Package 'roxyPackage'

June 9, 2017

Type Package

Title Utilities to Automate Package Builds

Author m.eik michalke [aut, cre], Robert Nuske [ctb]

Maintainer m.eik michalke <meik.michalke@hhu.de>

Depends R (>= 2.9.0),methods,roxygen2,XiMpLe (>= 0.03-20)

Suggests testthat

Imports tools

Description The intention of this package is to make packaging R code as easy as possible. 'roxyPackage' uses tools from the 'roxygen2' package to generate documentation. It also automatically generates and updates files like *-package.R, DESCRIPTION, CITATION, ChangeLog and NEWS.Rd. Building packages supports source format, as well as several binary formats (MS Windows, Mac OS X, Debian GNU/Linux) if the package contains pure R code only. The packages built are stored in a fully functional local R package repository which can be synced to a web server to share them with others. This includes the generation of browsable HTML pages similar to CRAN, with support for RSS feeds from the ChangeLog. Please read the vignette for a more detailed explanation by example.

License GPL (>= 3)

Encoding UTF-8

LazyLoad yes

URL https://reaktanz.de/?c=hacking&s=roxyPackage

BugReports https://github.com/unDocUMeantIt/roxyPackage/issues

Version 0.05-3

Date 2017-06-09

RoxygenNote 6.0.1

Collate '00_classes_01_ChangeLog_internal.R' '00_classes_02_sandbox_internal.R'

'01_methods_01_ChangeLog.R'
'01_methods_02_ChangeLog_internal.R'
'01_methods_03_sandbox_internal.R'
'ChangeLog_functions.R'
'archive.packages.R'
'citationText.R'
'cl2news.R'
'debianize.R'
'debianizeKeyring.R'
'dep4deb.R'
'entities.R'
'news2rss.R'
'roxy.description.R'
'roxy.html.R'
'roxy.package.R'
'roxyPackage-internal.R'
'roxyPackage-internal_debianize.R'
'roxyPackage-internal_generateTestPackage.R'
'roxyPackage-internal_readme.R'
'roxyPackage-internal_sandbox.R'
'roxyPackage-internal_templateFile.R'
'roxyPackage-package.R'
'sandbox.R'
'templateFile.R'

R topics documented:

roxyPackage-package *The roxyPackage Package*

Description

Utilities to Automate Package Builds.

Details

Package:	roxyPackage
Type:	Package
Version:	0.05-3
Date:	2017-06-09
Depends:	R (>= 2.9.0),methods,roxygen2,XiMpLe (>= 0.03-20)
Encoding:	UTF-8
License:	GPL (>= 3)
LazyLoad:	yes
URL:	https://reaktanz.de/?c=hacking&s=roxyPackage

The intention of this package is to make packaging R code as easy as possible. 'roxyPackage' uses tools from the 'roxygen2' package to generate documentation. It also automatically generates and updates files like *-package.R, DESCRIPTION, CITATION, ChangeLog and NEWS.Rd. Building packages supports source format, as well as several binary formats (MS Windows, Mac OS X, Debian GNU/Linux) if the package contains pure R code only. The packages built are stored in a fully functional local R package repository which can be synced to a web server to share them with others. This includes the generation of browsable HTML pages similar to CRAN, with support for RSS feeds from the ChangeLog. Please read the vignette for a more detailed explanation by example.

Author(s)

m.eik michalke, with contributions from Robert Nuske

archive.packages *Deal with old packages in your local repository*

Description

Use this function to move older versions of a package to a specified archive directory, or remove them completely.

Usage

```
archive.packages(repo.root, to.dir = "Archive", keep = 1,
  keep.revisions = 2, package = NULL, type = "source",
  archive.root = repo.root, overwrite = FALSE, reallyDoIt = FALSE,
  graceful = FALSE, deb.options = list(distribution = "unstable", component
  = "main", gpg.version = 2, gpg.key = NULL, keyring = NULL))
```

Arguments

repo.root	Path to the repository root, i.e., the directory which contains the src and bin directories. Usually this path should start with "file:///".
to.dir	Character string, name of the folder to move the old packages to.
keep	An integer value defining the maximum nuber of versions to keep. Setting this to 0 will completely remove all packages from the repository, which is probably only useful in combination with the option package.
keep.revisions	An integer value defining the maximum nuber of revisions to keep. This is only used when archiving Debian packages, i.e., if type includes "deb". Setting this to 0 or NULL will keep all revisions of package versions that are to be kept.
package	A character vector with package names to check. If set, archive.packages will only take actions on these packages. If NULL, all packages are affected.
type	A character vector defining the package formats to keep. Valid entries are "source", "win.binary", "mac.binary", and "deb". By default, only the source packages are archived, all other packages are deleted, except for Debian repos, which currently can only be archived or be left as is.
archive.root	Path to the archive root, i.e., the directory to which files should be moved. Usually the Archive is kept in repo.root.
overwrite	Logical, indicates whether existing files in the archive can be overwritten.
reallyDoIt	Logical, real actions are only taken if set to TRUE, otherwise the actions are only printed.
graceful	Logical, if TRUE the process will not freak out because of missing files. Use this for instance if you deleted files from the repo but did not update the package indices.
deb.options	A named list of options that must be properly set if you want to archive Debian packages. After packages were removed from the repo, all Packages, Sources and Release files must be re-written and signed, and all of the following information is required: distribution, component, gpg.key, and keyring (which might be NULL). If you omit gpg.version, version 2 is assumed by default. See debianize for details.

Note

This function responds to sandbox.

See Also

sandbox to run archive.packages() in a sandbox.

