUE)}}
print(1:10)
\r\n
The R code of this chunk can be toggled on/off, and starts hidden:
\r\n\r^{\text{(r, toggle=FALSE)}}
print(1:10)
\r\n
This is a plain chunk that cannot be toggled on/off:
\r\n\r^{\text{(r)}}
print(1:10)
\r\n
Now all chunks can be toggled and start visible:
\r\n\r^{\text{(r, toggle_all)}}
opts_chunk$set(toggle = TRUE)
\r\n\r^{\text{(r)}}
\r
sample(5)

To disable the toggle link, one can pass anything except TRUE/FALSE:
\r\n\r^{\text{(r, toggle = NA)}}
\r
sample(5)

", open = TRUE)
Arguments

- **PACKAGE**: package name
- **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. Bioc package citations)
- **GEO**: logical that indicate if general GEOmnibus commands should be added (e.g. urls to 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.

Details

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

Functions

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

Examples

```r
latex_preamble()
l溴ex_preamble(R=TRUE, only=TRUE)
l_ex_preamble(R=FALSE, CRAN=FALSE, GEO=FALSE)
l溴ex_preamble(GEO=TRUE, only=TRUE)
```
list.data  

List Package Data Objects

Description

Lists data objects that are shipped within package(s).

Usage

list.data(package = NULL)

Arguments

package  

a single character string that specifies the name of a particular package where to look for data objects.

Value

a data.frame object with columns:

- package: name of the package that holds the data object.
- data: name of the key to use in utils::data or ldata to load the data object.
- object: name of the (sub-)object that is contained in the data object.

See Also

utils::data, ldata

Examples

# list all data objects
head(list.data())

# list all data objects in package 'datasets'
subset(list.data("datasets"), data %in% "beavers")
**list.libs**  
*Library Files Utilities*

**Description**

Lists binary library files in a directory

**Usage**

```r
list.libs(dir, ..., all.platforms = FALSE)
libname(x)
```

**Arguments**

- `dir`: directory
- `...`: extra arguments passed to `list.files`.
- `all.platforms`: a logical that indicates whether to list library files for the current platform only (default) or all platforms (Unix, Windows, Mac).
- `x`: a filename

**Value**

`list.libs`: a character vector

**Functions**

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

**Examples**

```r
libname('mylib.so')
libname('/some/path/somewhere/mylib.dll')
```
**load_all_file**  
*Generate a Loading Script for Development Packages*

**Description**

Writes a script file that contains code that loads a given development package.

**Usage**

```r
load_all_file(path = path.package(package), package, dest = NULL)
```

**Arguments**

- `path`: a character string that contains the path to the development package.
- `package`: the name of the package for which the loading script must be generated. It must be a package that has already been loaded with `devtools::load_all` in the current session, so that its path can be retrieved.
- `dest`: the path to script file to create (as a character string). If not provided, then the script is written in a temporary .R file with prefix "load_all_<pkgname>_".

**Details**

This is useful when we want to load a development package in batchtools registries:

```r
library(devtools)
library(batchtools)

load_all("path/to/pkgA")
makeRegistry(..., source = load_all_file("pkgA"))
```

**Value**

a character string that contains the path to the script file.

---

**load_project**  
*Load Development Package*

**Description**

Load Development Package

**Usage**

```r
load_project(pkg, reset = FALSE, ..., utests = TRUE, verbose = FALSE,
              addlib = TRUE, character.only = FALSE, try.library = FALSE)

library_project(...)```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pkg</strong></td>
<td>name of the package/project to load.</td>
</tr>
<tr>
<td><strong>reset</strong></td>
<td>logical that indicates if the package should be reloaded (passed to <code>load_all</code>).</td>
</tr>
<tr>
<td><strong>...</strong></td>
<td>other arguments passed to <code>load_all</code>.</td>
</tr>
<tr>
<td><strong>utests</strong></td>
<td>logical that indicates if an environment containing the unit test functions should be created. If TRUE this environment is accessible at <code>pkgname::UnitTest::test.filename.r$function_name</code>.</td>
</tr>
<tr>
<td><strong>verbose</strong></td>
<td>logical that indicates if log messages should be printed.</td>
</tr>
<tr>
<td><strong>addlib</strong></td>
<td>logical that indicates if the <code>lib/</code> sub-directory, if it exists, should be prepended to the library path. This enables to control the version of the loaded dependencies.</td>
</tr>
<tr>
<td><strong>character.only</strong></td>
<td>logical that indicates if argument <code>pkg</code> should be evaluated or taken literal.</td>
</tr>
<tr>
<td><strong>try.library</strong></td>
<td>logical that indicates if projects that could not be found should be looked up in the installed packages.</td>
</tr>
</tbody>
</table>

**Functions**

- `library_project`: shortcut for `load_project(..., try.library = TRUE)`, to load project code from installed library if not found as a development project. All its arguments are passed to `load_project`.

---

**1verbose**

Log**ging Feature**

Description

`1verbose` returns/sets the current verbosity level.

Usage

