Package ‘rbundler’

February 20, 2015

Maintainer Yoni Ben-Meshulam <yoni.bmesh@gmail.com>

Author Yoni Ben-Meshulam <yoni.bmesh@gmail.com>

Version 0.3.7

License GPL-3

Title Rbundler manages an application's dependencies systematically and repeatedly.

Description Rbundler manages a project-specific library for dependency package installation. By specifying dependencies in a DESCRIPTION file in a project's root directory, one may install and use dependencies in a repeatable fashion without requiring manual maintenance. rbundler creates a project-specific R library in `PROJECT_ROOT/.Rbundle` (by default) and a project-specific `R_LIBS_USER` value, set in `PROJECT_ROOT/.Renviron`. It supports dependency management for R standard ``Depends'', ``Imports'', ``Suggests'', and ``LinkingTo'' package dependencies. rbundler also attempts to validate and install versioned dependencies, such as ``>='', ``=='', ``<=''. Note that, due to the way R manages package installation, differing nested versioned dependencies are not allowed. For example, if your project depends on packages A (== 1), and B (== 2), but package A depends on B (== 1), then a nested dependency violation will cause rbundler to error out.

Depends R (>= 2.15.1), devtools (>= 1.3)

Suggests testthat (>= 0.8), roxygen2

NeedsCompilation no

Repository CRAN

Date/Publication 2014-05-08 08:02:10

R topics documented:

`bundle` *Bundles a package and it's dependencies into a library.*

Description

Dependencies are installed into the package's bundle library. The library is also added to this session's .libPaths.

Usage

```
bundle(pkg = ".", bundle_path = file.path(pkg, ".Rbundle"),
  overwrite = FALSE, dependencies = c("Depends", "Imports", "LinkingTo",
  "Suggests"))
```

Arguments

`pkg`	package description, can be path or package name.
`bundle_path`	path to the bundle. Defaults to '.Rbundle' under the package directory
`overwrite`	whether to delete the existing bundle library and re-install all packages. This can be necessary when upgrading or downgrading package dependencies. Defaults to FALSE
`dependencies`	which package dependencies to install. Defaults to c("Depends", "Imports", "LinkingTo", "Suggests")

Details

Note that repository and pkgType options are temporarily overridden, according to the user's options, and set back to their previous values after bundle completes.

Examples

```
## Not run:
# Run bundle in the current path:
bundle()
# Check for the new `.Rbundle` entry in `.libPaths`:
.libPaths()

lib <- file.path(tempdir(), 'my_bundle_lib')
# Run bundle in the current path, overriding the target library:
bundle('.', lib)
# Check for the new entry in `.libPaths`:
.libPaths()

## End(Not run)
```

`compare_versions` *Compares the requested version to the available version using the compare operator.*

Description

Compares the requested version to the available version using the compare operator.

Usage

```
compare_versions(requested, compare, version)
```

Arguments

`requested`	the requested version
`compare`	the comparison operator
`version`	the available version

`construct_r_libs_user` *Constructs a new R_LIBS_USER setting using the current libraries and the new bundle library.*

Description

Constructs a new R_LIBS_USER setting using the current libraries and the new bundle library.

Usage

```
construct_r_libs_user(bundle_path)
```

Arguments

`bundle_path`	the new bundle path

Value

r_libs_user colon-separated libraries

`create_mock_packages` *Creates a series of mock packages, useful for testing and experimentation.*

Description

Creates a series of mock packages, useful for testing and experimentation.

Usage

```
create_mock_packages(path, dependency, repos = getOption("repos"))
```

Arguments

`path`	the path in which to create the mock packages
`dependency`	the dependency to create in the mock packages
`repos`	the repositories to use for the contrib.url path

Value

a list of named packages, each of which corresponds to the devtools ‘as.package‘ object

Examples

```
path <- tempdir()
repos <- 'http://cran.rstudio.com'
dependency <- mock_dependency('RCurl', repos)
create_mock_packages(path, dependency, repos)
```

create_package — *Creates a package matching the given description and dependencies.*

Description

Creates a package matching the given description and dependencies.

Usage

```
create_package(name, title, dependencies, path = ".")
```

Arguments

`name`	the package name
`title`	the package title
`dependencies`	a data.frame of package dependencies, including the package names, comparators, and versions
`path`	the path in which to create the package. Defaults to the current path

Value

the package, as constructed using the ‘devtools‘ ‘as.package‘ function

Examples

```
# Create a simple package with no dependencies:
path <- tempdir()
name <- 'simplepackage'
package <- create_package(name, 'A simple mock package', data.frame(), path)
```

create_package_description — *A Utility function for creating rbundler scenarios.*

Description

A Utility function for creating rbundler scenarios.

Usage

```
create_package_description(name, title, dependencies)
```

Arguments

`name`	the name of the package to create
`title`	the title of the package to create
`dependencies`	a data.frame with dependency type, package, compare, version set.

Examples

```
name <- 'simpledependency'
title <- 'A mock package with a single dependency.'
dependencies <- data.frame(type = c('Depends', 'Suggests'), package=c('foo', 'bar'),
                           compare=c(NA, '=='), version=c(NA, '1'))
description <- create_package_description(name, title, dependencies)

write(description, file='') # Write the output to the console
```

`dependency_clauses` *Creates the ‘Depends:‘ clause by concatenating individual packages and adding their compare clauses.*

Description

Creates the ‘Depends:‘ clause by concatenating individual packages and adding their compare clauses.

Usage

```
dependency_clauses(dependencies)
```

Arguments

`dependencies`	a data.frame with dependency package, compare, and version set.

`determine_version_to_install` *Determines the version to install by comparing available versions to the required version and compare.*

Description

Determines the version to install by comparing available versions to the required version and compare.