Examples

```
## Not run:
# dry run, only prints what would happen, so you can check
# if that's really what you want
archive.packages("file:///var/www/repo")

# after we've confirmed that the right packages will be moved
# and deleted, let's actually commit the changes
archive.packages("file:///var/www/repo", reallyDoIt=TRUE)

# if we don't want a standard archive, but for instance a parallel
# archive repository, we can have it. let's move all but the latest two
# versions from /var/www/repo to /var/www/archive. to suppress the
# creation of a special archive directory, we set to.dir=""
archive.packages("file:///var/www/repo", to.dir="", keep=2,
  type=c("source", "win.binary", "mac.binary"),
  archive.root="/var/www/archive", reallyDoIt=TRUE)

## End(Not run)
```

`cl2news` *Convert ChangeLog/NEWS into NEWS.Rd*

Description

This function attempts to translate ASCII ChangeLog (or NEWS) files into NEWS.Rd files.

Usage

```
cl2news(log, news = NULL, codify = TRUE, overwrite = TRUE)
```

Arguments

`log`	Character string, path to the ChangeLog or NEWS file to be converted.
`news`	Character string, path to the NEWS.Rd file to be written. If `NULL`, results are written to `stdout()`.
`codify`	Logical, whether to try to detect code snippets like function names and markup them accordingly.
`overwrite`	Logical, whether to overwrite an existing NEWS.Rd file.

Details

This should work for ChangeLog and NEWS files that

1. have entries named "Changes in version <version number>" (and optionally a YYYY-MM-DD date string afterwards)
2. have single changes properly itemized, by indentation and then either `"o"`, `"-"` or `"*"` followed by space

3. optionally have categories as subsections, like "Fixed" or "Added"

Any text string that isn't indented and doesn't start with "Changes in version" will likely be treated as a subsection. The ChangeLog related functions and methods of this package, e.g. `initChangeLog`, are a convenient way to maintain R ChangeLogs in a proper format.

This function is basically a wrapper for the internal function `tools::news2Rd`.

Value

No return value, writes a file.

See Also

`initChangeLog`, `readChangeLog`, `updateChangeLog`, `writeChangeLog`

Examples

```
## Not run:
cl2news(log="~/myFiles/myRPackage/ChangeLog", news="~/myFiles/myRPackage/inst/NEWS.Rd")

# use capture.output() to dump the results into a character vector
NEWS.object <- capture.output(cl2news(log="~/myFiles/myRPackage/ChangeLog"))

## End(Not run)
```

---

debianize *Basic Debian package creation from R source packages*

Description

This function attempts to 'debianize' your R source package. This means, it will add a `debian` directory to sources' root directory, and populate it with needed files for Debian package building, as outlined in the Debian R Policy by Eddelbuettel & Bates (2003) and the Debian Policy Manual[1], version 3.9.3.1.

Usage

```
debianize(pck.source.dir, repo.root, build.dir = tempdir(), revision = 1,
  repo.name = "roxypackage", origin = paste0("other-", repo.name),
  distribution = "unstable", component = "main", urgency = "low",
  changelog = c("new upstream release"), deb.description = NULL,
  depends.origin = "cran", depends.origin.alt = list(), actions = c("deb",
  "bin", "src"), overwrite = c("changelog", "control", "copyright", "rules",
  "compat"), bin.opts = "-rfakeroot -b -uc", arch = "all", compat = 9,
  epoch = NULL, gpg.key = NULL, keyring = NULL, gpg.version = 2,
  deb.keyring.options = NULL, compression = "xz", keep.build = FALSE,
  keep.existing.orig = FALSE, replace.dots = FALSE)
```

Arguments

pck.source.dir	Character string, path pointing to the root directory of your package sources, to a local R package source tarball, or a full URL to such a package tarball. Tarballs will be downloaded to a temporary directory, if needed, extracted, and then debianized.
repo.root	Character string, valid path to a directory where to build/update a local package repository.
build.dir	Character string, valid path to a directory where to build the package. If this directory is not empty, a temporary directory will be created inside automatically.
revision	Numeric or a character string, the Debian package revision information.
repo.name	Character string, the name for your debian package repository. This can be used to generate an OpenPGP debian package from the given gpg.key, unless you change the default behaviour with the parameter deb.keyring.options
origin	Character string, should be either "noncran" or "other-<yourname>", used for the package name. This indicates that your package is not an official CRAN or BioC package.
distribution	Character string, the Debain (based) distribution your package is intended for.
component	Character string, the Debain component of the distribution.
urgency	Character string, urgency information for this release (refer to [1] if you want to change this).
changelog	Character vector, log entries for the ./debian/changelog file if it is going to be changed.
deb.description	A named list or data.frame with further information, especially for the ./debian/control file. This is similar to the pck.description parameter of roxy.package, only with different variables. Note that if certain key values are missing, debianize will automatically use some defaults: **Build.Depends.Indep** "debhelper (>> 7.0.0), r-base-dev (>= <R.vers>), cdbs", plus Depends/Imports in DESCRIPTION in debianized format; if arch is not set to "all", the field Build.Depends is used instead **Depends** "r-base-core (>= <R vers>)", plus Depends/Imports in DESCRIPTION in debianized format **Suggests** Suggests in DESCRIPTION in debianized format **Maintainer** generated from Sys.info (user <login@nodename>), with a warning. **Section** "gnu-r" **Priority** "optional" **Homepage** URL in DESCRIPTION Refer to [1] for further available fields in the ./debian/control file. In case you would like to add to the fields definig relations to other packages like Build.Depends.Indep or Depends rather than replacing them, provide a named list with a character vector called "append". For example: Depends=list(append=c("libmysql++3")).

`depends.origin` A character string to set the default origin for packages which are a dependency of this one. In case all dependencies can be met by Debian packages from CRAN releases, you can leave this to the default setting. If you need more control, see `depends.origin.alt`.

`depends.origin.alt`
A named list of alternative origins for packages which are a dependency of this one. By default, `depends.origin` is used, but if you know that certain dependencies are of different origin (e.g., your own repository), you can set this here. Each list element must be named after the R package you want to set an origin for, and must be a character vector or single string, like `list(foo="other-janedoe")`. If more than one origin is given, they will be set as alternatives (using the pipe `"|"` as "or").

`actions` Character vector, naming the actions to perform:

"deb" Debianize the package sources.

"bin" Build the Debian package.

"src" Build a Debian source package.