```r
1verbose(val, global = FALSE)
lsilent()
is.verbose()
1message(level, ..., appendLF = TRUE, sep = "", force = FALSE)
vmessage(...)
log_append(...)```
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

```r
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

make_vignette_auxfiles

Generate RMarkdown Vignette Auxiliary Files

Description

Generate RMarkdown Vignette Auxiliary Files

Usage

```r
make_vignette_auxfiles(PACKAGE, input = NULL, bibfile = "library.bib", Rpkg.prefix = "Rpackage_", ...)
```

Arguments

- `PACKAGE`: package name
- `input`: vignette source file. If NULL then the current file is obtained via a call to `knitr::current_input`
- `bibfile`: output file for R package citations.
- `Rpkg.prefix`: prefix to use when generating the bibtex entries of cited R packages. If `Rpkg.prefix` = 'Rpackage_', then Rmarkdown citations should be @Rpackage_mypkg.
- `...`: other arguments passed to `latex_preamble`
messagef

Details
To use this feature add the following in your YAML header:

header-includes:
- \input{``r pkgmaker::make_vignette_auxfiles('pkgmaker')``}
bibliography: library.bib

messagef  General Log Formatting

Description
Generate a formatted diagnostic message. This function is a shortcut for message(sprintf(...)).

Usage
messagef(fmt, ..., domain = NULL, appendLF = TRUE)
wnote(..., immediate. = TRUE)

Arguments
fmt  a character vector of format strings, each of up to 8192 bytes.
... values to be passed into fmt. Only logical, integer, real and character vectors are supported, but some coercion will be done: see the 'Details' section. Up to 100.
domain see gettext.
appendLF logical: should messages given as a character string have a newline appended?
immediate. logical, indicating if the call should be output immediately, even if getOption("warn") <= 0.

Functions
- wnote: throws a simple note as an immediate warning. It is a shortcut for warning(..., immediate. = TRUE, call. = FALSE).

See Also
  sprintf, message

Examples

messagef("Hello %s number %i", 'world', 4)
**Description**

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

**Usage**

`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`.

**Functions**

- `.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`)

**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())
```
new2  

*Alternative S4 Constructor*

**Description**

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

**Usage**

`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")))
```
oneoffVariable

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
option_symlink

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.
chname a character string naming a DLL (also known as a dynamic shared object or library) to load.
libpath a character string giving the complete path to the package.

Examples

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

  pkgmaker::onLoad(libname, pkgname)
}

# 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 stores 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

option_symlink(x)

is_option_symlink(x, opts)

option_symlink_target(x, opts)

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

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

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

listPackageOptions()

Arguments

x a character string, a list or an object of class package_options.

opts a list of options

... arguments passed to getOption (only first one is used).

defaults NULL or a list of default options with their values.

PACKAGE a package name

Value

listPackageOptions: a character vector (possibly empty).

Examples

listPackageOptions()

orderVersion

Ordering Version Numbers

Description

Orders a vector of version numbers, in natural order.

Usage

orderVersion(x, ..., decreasing = FALSE)

sortVersion(x, ...)

---

Usage

orderVersion(x, ...)
Arguments

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

Examples

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

sort(v)
sortVersion(v)
```

packageData

**Loading Package Data**

**Description**

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

**Usage**

```r
packageData(list, envir = .GlobalEnv, ..., options = NULL,
            stringsAsFactors = getOption("stringsAsFactors", TRUE))

ldata(list, ..., package = NULL, error = TRUE, simplify = TRUE)
```