Usage

```
determine_version_to_install(available_versions, version, compare)
```

Arguments

`available_versions`	a vector of version identifiers corresponding to all versions of this package
`version`	the version requested
`compare`	the compare requested

`find_available_versions` *Retrieves a list of available versions for a package.*

Description

Retrieves a list of available versions for a package.

Usage

```
find_available_versions(package, repos = getOption("repos"),
  type = getOption("pkgType"))
```

Arguments

`package`	the package name
`repos`	character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as `"http://cran.us.r-project.org"`. Can be `NULL` to install from local files (with extension '`.tar.gz`' for source packages).
`type`	character, indicating the type of package to download and install. Possible values are `"source"`, `"mac.binary.leopard"` and `"win.binary"`: the binary types can be listed and downloaded but not installed on other platforms. The default is the appropriate binary type on Windows and on the CRAN binary OS X distribution, otherwise `"source"`. For the platforms where binary packages are the default, an alternative is `"both"` which means 'try binary if available, otherwise try source'. (This will only choose the binary package if its version number is no older than the source version. In interactive use it will ask before attempting to install source packages.)

`install_version` *Install specified version or relative version of a CRAN package.*

Description

If you are installing an package that contains compiled code, you will need to have an R development environment installed. You can check if you do by running `has_devel`.

Usage

```
install_version(package, version = NA, compare = NA,
  repos = getOption("repos"), type = getOption("pkgType"), ...)
```

Arguments

`package`	package name
`version`	If the specified version is NA or the same as the most recent version of the package, this function simply calls `install`. Otherwise, it looks at the list of archived source tarballs and tries to install an older version instead.
`compare`	If specified, and if the version is specified, enforces comparison of the package version. Valid values: ==, <, >, >=, or <=
`...`	Other arguments passed on to `install`.
`repos`	character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as `"http://cran.us.r-project.org"`. Can be `NULL` to install from local files (with extension '`.tar.gz`' for source packages).
`type`	character, indicating the type of package to download and install. Possible values are `"source"`, `"mac.binary.leopard"` and `"win.binary"`: the binary types can be listed and downloaded but not installed on other platforms. The default is the appropriate binary type on Windows and on the CRAN binary OS X distribution, otherwise `"source"`. For the platforms where binary packages are the default, an alternative is `"both"` which means 'try binary if available, otherwise try source'. (This will only choose the binary package if its version number is no older than the source version. In interactive use it will ask before attempting to install source packages.)

Details

Note: This is an updated version of devtools 'install_version' It has been fixed to work with the latest CRAN repository and updated to support version comparisons (i.e. >, ==, <, etc.)

Value

whether the version was installed

Author(s)

Jeremy Stephens

Yoni Ben-Meshulam

`load_available_packages` *Loads available packages from the given repository.*

Description

Loads available packages from the given repository.

Usage

```
load_available_packages(repos)
```

Arguments

`repos`	character vector, the base URLs of the repositories to use

Value

data.frame of available packages

`mock_dependency` *Creates a mock dependency - corresponding to a real package - for use in testing and experimentation.*

Description

Creates a mock dependency - corresponding to a real package - for use in testing and experimentation.

Usage

```
mock_dependency(name = "tempdisagg", repos = getOption("repos"))
```

Arguments

`name`	the name of the package dependency
`repos`	the repositories to use for the contrib.url path

Value

list with the name and version of the dependency

rbundler — *A package dependency management utility.*

Description

Rbundler is an R package dependency management utility.

Author(s)

Yoni Ben-Meshulam <yoni.bmesh@gmail.com>

Examples

```
## Not run:
# Run bundle in the current path:
bundle()
# Check for the new `.Rbundle` entry in `.libPaths`:
.libPaths()

lib <- file.path(tempdir(), 'my_bundle_lib')
# Run bundle in the current path, overriding the target library:
bundle('.', lib)
# Check for the new entry in `.libPaths`:
.libPaths()

## End(Not run)
```

read_archive_rds — *Loads archive from CRAN-like repositories. Returns empty list for non-CRAN (i.e. flat) repositories.*

Description

Loads archive from CRAN-like repositories. Returns empty list for non-CRAN (i.e. flat) repositories.

Usage

```
read_archive_rds(repos)
```

Arguments

`repos`	character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as `"http://cran.us.r-project.org"`. Can be `NULL` to install from local files (with extension ‘`.tar.gz`’ for source packages).

`update_current_environment` *Updates the current environment.*

Description

Updates the current environment.

Usage

```
update_current_environment(lib, r_libs_user)
```

Arguments

`lib`	the R library to add.
`r_libs_user`	the new value of R_LIBS_USER

`update_renviron_file` *Updates a .Renviron file in the given path.*

Description

Updates a .Renviron file in the given path.

Usage

```
update_renviron_file(path, r_libs_user)
```

Arguments

`path`	to the .Renviron file
`r_libs_user`	the new value of R_LIBS_USER

`validate_compare` *Validates the compare clause.*

Description

Validates the compare clause.

Usage

```
validate_compare(compare)
```

Arguments

`compare`	the compare clause to validate.

`validate_installed_package`

Checks whether a package has already been installed. If it has, and if the version corresponds to the required package version, then it returns TRUE. If it has been installed and the version does not correspond to the required version, then it throws an exception. Otherwise, it returns false.

Description

Checks whether a package has already been installed. If it has, and if the version corresponds to the required package version, then it returns TRUE. If it has been installed and the version does not correspond to the required version, then it throws an exception. Otherwise, it returns false.

Usage

```
validate_installed_package(package, version, compare)
```

Arguments

`package`	the package to check
`version`	the required version
`compare`	the comparison operator

Value

whether we should install the package

Index