`overwrite` Character vector, naming the files which should be updated:

"changelog" Update `./debian/changelog`, but only if no entry for this package version and revision is there yet

"compat" Re-write `./debian/compat`

"control" Re-write `./debian/control`

"copyright" Re-write `./debian/copyright`

"rules" Re-write `./debian/rules`

"gpg.key" Re-write the keyring package in the repository (by default present packages are left unchanged)

`bin.opts` Character string, options to pass through to `dpkg-buildpackage` for the `"bin"` action.

`arch` Character string, architecture the package is build for.

`compat` Integer value, specifying the `debhelper` compatibility level.

`epoch` Integer value, the Debian package epoch information.

`gpg.key` Character string, the GnuPG key ID for the key that should be used for signing the Release file (secure apt). This key must be available in your keyring (or in the one specified by `keyring`). Note that this function defaults to using the SHA256 algorithm for signing (not SHA1). Mandatory for `"bin"` and `"src"` actions.

`keyring` Character string, path to an additional keyring file to use.

`gpg.version` Integer number, specifiying the GnuPG major version number. By default `gpg2` is assumed.

`deb.keyring.options`
Named list, extra options to pass through to `debianizeKeyring`. By default, the value for `maintainer` will be taken from `deb.description`, and the values for `gpg.key`, `repo.name`, `repo.root`, `build.dir`, `distribution`, `component`, `urgency`, `keyring`, and `gpg.version` are taken from the settings given with the `debianize` function call.

`compression` Character string, compression format for Debian source packages. Currently `"xz"` and `"gzip"` are supported.

`keep.build` Logical. If `build.dir` is not `pck.source.dir`, work is done in generated folder with a random name. Usually it is removed afterwards, unless you set this option to `TRUE`.

`keep.existing.orig` Logical, if `TRUE` and there is already a `*.orig.tar.[gz|xz]` archive in the repository matching this version, it will not be replaced with a re-packaged one but remains as is. This is useful for binary-only rebuilds.

`replace.dots` Logical. The proposed Debian R Policy actually asks to replace all dots in package names by hyphens. However, this is implemented differently in `r-cran.mk` and will lead to unbuildable packages. So the default here is to ignore the policy draft and keep dots in package names, as is true for a lot of CRAN packages as well (code is law). In case you run into problems here (symptoms include a failing .deb build because the directory `build/<package name>` doesn't exist), try turning this switch. If `TRUE` dots will be replaced by hyphens in both source and binary package names. Note that building a package by calling this function should always work, because it will automatically create a symlink in the build directory if needed.

Details

The file `./debian/source/format` will also be created only once. The files `./debian/control`, `./debian/copyright` and `./debian/rules` will be generated from the information found in the `DESCRIPTION` file of the R package. Once created, these files won't be touched again if they are not defined in the `overwrite` parameter. This enables you to save files from being re-written, e.g. if you altered them manually.

The `./debian/changelog` is special here, as `overwrite` doesn't mean the whole file will be overwritten, but rather that the function checks if the changelog already contains an entry for this particular package version and revision, and only if this is not the case will add one at the beginning of the file, including the log entries defined by the `changelog` parameter (each string will become one log entry).

The function will try to detect the license you specified in the `DESCRIPTION` file, and if it is one of the following licenses, generate some useful info on how to get the full license on a Debian system:

- Apache License
- Artisitic License
- BSD License
- GNU General Public License (GPL)
- GNU Lesser General Public License (LGPL)

Building the actual package

If you're running the R session on a Debian based system, the function can build the debian package, but it would likely fail when it comes to signing the .changes/.dsc files, because gpg gets invoked without `"--no-tty"`. You'd have to sign those files later, e.g. with `debsign`, if you really need this. However, secure-apt can still be ensured, if you provide a valid GnuPG key ID from your keyring,

which will then be used to sign the generated `Release` file. If not present yet, a copy of the public key will automatically be saved to the repository, in a file named `<key ID>.asc`.

Package building is done in a temporal directory, and the source files a copied there first. Set `build.dir=pck.source.dir` if you want to build in-place instead.

Package dependencies

This function will make no attempts to guess what package dependencies must be fulfilled. That is, if the defaults don't fit (see below), then you must define these dependencies yourself via the `deb.description` parameter (setting appropriate values for fields like `Build.Depends`, `Build.Depends.Indep` and `Depends`). In case your R package depends on other R packages, you will have to ensure that these are also available as Debian packages (and define them as dependencies), so the package management can take care of resolving these dependencies transparently. Otherwise users might have a hard time figuring out how to get your package to work, if the building process doesn't fail in the first place.

That said, you should always try to debianize the package without manual dependencies set first. After that, look at the generated `control` file and see if there are problems at all. Usually the default method is supposed to be quite clever when it comes to detect dependencies from the actual package `DESCRIPTION` file (it will automatically translate those into proper Debain package names, where tuning is possible via the `depends.origin` and `depends.origin.alt` parameters).

Repository access

After you debianized your package and built some Debian packages, `debianize` will prepare a Debain package repository in the specified directory (can be the same as used with `roxy.package`). You can now access it locally on your machine, or upload the whole thing to a web server etc. Basically, it should work if you add these lines to your repository configuration:

```
deb http://<URL you uploaded to>/deb <distribution> <component>
```

```
deb-src http://<URL you uploaded to>/deb <distribution> <component>
```

Debianizing arbitrary packages

With a little luck, this function can almot automatically debianize any R package sources. You can even provide the `pck.source.dir` parameter with a URL to package sources (e.g., a source package from CRAN), and `debianize` will do its best to end up with an installable debian package in the specified repository root.

Note

Please note that the package will always be built against the R version installed by your package management! Also, this function responds to `sandbox`.

References

Eddelbuettel, D. & Bates, D. (2003). *Debian R Policy – Draft Proposal v 0.1.3*. Available from http://lists.debian.org/debian-devel/2003/12/msg02332.html

[1] Debian Policy Manual: http://www.debian.org/doc/debian-policy

See Also

sandbox to run debianize() in a sandbox.