**Arguments**

- **list**: character vector containing the names of the data to load.
- **envir**: the `environment` where the data should be loaded.
- **...**: other arguments eventually passed to `data`.
- **options**: list of R options to set before calling `data`. This may be useful if the data is shipped as an R script.
- **stringsAsFactors**: logical that indicates if character columns of tabular data should be converted into factors.
packageEnv

package is a character vector giving the package(s) to look in for data sets, or NULL. By default, all packages in the search path are used, then the 'data' subdirectory (if present) of the current working directory.

error is a logical that indicates whether an error should be thrown if the requested data cannot be found.
simplify is a logical that indicates if queries of one object only (i.e. argument list is of length one) should return the data object itself.

Value

the loaded data.

Functions

- ldata: loads a package data in the parent frame. It is a shortcut for packageData(list, ..., envir=parent.frame()).

Examples

```r
## Not run: mydata <- packageData('mydata')

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

## End(Not run)
```

---

packageEnv  Package Development Utilities

Description

packageEnv is a slight modification from toplevel, 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.

Usage

```r
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, check = TRUE)

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.

... arguments passed to file.path.

package optional name of an installed package

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

check logical that indicates if an error should be thrown if the path to the package root directory cannot be found. If this is the case and check = FALSE, then the function returns NULL.

x package specified by its installation/development path or its name as 'package:*'.

quiet a logical that indicate if an error should be thrown if a package is not found. It is also passed to find.package.

extract logical that indicates if DESCRIPTION of package source files should be extracted. In this case there will be no valid path.

Value

packageEnv: packageEnv returns an environment

packageName: a character string

packagePath: a character string
Functions

- `topns_name`: returns the name of the runtime sequence of top namespace(s), i.e. the name of
  the top calling package(s), from top to bottom.
  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`: an 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`.

---

packageReference | 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, PACKAGE = NULL)`

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.
- `PACKAGE`: package in which the BiBTeX entry is defined.

Value

a character string containing the text formatted BibTeX entries
packageRegistry

Description

packageRegistry provides ways to create query package specific registries.

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/formatted/listed.
entrydesc human readable description that is used in log messages when registering/removing
entries.
... named values used to set extra information about the new registry, that are stored
in the corresponding fields of the meta-registry. Currently not used, as no extra
field other than 'description' is defined.
overwrite: a logical that indicate if an existing registry with the same should be overwritten if it exists.

key: entry identifier.

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

where: package name or namespace that owns the registry.

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

packageRegistry: a registry object or NULL (see argument quiet).

Functions

- `packageRegistries`: lists registries from loaded packages.
- `hasPackageRegistry`: tells if a given package has a meta-registry or a given registry.
- `setPackageRegistry`: creates a package-specific registry within a package.
  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.
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.

---

pkgmaker-deprecated

*Deprecated Functions in pkgmaker*

**Description**

These functions have been deprecated and will be defunct in the next release.

**Usage**

`requirePackage(pkg, ...)`

**Arguments**

- `pkg` package name to load.
- `...` 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 \link{.onLoad} function but can be defined close to their context.

**Usage**

```r
postponeAction(expr, key = digest(tempfile()), group = NULL,
    envir = topns(strict = FALSE), verbose = getOption("verbose"))

runPostponedAction(group = NULL, verbose = getOption("verbose"))
```

**Arguments**

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

**Examples**

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

# define actions
postponeAction(function(){print(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 install 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.

---

**R.exec**  

**Executing R Commands**

**Description**

Functions to execute R commands.

**Usage**

```r
R.exec(..., lib.loc = NULL)
```

```r
R.CMD(cmd, ...)
```

```r
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

Functions

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

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)

Example section

This is a nice section, with a bullet list:

• tata
• toto

Examples

RdSection2latex('RdSection2latex', package = 'pkgmaker')
read.yaml_section  
**Reads YAML Options Embedded into a File**

**Description**
Reads YAML Options Embedded into a File

**Usage**
```r
read.yaml_section(section, file = "~/.Rprofile", text = NULL)
```

**Arguments**
- **section**: section name to lookup in the file. In the file this defined by paired tags "#<section_name>" , "#/section_name" or a single tag "#<section_name@file_path>" that redirect to a YAML file.
- **file**: path to the file to parse. Default is to parse the user's .Rprofile.
- **text**: text to parse. If provided, then argument file is not used.

regfetch  
**Finds an entry in a registry.**

**Description**
This function provides extra control on how entries are queried from a registry object.

**Usage**
```r
regfetch(regobj, ..., all = FALSE, error = TRUE, exact = FALSE, KEYS = NULL, verbose = FALSE, entry = FALSE, msg = NULL)
```

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

```r
pkgreg_remove(regname, ..., msg = NULL, where = topenv(parent.frame()), quiet = FALSE)
```

**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.

KEYS alternative way of passing the key value(s). If not missing, then arguments in ... are discarded.

verbose a logical that indicates if verbosity should be toggle on

entry a logical that indicates if the

msg a header to use in case of error.

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

where package name or namespace that owns the registry.

quiet a logical that indicates if the operation should be performed quietly, without throwing errors or warnings.

Functions

- pkgreg_fetch: fetches entries in a package registry, as set up by setPackageRegistry. It loads the requested package registry and uses regfetch to retrieve data from it.
- pkgreg_remove: removes an entry from a package registry.

render_notes

Renders Rmarkdown Documents Using User Default Options

Description

Renders Rmarkdown Documents Using User Default Options

Usage

render_notes(input, output_format = NULL, output_options = NULL, ..., .config = NULL)

Arguments

input Input file (R script, Rmd, or plain markdown).

output_format R Markdown output format to convert to. Pass "all" to render all formats defined within the file. Pass the name of a format (e.g. "html_document") to render a single format or pass a vector of format names to render multiple formats. Alternatively you can pass an output format object; e.g. html_document(). If NULL is passed then the output format is the first one defined in the YAML metadata of the input file (defaulting to HTML if none is specified).

output_options List of output options that can override the options specified in metadata (e.g. could be used to force self_contained or mathjax = "local"). Note that this is only valid when the output format is read from metadata (i.e. not a custom format object passed to output_format).

... other arguments passed to render

.config location of the default options (a YAML file). Default behaviour is to look for file '.rmarkdown.yaml' in the user’s home directory, or, if missing, for a yaml section rmarkdown::render in the user’s R profile.
See Also

read.yaml_section

---

reorder_columns  Reordering Columns

Description

Reorders columns according to a preferred target order

Usage

reorder_columns(x, target, decreasing = FALSE)

Arguments

- **x**
  - an object with columns, such as a matrix or a data.frame, or from a class that support subsetting via x[,i,drop = FALSE] and has a method colnames.

- **target**
  - a character or named numeric vector that specifies the column preferred order. If a numeric vector, then its names are assumed to correspond to columns, and its values determine the target order – according to argument decreasing.

- **decreasing**
  - logical that indicates in which direction a numeric target vector should be ordered.

Details

Column names will be reordered so that their order match the one in target. Any column that does not appear in target will be put after those that are listed in target.

Value

- an object of the same type and dimension

---

require.quiet  Loading Packages

Description

require.quiet silently requires a package, and qrequire is an alias to require.quiet.
require.quiet

Usage

require.quiet(...) qrequire(...) qlibrary(...) mrequire(msg, package, lib.loc = NULL, quietly = FALSE)

Arguments

... extra arguments passed to library or require.
msg error message to use, to which is appended the string 'requires package <pkg>' to build the error message.
package name of the package to load.
lib.loc a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known to .libPaths(). Non-existent library trees are silently ignored.
quietly a logical. If TRUE, no message confirming package attaching is printed, and most often, no errors/warnings are printed if package attaching fails.

Functions

• qlibrary: silently loads a package.
• mrequire: tries loading a package with base require and stops with a – custom – error message if it fails to do so.

See Also

Other require: irequire

Examples

mrequire('Running this example', 'stringr')
try(mrequire('Doing impossible things', 'notapackage'))
requireRUnit

Description

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

Usage

requireRUnit(...)  

Arguments

... arguments not used.

Value

nothing

rnw

Utilities for Vignettes

Description

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).

Usage

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

isManualVignette()

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

rnwCompiler(x, verbose = TRUE)

rnwWrapper(x, verbose = TRUE)

rnwDriver(x)

rnwIncludes(x)

rnwChildren(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 file-name (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.