Examples

```
## Not run:
debianize(
  pck.source.dir="~/my_R_stuff/SquareTheCircle",
  repo.root="/var/www/repo",
  origin="other-doelle",
  revision=4,
  changelog=c("re-compiled docs"),
  deb.description=list(
    Depends=c("r-base-dev (>> 2.12.0), r-cran-foreign"),
    Maintainer="A. Sistent <sistent@eternalwondermaths.example.org>"),
  actions=c("deb"))

# let's try to debianize some R package from CRAN
debianize(
  pck.source.dir="http://cran.r-project.org/src/contrib/roxygen2_4.0.1.tar.gz",
  repo.root=tempdir(),
  deb.description=list(
    Maintainer="A. Sistent <sistent@eternalwondermaths.example.org>"
  )
)

## End(Not run)
```

---

debianizeKeyring *Package your OpenPGP keyring in Debian package format*

Description

Similar to debianize, this function generates a Debian package, but it specialises on packaging OpenPGP/GnuPG keyrings. The resulting package can be used to provide keys in a Debian package repository, hence enabling secure apt. They are probably easier to handle for users.

Usage

```
debianizeKeyring(gpg.key, repo.name, repo.root, maintainer,
  build.dir = tempdir(), keyname = paste0(repo.name, "-keyring"),
  pck.source.dir = file.path(tempdir(), keyname), version = "0.01",
  revision = 1, distribution = "unstable", component = "main",
  urgency = "low", URL = NULL, changelog = c("new upstream release"),
  description = paste("OpenPGP keyring for Debian packages hosted at the",
  repo.name, "repository.", "This keyring is neccessary to use secure apt."),
  actions = c("bin", "src"), overwrite = c("changelog", "control",
  "copyright", "postinst", "prerm", "rules", "compat"),
```

```
bin.opts = "-rfakeroot -b -uc", compat = 9, epoch = NULL,
keyring = NULL, gpg.version = 2, sign.key = gpg.key,
compression = "xz", keep.build = FALSE)
```

Arguments

gpg.key	Character string, the OpenPGP key ID for the key that should be included in the package. This key must be available in your keyring (or in the one specified by keyring).
repo.name	Character string, name of the repository this keyring will be used for. Must not include spaces or special characters!
repo.root	Character string, valid path to a directory where to build/update a local package repository.
maintainer	Character string, name an mail address of the maintainer of the keyring package, in the format of firstName lastName <your@mail.address>.
build.dir	Character string, valid path to a directory where to build the package. If this directory is not empty, a temporary directory will be created inside automatically.
keyname	Character string, a name for keyring. Will be used for both the exported keyring file and debian package name. Using something like "myrepo-keyring" is a good choice.
pck.source.dir	Character string, path pointing to the root directory of the keyring package sources. If this directory does not exist yet, it will be created and filled with the necessary files.
version	Numeric or a character string, the main Debian package version indicator for the keyring package.
revision	Numeric or a character string, the Debian package revision information.
distribution	Character string, the Debain (based) distribution your package is intended for.
component	Character string, the Debain component of the distribution.
urgency	Character string, urgency information for this release (refer to [1] if you want to change this).
URL	Character string, should point to the repository this keyring package is built for.
changelog	Character vector, log entries for the ./debian/changelog file if it is going to be changed.
description	Character string, some description of the keyring package.
actions	Character vector, naming the actions to perform: **"bin"** Build the Debian package. **"src"** Build a Debian source package.
overwrite	Character vector, naming the files which should be updated: **"changelog"** Update ./debian/changelog, but only if no entry for this package version and revision is there yet **"compat"** Re-write ./debian/compat **"control"** Re-write ./debian/control

"copyright" Re-write `./debian/copyright`

"postinst" Re-write `./debian/postinst`

"prerm" Re-write `./debian/prerm`

"rules" Re-write `./debian/rules`

"gpg.key" Re-write the exported key in `./keyrings/`

`bin.opts` Character string, options to pass through to `dpkg-buildpackage` for the `"bin"` action.

`compat` Integer value, specifying the `debhelper` compatibility level.

`epoch` Integer value, the Debian package epoch information.

`keyring` Character string, path to an additional keyring file to use.

`gpg.version` Integer number, specifiying the GnuPG major version number. By default `gpg2` is assumed.

`sign.key` Character string, the OpenPGP key ID for the key that should be used for signing the Release file (secure apt). This key must be available in your keyring (or in the one specified by `keyring`). Skipped if `NULL`.

`compression` Character string, compression format for Debian source packages. Currently `"xz"` and `"gzip"` are supported.

`keep.build` Logical. If `build.dir` is not `pck.source.dir`, work is done in generated folder with a random name. Usually it is removed afterwards, unless you set this option to `TRUE`.

See Also

`debianize`.

Examples

```
## Not run:
debianizeKeyring(
  gpg.key="DDCDA632",
  repo.name="doelle",
  repo.root="/var/www/repo",
  maintainer="A. Sistent <sistent@eternalwondermaths.example.org>"
)

## End(Not run)
```

---

dep4deb *Download package dependencies*

Description

Tries to fetch all (missing) R packages to successfully build a Debian package. The packages are downloaded in source format for you to `debianize`, but `dep4deb` can try to check for available Debian packages instead.

Usage

```
dep4deb(pck.source.dir, pck.name = NULL, destdir = file.path(tempdir(),
  "roxyPackge", "downloads"), repos = getOption("repos"), all = FALSE,
  check.deb = TRUE, origin = "cran", origin.alt = list(),
  available = NULL)
```

Arguments

pck.source.dir	Character string, path pointing to the root directory of your package sources, to a local R package source tarball, or a full URL to such a package tarball. Tarballs will be downloaded to destdir, if needed, extracted, and then checked for dependencies. Will be ignored if pck.name is not NULL.
pck.name	Character string, the package name. This is an alternative to using pck.source.dir.
destdir	File path to the directory where all downloaded files should be saved to.
repos	Character vector, the base URL(s) of the repositories to use (see download.packages).
all	Logical, if FALSE only currently missing packages are downloaded, where "missing" means that there is no Debian package if these packages installed. If TRUE and check.deb=FALSE, all dependencies will be downloaded.
check.deb	Logical, TRUE it will be checked if a debian package can be found, and if that is the case, its name is added to the results and the download skipped. If all=FALSE, packages will only be listed in the results if they are not installed.
origin	A character string for the package origin, see debianize.
origin.alt	A named list for more complex origin configuration, see debianize.
available	An object as returned by available.packages listing packages available at the repositories, or NULL which makes an internal call to available.packages.