Functions

- 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 R CMD check.
• \texttt{as.rnw}: creates a S3 \texttt{rnw} object that contains information about a vignette, e.g., source filename, driver, fixed included files, etc..

• \texttt{rnwCompiler}: tries to detect the vignette compiler to use on a vignette source file, e.g., \texttt{Sweave} or \texttt{knitr}.

• \texttt{rnwWrapper}: tries to detect the type of vignette and if it is meant to be wrapped into another main file.

• \texttt{rnwDriver}: tries to detect Sweave driver to use on a vignette source file, e.g., \texttt{SweaveCache}, \texttt{highlight}, etc..

• \texttt{rnwIncludes}: detects fixed includes, e.g., image or pdf files, that are required to build the final document.

• \texttt{rnwChildren}: detects included vignette documents and return them as a list of vignette objects.

• \texttt{vignetteMakefile}: returns the path to a generic makefile used to make vignettes.

• \texttt{compactVignettes}: compacts vignette PDFs using either \texttt{gs\_quality='none'} or \texttt{'ebook'}, depending on which compacts best (as per CRAN check criteria).

---

### Description

Rversion

Returns the complete R version, e.g. 2.15.0

### Usage

\texttt{Rversion()}

### Examples

\texttt{Rversion()}

---

### Description

setBiocMirror

Setting Mirrors and Repositories

setBiocMirror sets all Bioconductor repositories (software, data, annotation, etc.) so that they are directly available to \texttt{install.packages}. It differs from \texttt{chooseBioCmirror} in that it effectively enables the repositories.
**setClassRegistry**

Automatic S4 Class for Registry Entries

**Description**

Automatic S4 Class for Registry Entries

**Usage**

```r
classRegistry(registry, Class, ...)```

**Arguments**

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

**Functions**

- `setBiocMirror(url = "http://www.bioconductor.org", version = NULL, unique = TRUE)`: is a shortcut for `getOption('BioC_mirror')`, which returns the current Bioconductor mirror as used by `biocLite`.
- `getBiocRepos(url = "http://www.bioconductor.org", version = NULL)`: returns urls to all Bioconductor repositories on a given mirror.
- `setCRANMirror(url = CRAN, unique = TRUE)`: sets the preferred CRAN mirror.
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`.

**Usage**

```r
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,...)`
- `...` 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.
- `package` package name where to store/look for the internal registries. End users should not need to use this argument.
- `.wrap` logical that indicates if a function that runs the extra action should be returned or only the default arguments
setupPackageOptions

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

ame 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, ...)
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

# 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.

Usage

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

str_desc(object, exdent = 0L)

str_fun(object)

str_class(x, max = Inf, ...)

str_pkg(pkg, lib.loc = NULL)

str_md5sum(x)

str_hash(x, algo = "md5")

str_dim(x, dims = dim(x))

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.
  quote      a logical indicating whether the values should be quoted with single quotes (de-
             faults) 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',..., 'z' (<N> total)".

**str_out**

An R object

**exdent**

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

... other arguments passed to `str_out`.

**pkg**

Package name

**lib.loc**

Path to a library of R packages

**algo**

The algorithms to be used; currently available choices are `md5`, which is also the default, `sha1`, `crc32`, `sha256`, `sha512`, `xxhash32`, `xxhash64`, `murmur32` and `spookyhash`.

**dims**

A numeric vector of dimensions. Default is to use the input object dimensions (via function `dims()`)

**Value**

A single character string

**Functions**

- `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_class`: outputs the class(es) of an object using `str_out`.
- `str_pkg`: formats a package name and version
- `str_md5sum`: computes md5sum on character vector using `md5sum`.
- `str_hash`: computes hash of a character vector using `digest`.
- `str_dim`: builds a string that describes the dimension of an object, in the form n x m for 2D-objects, n x m x p for 3D-objects, and so on.
- `str_bs`: substitutes backspace characters (\b) to produce a character string as it would be displayed in the console.

**Author(s)**

Renaud Gaujoux

`str_bs` was adapted from a proposal from Yihui Xie.

**Examples**

```r
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(install.packages)
str_class(matrix())
```
### sVariable

**Global Static Variable**

**Description**

`sVariable` defines a function that acts as a global static variable.

**Usage**

`sVariable(default = NULL)`

**Arguments**

- `default` default value for the static variable.

**Examples**

```r
# define variable
x <- sVariable(1)
# get value (default)
x()
# set new value: return old value
old <- x(3)
old
# get new value
x()
```
**Sys.getenv_value**

*System Environment Variables*

**Description**

System Environment Variables

**Usage**

`Sys.getenv_value(name, raw = FALSE)`

**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 is the variable is not defined at all.

**Examples**

```r
# 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')
```
### sys_call_stack

#### System Call Stack Utilities

**Description**

System Call Stack Utilities

**Usage**

```r
sys.function_digest(n = NULL)
sys.function_nframe(fun)
sys.function_frame(fun)
sys.source_file()
```

**Arguments**

- `n`: a single frame
- `fun`: the function object to find in the call stack.

**Functions**

- `sys.function_digest`: computes digest hash for each function in the call stack.
- `sys.function_nframe`: returns the index of the frame that calls a given function.
- `sys.function_frame`: returns the frame that calls a given function.
- `sys.source_file`: returns path to the script that is being sourced either by `base::source` or `base::sys.source`.

---

### testRversion

#### Testing R Version

**Description**

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

**Usage**

```r
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()))

unit.test  Embedded Unit Tests

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
evir the definition environment of object x.
Value

a test function with no arguments that wrapping around expr

unlist_

Flatten a List Conserving Names

Description

unlist2 is a replacement for base::unlist that does not mangle the names.

Usage

unlist_(x, recursive = TRUE, use.names = TRUE,
        what.names = "inherited")

Arguments

x See ?unlist.
recursive See ?unlist.
use.names See ?unlist.
what.names "inherited" or "full".

Details

Use this function if you don’t like the mangled names returned by the standard unlist function from the base package. Using unlist with annotation data is dangerous and it is highly recommended to use unlist_ instead.

Author(s)

Herve Pages

Source

Bioconductor AnnotationDbi::unlist2

Examples

x <- list(A=c(b=-4, 2, b=7), B=3:-1, c(a=1, a=-2), C=list(c(2:-1, d=55), e=99))
unlist(x)
unlist_(x)

# annotation maps (as in AnnotationDbi objects
egids2pbids <- list('10' = 'a', '100' = c('b', 'c'), '1000' = c('d', 'e'))
egids2pbids
unlist_with_sep  Flattens All List Levels Using Separated Names

Description

Flattens All List Levels Using Separated Names

Usage

unlist_with_sep(x, sep = "/", use.names = TRUE, depth = Inf)

Arguments

x a list object, usually containing other lists – of lists.
sep character string used to separate each component of the final element names.
use.names logical that indicates if the original names of each the successive nested list elements should be used to build the final names of the result list.
depth maximum number of levels to unlist. Root level is 1L.

Examples

x <- list(X = list(a = 1,
              , b = list(b.1 = 2
                            , b.2 = list(b.2.1 = 4, b.2.2 = data.frame())
                            , b.3 = 3)
              , c = matrix()))
unlist_with_sep(x)
unlist_with_sep(x, '###')
**Description**

The package-specific user data base directory is the sub-directory `R-data/`, located in the user’s home or within a directory defined by global option `userData.path`.

If in interactive mode, and the base directory does not exist yet, the user is asked if it should be created in his home directory. Otherwise, or if the user does not allow the creation in his home, this directory is created in the current R session’s temporary directory.

**Usage**

```r
userData(..., create = NULL, package = topenv(parent.frame()))
```

**Arguments**

- `...`  path parts passed to `file.path` to be appended to the main path.
- `create`  logical that indicates if the base directory should be created if it does not exists. Note that directories – and files – under the base directory are not automatically created. The user should therefore care of it in the caller function if necessary. If `create=TRUE`, then the base directory is forced to be created in the user’s home directory. If `create=FALSE`, then the base directory is never created. See also section `Details`.
- `package`  name of the package associated with the user data path. It is used to prefix the path, within the user R data directory.

**See Also**

- `tempdir`

---

**userIs**  

**Checking R User**

**Description**

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

**Usage**