Details

The function works its way recursively through the dependencies of the dependencies, beginning with the original package given. To make it easier for you to debianize the downloaded packages in a proper order, all downloads will be stored in numbered subfolders of the main download folder, and you should work from the highest number backwards.

Value

Returns a list with two elements:

dl A matrix as returned by download.packages, listing all downloaded sources

deb A character vector naming already available Debian packages

Examples

```
## Not run:
dep4deb(pck.name="roxyPackage")

## End(Not run)
```

entities *Translate character string into HTML entities*

Description

Translate character string into HTML entities

Usage

```
entities(string, collapse = TRUE)
```

Arguments

string	Character string to be translated.
collapse	Logical, if TRUE one single string is returned

Value

Either a named character vector, one element for each character, or a single string.

Examples

```
entities("foo_bar")
```

getChangeLogEntry *Read/write ChangeLog objects*

Description

This methods can be used to manage ChangeLog objects.

Usage

```
getChangeLogEntry(log, ...)

## S4 method for signature 'ChangeLog'
getChangeLogEntry(log, version = NULL)
```

Arguments

log	An object of class ChangeLog.
...	Additional options, as of now only version is supported (see below).
version	Character string, version number to look up.

Details

getChangeLogEntry takes a ChangeLog object and a version number string and returns the according entry.

Value

An object of class ChangeLog.

See Also

readChangeLog, updateChangeLog

Examples

```
## Not run:
changelog <- readChangeLog("/home/user/myRsources/myRpackage/ChangeLog")
CL.entry <- getChangeLogEntry(changelog, version="0.02-22")

## End(Not run)
```

---

news2rss *Generate RSS feeds from R NEWS files*

Description

This function should take either HTML or Rd files and return a valid RSS 2.0 XML file.

Usage

```
news2rss(news, rss = NULL, html = NULL, encoding = "UTF-8",
  channel = c(title = "", link = "", description = "", language = "", atom =
  ""))
```

Arguments

news	Character string, path to the R NEWS file to be converted.
rss	Character string, path to the RSS.xml file to be written. If NULL, results are written to stdout().
html	Logical, whether news is in HTML or Rd format. If NULL, guess this from the file ending.
encoding	Character string, how the feed is encoded.
channel	A named character vector with information on this RSS feed: title: Title of the feed, probably the package name. link: URL to the package web page, e.g. its repository site. description: Descriptions of the feed, e.g. the package. language: Optional, a valid RSS language code, see http://www.rssboard.org/rss-language-codes atom: Optional, full URL to the RSS feed on the web, used for atom:link rel="self".

Value

No return value, writes a file or to stdout()

Examples

```
## Not run:
channel.info <- c(
  title="roxyPackage",
  link="http://R.reaktanz.de/pckg/roxyPackage",
  description=roxyPackage:::pckg.dscrptn[["Description"]],
  atom="http://R.reaktanz.de/pckg/roxyPackage/rss.xml")
rss.tree <- news2rss("~/R/roxyPackage/NEWS.Rd",
  channel=channel.info)

## End(Not run)
```

---

readChangeLog *Read/write ChangeLog files*

Description

These functions ans methods can be used to manage ChangeLog files.

Usage

```
readChangeLog(file, head = "ChangeLog for package",
  change = "changes in version", item = "  -")

writeChangeLog(log, file = NULL, head = "ChangeLog for package",
  change = "changes in version", item = "  -", lineEnd = 78)

initChangeLog(entry = list(changed = c("initial release"), fixed =
  c("missing ChangeLog")), package = "unknown", version = "0.01-1",
  date = Sys.Date())
```

Arguments

file	Character string, path to the ChangeLog file to read.
head	Character string, the headline text of the ChangeLog file (without the package name).
change	Character string, the text introducing each ChnageLog entry for a package version.
item	Character string, the text marking each entry item.
log	An object of class ChangeLog.
lineEnd	Integer number, indicates where to do line breaks.

entry	A (named) list of character vectors. The element names will become the ChangeLog sections, each vector element an item.
package	Character string, the package name.
version	Character string, version number to look up.
date	The date of the ChangeLog entry in YYYY-MM-DD format. will be coerced into character. To keep the date stamp of a present entry, set date=NULL.

Details

The ChangeLog files used for R packages are usually required to have a standard format, if they are supposed to be parsed by functions like tools::news2Rd:

1. entries are named "Changes in version <version number>" (and optionally a YYYY-MM-DD date string afterwards). The date string is mandatory if you want to use the ChageLog functions in roxyPackage. The version number can be given in both <major>.<minor>-<revision> or <major>.<minor>.<revision> format.
2. they have single changes properly itemized, by indentation and then either "o", "-" or "*" followed by space
3. optionally have categories as subsections, like "Fixed", "Changed" or "Added"

readChangeLog tries to read a given ChangeLog file and parse its content to generate a special ChangeLog object.

writeChangeLog takes such a ChangeLog object to write it back to a file. If file=NULL, the log will be returned to stdout.

initChangeLog generates a ChangeLog object from scratch, e.g., to get started with a new package.

Value

An object of class ChangeLog.

See Also

getChangeLogEntry, updateChangeLog

Examples

```
## Not run:
changelog <- readChangeLog("/home/user/myRsources/myRpackage/ChangeLog")

## End(Not run)
```

roxy.package *Automatic doc creation, package building and repository update*

Description

This function should help to create R packages with full documentation and updates to a local repository. It supports source and binary packaging (Windows and Mac OS X; see Note section on the limitations).