```r
userIs(user)
```

**Arguments**

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

Execute code in temporarily altered environment.

Description
These functions were extracted from the devtools package to make them available without a dependency to devtools.

Usage
using_envvar(new, code, action = "replace")
using_env(new, code)
using_locale(new, code)
using_collate(new, code)
using_dir(new, code)
using_libpaths(new, code)
using_lib(new, code)
using_options(new, code)
using_par(new, code)
using_path(new, code, add = TRUE, prepend = FALSE)

Arguments
new values for setting
code code to execute in that environment
action (for using_envvar only): should new values "replace", "suffix", "prefix" existing environmental variables with the same name.
add Combine with existing values? Currently for using_path only. If FALSE all existing paths are overwritten, which you don’t usually want.
prepend logical that indicates if the new paths should be added in front of the current ones.

Details
- using_dir: working directory
- using_collate: collation order
- `using_envvar`: environmental variables
- `using_libpaths`: library paths, replacing current libpaths
- `using_lib`: library paths, prepending to current libpaths
- `using_locale`: any locale setting
- `using_options`: options
- `using_path`: PATH environment variable
- `using_par`: graphics parameters

**Deprecation**

`using_env` will be deprecated in devtools 1.2 and removed in devtools 1.3

**Author(s)**

Hadley Wickham

**Examples**

```r
getwd()
using_dir(tempdir(), getwd())
getwd()
Sys.getenv("HADLEY")
using_envvar("HADLEY" = 2), Sys.getenv("HADLEY")
Sys.getenv("HADLEY")

using_envvar("A" = 1),
using_envvar("A" = 2), action = "suffix", Sys.getenv("A")
```

---

**utest**  

*Running Unit Tests*

**Description**

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

**Usage**

```r
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
...
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 (by class)

* function: Run the unit test associated to a function.
* character: Run a package test suite
* RUnitTestSuite: Runs a RUnit test suite

utestFramework

Inferring Unit Test Framework

Description

Inferring Unit Test Framework

Usage

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`.

---

**winbuild**  
*Build a Windows Binary Package*

**Description**

Build a Windows Binary Package

**Usage**

`winbuild(path, outdir = ".", verbose = TRUE)`

**Arguments**

`path` path to a source or already installed package  
`outdir` output directory  
`verbose` logical or numeric that indicates the verbosity level

**Value**

Invisibly returns the full path to the generated zip file.
Examples

## Not run:

# from source directory
winbuild('path/to/package/source/dir/')
# from tar ball
winbuild('PKG_1.0.tar.gz')

## End(Not run)

write.bib  

### Defunct Functions in pkgmaker

**Description**

These functions have been defunct or superseded by other functions.

**Usage**

```r
write.bib(...)```

**Arguments**

```r
... extra arguments```

write.pkgbib  

### Generate a Bibtex File from Package Citations

**Description**

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

**Usage**

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

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 Bibtex file. It can be specified as a filename (as a single character string), NULL for stdout, or a `link{connection}` 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’ Bibtex key.
- **append**: a logical that indicates that the Bibtex 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 "<pkgname>%i" (e.g. pkg, pkg2, pkg3).

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

Value

the list of Bibtex objects – invisibly.

Author(s)

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

References


See Also

`link{connection}`, `link{bibentry}`

Examples

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

# write to stdout()
write.pkgbib(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.

Usage

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 cal 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 \texttt{utest}.  


Index

+Topic datasets
  CRAN, 12
  .libPaths, 6, 57
  .options (mkoptions), 39
  .silenceF, 4
  [[.package_options (option_symlink), 42
$,, 16

add_lib, 6
addnames, 5
addNamespaceExport
  (getLoadingNamespace), 19
addToLogger, 6
allFormals (extractLocalFun), 17
alphacol, 7
as.package, 47
as.package_options (option_symlink), 42
as.rnw (rnw), 58
as_package (packageEnv), 45
askUser, 8
attr_mode (ExposeAttribute), 16
attr_mode<-' (ExposeAttribute), 16

base::source, 70
base::sys.source, 70
base::unlist, 72
bibentry, 80
bibtex, 9

cgetAnywhere, 10
charmap, 10
checkWarning, 11
chooseBioCmirror, 60
citecmd, 11
colors, 8
compactVignettes (rnw), 58
compile_src, 12
CRAN, 12