Usage

```
roxy.package(pck.source.dir, pck.version, pck.description, R.libs, repo.root,
  pck.date = Sys.Date(), actions = c("roxy", "package"), cleanup = FALSE,
  rm.vignette = FALSE, R.homes = R.home(),
  html.index = "Available R Packages", html.title = "R package",
  Rcmd.options = c(install = "--install-tests", build =
  "--no-manual --no-build-vignettes --md5", check = "--as-cran", Rd2pdf =
  "--pdf --no-preview"), URL = NULL, deb.options = NULL,
  readme.options = NULL, ChangeLog = list(changed = c("initial release"),
  fixed = c("missing ChangeLog")), Rbuildignore = NULL, Rinstignore = NULL,
  OSX.repo = list(main = "contrib", symlinks = "mavericks"), ...)
```

Arguments

pck.source.dir	Character string, path pointing to the root directory of your package sources.
pck.version	Character string, defining the designated version number. Can be omitted if actions don't include "roxy", then this information is read from the present DESCRIPTION file.
pck.description	Data frame holding the package description (see Examples section).
R.libs	Character string, valid path to the R library where the package should be installed to.
repo.root	Character string, valid path to a directory where to build/update a local package repository.
pck.date	Date class object or character string of the release date in YYYY-MM-DD format. Defaults to Sys.Date(). If actions don't include "roxy" and neither Date, Packaged, nor Date/Publication are found in the present DESCRIPTION file, then pck.date will be used. Otherwise, the information from the DESCRIPTION file is used.
actions	Character vector, must contain at least one of the following values: **"roxy"** Roxygenize the docs **"cite"** Update CITATION file **"license"** Update LICENSE file **"readme"** Generate initial README.md file

"check" Do a full package check, calling `R CMD check`. Combine with `"package"` to do the check on the tarball, not the source directory.

"package" Build & install the package, update source repository, calling `R CMD build` and `R CMD INSTALL`

"binonly" Like `"package"`, but doesn't copy the source package to the repository, to enable binary-only rebuilds

"cl2news" Try to convert a ChangeLog file into an NEWS.Rd file

"news2rss" Try to convert `inst/NEWS.Rd` into an RSS feed. You must also set `URL` accordingly.

"doc" Update PDF documentation and vignette (if present), `R CMD Rd2pdf` (or `R CMD Rd2dvi` for R < 2.15)

"html" Update HTML index files

"win" Update the Windows binary package

"macosx" Update the Mac OS X binary package

"log" Generate initial ChangeLog or update a present ChangeLog file

"deb" Update the Debian binary package with `debianize` (works only on Debian systems; see `deb.options`, too). `URL` must also be set to generate Debian repository information.

"cleanRd" Insert line breaks in Rd files with lines longer than 90 chars

Note that `"cl2news"` will write the `NEWS.Rd` file to the `inst` directory of your sources, which will overwrite an existing file with the same name! Also note that if both a `NEWS/NEWS.Rd` and `ChangeLog` file are found, only news files will be linked by the `"html"` action.

`cleanup` Logical, if `TRUE` will remove backup files (matching `.*~$` or `.*backup$`) from the source directory.

`rm.vignette` Logical, if `TRUE` and a vignette was build during the `"doc"` action and vignettes live in the directory `inst/doc`, they will not be kept in the source package but just be moved to the `./pckg/$PACKAGENAME` directory of the repository.

`R.homes` Path to the R installation to use. Can be set manually to build packages for other R versions than the default one, if you have installed them in parallel. Should probably be used together with `R.libs`.

`html.index` A character string for the headline of the global index HTML file.

`html.title` A character string for the title tag prefix of the package index HTML file.

`Rcmd.options` A named character vector with options to be passed on to the internal calls of `R CMD build`, `R CMD INSTALL`, `R CMD check` and `R CMD Rd2pdf` (or `R CMD Rd2dvi` for R < 2.15). Change these only if you know what you're doing! Will be passed on as given here. To deactivate, options must explicitly be se to `""`, missing options will be used with the default values.

`URL` Either a single character string defining the URL to the root of the repository (i.e., which holds the directories `src` etc., see below), or a named character vector if you need different URLs for different services. If you provide more than one URL, these are valid names for values:

`default` A mandatory fallback URL, will be used if not overridden by one of the other values. This is fully equivalent to the global value if only one character string is provided.

`debian` Used for the Debian package repository if different from the default.

`mirror.list` URL pointing to a list of mirrors users should choose from, rather than using one particular host name for the Debian repository. Will only be used in the HTML instructions for a Debian repository.

`debian.path` Can be used to define a custom path users would need to specify in addition to the main URL. Defaults to `"/deb"`, and if given, it must start with a slash. Will be used in combination with `default`, `debian` or `mirror.list`. It is not advisable to combine it with `default`, because you will have to manually rename the directory generated after each run!

These URLs are not the path to the local file system, but should be the URLs to the respecive repository as it is available via internet. This option is neccessary for (and only interpreted by) the actions `"news2rss"`, `"deb"`, and possibly `"html"` – if `flattrUser` is also set in `readme.options`, a Flattr button will be added to the HTML page, using the default URL.

`deb.options` A named list with parameters to pass through to `debianize`. By default, `pck.source.dir` and `repo.root` are set to the values given to the parameters above. As for the other options, if not set, the defaults of `debianize` will be used.

`readme.options` A named list with parameters that add optional extra information to an initial README.md file, namely instructions to install the package directly from a GitHub repository, and a Flattr button. Ignore this if you don't use either. Theoretically, you can overwrite all values of the internal function `readme_text` (e.g., try `formals(roxyPackage:::readme_text)`). But in practice, these two should be all you need to set:

`githubUser` Your GitHub user name, can be used both to contruct the GitHub repo URL as well as the Flattr URL

`flattrUser` Your Flattr user name, also used by the `"html"` action in combination with `URL`

All other missing values are then guessed from the other package information. It is then assumed that the GitHub repo has the same name as the package.

`ChangeLog` A named list of character vectors with log entry items. The element names will be used as section names in the ChangeLog entry, and each character string in a vector will be pasted as a log item. The news you provide here will be appended to probably present news, while trying to prevent duplicate entries to appear. If you need more control, don't use the `"log"` action, but have a look at `updateChangeLog`. Also note that the date of altered entries will be updated automatically, unless you don't call the `"roxy"` action, too.

`Rbuildignore` A character vector to be used as lines of an `.Rbuildignore` file. If set, this will replace an existing `.Rbuildignore` file. Setting it to an empty string (`""`) will remove the file, the default value `NULL` will simply keep the file, if one is present.

`Rinstignore` A character vector to be used as lines of an `.Rinstignore` file. If set, this will replace an existing `.Rinstignore` file. Setting it to an empty string (`""`) will remove the file, the default value `NULL` will simply keep the file, if one is present.

`OSX.repo` A named list of character vectors, one named `"main"` defines the main directory below `./bin/macosx/` where packages for Mac OS X should be copied,

and the second optional one named "symlink" can be used to set symbolic links, e.g., symlinks="mavericks" would also make the repository available via ./bin/macosx/mavericks. Symbolic links will be ignored when run on on Windows. If you use them, make sure they're correctly transferred to your server, where applicable.

... Additional options passed through to roxygenize.

Details

For the documentation roxygen2[1] is used. Next to the actual in-line documentation of the package's contents, you only need to prepare a data.frame to be used to write a package DESCRIPTION file. See the example section for details on that. This means that you *neither* edit the DESCRIPTION *nor* the *-package.R file manually, they will both be created *automatically* by this function with contents according to these settings!

Sandboxing

If you want to check out the effects of roxy.package() without touching you actual package sources, try sandbox to set up a safe testing environment.

Repository layout

The repository will have this directory structure, that is, below the defined repo.root:

./src/contrib Here go the source packages

./bin/windows/contrib/$RVERSION Here go the Windows binaries

./bin/macosx/contrib/$RVERSION Here go the Mac OS X binaries (see OSX.repo for further options)

./pckg/index.html A global package index with links to packages' index files, if actions included "html"

./pckg/web.css A CRAN-style CSS file, if actions included "html"

./pckg/$PACKAGENAME Here go documentation PDF and vignette, as well as a ChangeLog file, if found. and an index.html with package information, if actions included "html". This is probably a bit off-standard, but practical if you several packages.

Converting ChangeLogs into NEWS

See cl2news for details.

Build for several R versions

The options R.libs and R.homes can actually take more than one string, i.e., a vector of strings. This can be used to build packages for different R versions, provided you installed them on your system. If you're running GNU/Linux, an easy way of doing so is to fetch the R sources from CRAN, calling "./configure" with something like "--prefix=$HOME/R/<R version>", so that "make install" installs to that path. Let's assume you did that with R 3.2.2 and 3.1.3, you could then call roxy.package with options like R.homes=c("home/user/R/R-3.2.2", "home/user/R/R-3.1.3")

and `R.libs=c("home/user/R/R-3.2.2/lib64/R/library", "home/user/R/R-3.1.3/lib64/R/library")`. `roxy.package` will then call itself recursively for each given R installation.

One thing you should be aware of is that `roxy.package` will not perform all actions each time. That is because some of them, namely `"roxy"`, `"cite"`, `"license"`, `"doc"`, `"cl2news"`, `"news2rss"`, `"cleanRd"`, and `"readme"`, would overwrite previous results anyway, so they are only considered during the first run. Therefore, you should always place the R version which should be used for these actions first in line. The `"html"` action will list all Windows and OS X binary packages. The `"deb"` action will only actually debianize and build a binary package during the first run, too.

Windows

On Windows, the actions `"doc"` and `"check"` will only work correctly if you have installed and configured LaTeX accordingly, and you will also need Rtools set up for packaging.

CRAN compliance

The CRAN policies can sometimes be very strict. This package should allow you to produce packages which are suitable for release on CRAN. But some steps have to be taken care of by yourself. For instance, CRAN does currently not allow copies of common licenses in a source package, nor a `debian` folder. Therefore, if your package is supposed to be released on CRAN, you should include `Rbuildignore=c("debian", "LICENSE")` to the function call.

Note

The binary packaging is done simply by zipping (Windows) or targzipping (Mac OS X) the built and installed package. This should do the trick as long as your package is written in pure R code. It will most likely not produce usable packages if it contains code in other languages like C++.

References

[1] http://cran.r-project.org/package=roxygen2

See Also

sandbox to run roxy.package() in a sandbox.

Examples

```
## Not run:
## package description as data.frame:
pckg.dscrptn <- data.frame(
  Package="SquareTheCircle",
  Type="Package",
  Title="Squaring the circle using Heisenberg compensation",
  Author="Ernst Dölle [aut, cre, cph], Ludwig Dölle [trl,
      ctb] (initial translation to whitespace)",
  AuthorR="c(person(given=\"Ernst\", family=\"Dölle\",
       email=\"e.a.doelle@example.com\",
       role=c(\"aut\", \"cre\", \"cph\")),
     person(given=\"Ludwig\", family=\"Dölle\",
       role=c(\"trl\", \"ctb\"),
```

```
      comment=\"initial translation to whitespace\")
   )",
  Maintainer="E.A. Dölle <doelle@eternalwondermaths.example.org>",
  Depends="R (>= 2.10.0),heisenberg (>= 0.23),tools",
  Enhances="rkward",
  Description="This package squares the circle using Heisenberg compensation.
      The code came from a meeting with Yrla Nor that i had in a dream. Please
      don't forget to chain your computer to the ground, because these
      algorithms might make it fly.",
  License="GPL (>= 3)",
  Encoding="UTF-8",
  LazyLoad="yes",
  URL="http://eternalwondermaths.example.org",
  stringsAsFactors=FALSE)
# hint no. 1: you *don't* specify version number and release date here,
#   but all other valid fields for DESCRIPTION files must/can be defined
# hint no. 2: most of this rarely changes, so you can add this to the
#   internals of your package and refer to it as
#   roxy.package(pck.description=SquareTheCircle:::pckg.dscrptn, ...)
# hint no. 3: use "AuthorR" for the "Author@R" field, or "AuthorsR" for
# R >= 2.14, to work around naming problems

roxy.package(pck.source.dir="~/my_R_stuff/SquareTheCircle",
  pck.version="0.01-2",
  pck.description=pckg.dscrptn,
  R.libs="~/R",
  repo.root="/var/www/repo",
  actions=c("roxy", "package", "doc"))

## End(Not run)
```

---

sandbox *Run actions in a sandbox*

Description

If you want to test the effects of `roxy.package`, `archive.packages` or `debianize`, you can activate a sandbox with this function.