data, 44
devtools::load_all, 34
digest, 67
digest::digest, 13
digest_function, 13
environment, 44
exitCheck, 13
expand_dots (expand_list), 14
expand_list, 14
ExposeAttribute, 16
extractLocalFun, 17
factor2character, 17
file.path, 46, 74
file_extension, 18
find.package, 46, 47
find_devpackage, 18
formals, 17

get0, 19
getAnywhere, 10
getBioCmirror (setBioCMirror), 60
getBioCRepos (setBioCMirror), 60
getLoadingNamespace, 19
ggetOption, 38
gettext, 38
gfile, 20
graphics-utils, 20

hasArg, 21
hasArg2, 21
hasEnvvar, 22
hasNames (is_something), 25
hasPackageRegistry (packageRegistry), 48
hook_backspace (knit_ex), 27
hook_toggle (knit_ex), 27
hook_try (knit_ex), 27

install.extrapackages
  (setPackageExtraHandler), 62
install.extras
(setPackageExtraHandler), 62
install.packages, 12, 23, 60, 63
inSweave, 22
require, 23, 57
is.dir (is_something), 25
is.file (is_something), 25
is.verbosel (lverbose), 35
is_NA (is_something), 25
is_option_symlink (option_symlink), 42
is_package_path, 25
is_something, 25
isCHECK (isCRANcheck), 24
isCRAN_timing (isCRANcheck), 24
isCRANcheck, 24
isDevNamespace (getLoadingNamespace), 19
isFALSE (is_something), 25
isInteger (is_something), 25
isLoadingNamespace
(setLoadingNamespace), 19
isManualVignette (rnw), 58
isNamespace, 20
isNamespaceLoaded, 20
isNamespaceLoaded2
(setLoadingNamespace), 19
isNumber (is_something), 25
isPackageInstalled (packageEnv), 45
isReal (is_something), 25
isString (is_something), 25
isTRUE, 26
iterCount, 27
knit, 28
knit2html, 28
knit_ex, 27
knitr, 58, 60
knitr::current_input, 37
knitr::knit_hooks, 28
latex_bibliography (latex_preamble), 30
latex_preamble, 30, 37
ldata, 32
ldata (packageData), 44
libname (list.libs), 33
library, 57
library_project (load_project), 34
list.data, 32
list.files, 33
list.libs, 33
listPackageOptions (option_symlink), 42
lmessage (lverbose), 35
load_all, 35, 45, 49
load_all_file, 34
load_project, 34
loadingNamespaceInfo, 19
log_append (lverbose), 35
lsilent (lverbose), 35
lverbose, 35
make_vignette_auxfiles, 37
makeFakeVignette, 36
makeUnitVignette, 37, 78
md5sum, 67
message, 38
messagef, 38
mfrow (graphics-utils), 20
mkoptions, 39
mrequire (require.quiet), 56
new, 40
new2, 40
ns_get (getLoadingNamespace), 19
on.exit, 13, 14
oneoffVariable, 41
onLoad, 41
onUnload (onLoad), 41
option_symlink, 42
option_symlink_target (option_symlink), 42
options, 39, 42
order, 44
orderVersion, 43
package_bibliography (bibtex), 9
packageData, 44
packageEnv, 45
packageExtra (setPackageExtraHandler), 62
packageExtraHandler
(setPackageExtraHandler), 62
packageExtraRunner
(setPackageExtraHandler), 62
packageName, 47
packageName (packageEnv), 45
packageOptions (option_symlink), 42
packagePath, 78
packagePath (packageEnv), 45
using_dir(using_something), 75
using_env(using_something), 75
using_envvar(using_something), 75
using_lib(using_something), 75
using_libpaths(using_something), 75
using_locale(using_something), 75
using_options(using_something), 75
using_par(using_something), 75
using_path, 75
using_path(using_something), 75
using_something, 75
utest, 24, 37, 76, 78, 81
utest, character-method (utest), 76
utest, function-method (utest), 76
utest, RUnitTestSuite-method (utest), 76
utestFramework, 77
utestPath, 78
utils::bibentry, 9
utils::data, 32
vignetteMakefile (rnw), 58
vmessage (lverbose), 35
winbuild, 78
wnote (messagef), 38
write.bib, 79
write.pkgbib, 79
writeUnitVignette, 81