Usage

```
sandbox(active = FALSE, sandbox.dir = file.path(tempdir(), "roxyPackge",
  "sandbox"), pck.source.dir = TRUE, R.libs = TRUE, repo.root = TRUE,
  archive = repo.root)
```

Arguments

`active`	Logical, whether sandboxing should be active or not
`sandbox.dir`	Character string, full path to the sandbox root directory to use. Will be created if neccessary (at first use, not when setting this here!).

`pck.source.dir` Logical, whether to sandbox the package sources. If `TRUE` the full package sources will be copied to `file.path(sandbox.dir, "src")` (at first use, not when setting this here!).

`R.libs` Logical, whether to sandbox the R library directory, that is, the directory to install the package to. Since this needs also to provide all package dependencies, those packages will be copied to `file.path(sandbox.dir, "R")` (at first use, not when setting this here!).

`repo.root` Logical, whether to sandbox the repository. This repository will be set up in `file.path(sandbox.dir, "repo")` (at first use, not when setting this here!).

`archive` Logical, whether to sandbox the repository archive. The archive will be set up in `file.path(sandbox.dir, "repo_archive")` (at first use, not when setting this here!).

Details

Sandboxing means that you are able to specify which groups of actions should only be run in a separate environment. This can be useful if you don't want to make changes to your actual package code, but inspect the result first.

With this function, you can turn sandboxing on and off. This setting has effects only in the currently running R session. By default, sandboxing is off.

Value

Settings are stored in an internal environment, so there is no actual return value.

See Also

`sandbox.status` to see the current settings.

Examples

```
## Not run:
# turn sandboxing on
sandbox(active=TRUE)

## End(Not run)
```

---

sandbox.status *Show sandboxing status*

Description

This function prints the current sandbox settings. It has no parameters.

Usage

```
sandbox.status()
```

Value

The function invisibly returns the sandbox root directory path (`sandbox.dir`). If sandboxing is inactive, this is an empty character string (`""`).

See Also

sandbox to change these settings.

Examples

```
## Not run:
sandbox.status()

## End(Not run)
```

`templateFile` *Create template file for new function/class/method*

Description

This function can be used to generate template files for new functions, S4 classes or methods.

Usage

```
templateFile(name, path = getwd(), pck.description = data.frame(Package =
  "", Author = "", License = "GPL (>= 3)", stringsAsFactors = FALSE),
  year = format(Sys.time(), "%Y"), params = list(obj = "someClass", ... =
  "\\code{\\link[somepackage]{somefunction}}"), seealso = list(aPackage =
  "aFunction"), return = list(aPackage = "aFunction"), type = "function",
  write = FALSE, overwrite = FALSE)
```

Arguments

`name`	Character string, name of the function/method/class.
`path`	Full path to the directory where the file should be added if `write=TRUE`.
`pck.description`	Data frame holding the package description. Only the fields "Package", "Author" and "License" are needed/used (see Examples section).
`year`	Character string, the year to use in the copyright information.
`params`	A named list of parameters, where the element name is the parameter name and its value is the type of object expected as input. Some objects types are recognized, like "c", "character", "numeric", "logical", "matrix" or "data.frame". If an element is called `"..."`, the value is assumend to point to a function or method where additional arguments are passed to. If `type="S4class"`, this argument is used to define the slots.

seealso	A named list, where element names define packages and values objects of that package to link.
return	A named list, similar to seealso, but generating references for returned objects.
type	Character string, either "function", "S4class", or "S4method", depending on the template you want to create.
write	Logical, if TRUE output will be written to a file in path, otherwise returned as a character string.
overwrite	Logical, if TRUE and the output file already exists, it will be replaced with the generated template. Otherwise you'll just get an error.

Details

Set the parameters to your needs, perhaps setwd into the target directory, and set write=TRUE if you like what you see so far. The result should include a copyright note, insitial roxygen-style documentation and some useful first lines of code, guessed from the provided arguments.

Value

If write=TRUE, writes a file in the path directory. If write=FALSE, returns a character string.

Examples

```
pckg.dscrptn <- data.frame(
  Package="SquareTheCircle",
  Author="E.A. Dölle <doelle@eternalwondermaths.example.org>",
  License="GPL (>= 3)",
  stringsAsFactors=FALSE
)
cat(
  templateFile(
    name="exampleFunction",
    pck.description=pckg.dscrptn
  )
)
```

---

updateChangeLog *Update ChangeLog objects*

Description

This method can be used to update ChangeLog objects.

Usage

```
updateChangeLog(log, entry, version, date = Sys.Date(), append = TRUE)

## S4 method for signature 'ChangeLog'
updateChangeLog(log, entry, version, date = Sys.Date(),
  append = TRUE)
```

Arguments

log	An object of class ChangeLog.
entry	A (named) list of character vectors. The element names will become the ChangeLog sections, each vector element an item.
version	Character string, version number to look up.
date	The date of the ChangeLog entry in YYYY-MM-DD format. will be coerced into character. To keep the date stamp of a present entry, set date=NULL.
append	Logical, whether a present entry should be replaced or added to.

Details

updateChangeLog takes a ChangeLog object and a version number string, replaces the complete entry with the contents of entry and updates the time stamp to date.

Value

An object of class ChangeLog.

See Also

readChangeLog

Index