Refresh / Enhance Library and Package Functions

Two files are provided on the github, @ColinGoodall1/editUtils, one with the original versions of 6 functions, the other with lightly-edited versions of the 6 functions.

For a function-by-function view of the diffs, see R, and compare the two branches.

find.package

install.packages

.libPaths

library

old.packages

update.packages

**Motivation and Description of Edits**

The set of package / library functions has in many ways stood the test of time. But the functions can be confusing and unintuitive.

It's been a while since I did this. Hark back to the days of John, Rick, and Allan and providing lots of S comments/bug reports, in the 80s.

Some specific concerns are addressed by new function arguments and corresponding light code edits.

(1) When there is a base package, eg nlme, in R\_HOME/library and a newer version of the same package in R\_USER\_LIBS, then old.packages() will repeatedly advise that the R\_HOME/library version is old, without explanation or change. To fix this, argument noDupl is set to default TRUE.

(2) R on Windows will attempt to install to R\_HOME/library. This is not wanted. The user may have administrative privileges, but, unlike su/sudo in UNIX, this provides at most the possibility of installing into C:/Program Files/... Typically, program installation generates a splash screen and the question do you want this progrm to make changes to your device? That is not what update.packages() or install.packages() does. It is fairly futile to attempt to change permission properties in C:/Program Files/R/x.y.z to allow updates to the base packages in situ.

Thus, argument noInstallRHOME is set to default TRUE. This codifies what happens in Windows anyway, without the confusion.

Generally, it makes sense to keep the install of R under R\_HOME intact, as an unchanged copy of the original install. For all OS's.

(3) library(), require() and find.package() will return the first instance of the package on .libPaths. This is not desirable if the newer version is further on the path. With argument latest = TRUE, the default, the package that is loaded / returned will have the highest version string. The code snippet in find.package() standardizes formating using "valid\_package\_version\_regexp".

With latest = FALSE (the original behavior) and R\_HOME at the end of .libPaths(), then, if ever one of the R\_HOME library packages is updated with the update put in say the personal library, then the package from R\_HOME will never ever be used again, even after installing a new version of R. With latest = TRUE that idiosyncracy goes away.

(4) A second new argument to find.package() is showVersion, with default FALSE, which provides a quick list of all the installed packages with the given names. An alternative with more extensive results is to filter the output of installed.packages()

(5) In my experience, more complex packages might not install properly, or when installed might not load, and when loaded some functions might not run. An example is the XLConnect version jump from 0.2-15 to 1.0.0 and then 1.0.1. Problems with the redis package are mentioned on the web. The mechanism 00LOCK goes only part way to remedy this. The new argument saveOld to install.packages() and update.packages() suggests a possible directory 01SAVE in each library.

I haven't developed this further, but can imagine:

.libPaths() will include each existing library/01SAVE/,

find.packages() will search both the library/ and the library/01SAVE/,

(new) function move.packages() will move packages from library/01SAVE/ to library/, and

remove.packages() will cleanup 01SAVE/.

By default, any risky package, requiring say a particular version of Java or some specific include files, should have saveOld set to TRUE, thus saveOld could be named useSaveOld instead.

(6) For clarity, in install.packages.R, I added a message to install a make program / Rtools in the appropriate place

(7) Improved text for Windows FAQ #2.8, suggestion as follows:

https://cran.r-project.org/bin/windows/base/rw-FAQ.html

*existing text*:-

2.8 What’s the best way to upgrade?

That’s a matter of taste. For most people the best thing to do is to uninstall R (see the previous Q), install the new version, copy any installed packages to the library folder in the new installation, run update.packages(checkBuilt=TRUE, ask=FALSE) in the new R and then delete anything left of the old installation. Different versions of R are quite deliberately installed in parallel folders so you can keep old versions around if you wish.

For those with a personal library (folder R\win-library\x.y of your home directory, R\win64-library\x.y on 64-bit builds), you will need to update that too when the minor version of R changes (e.g. from 3.0.2 to 3.1.0). A simple way to do so is to copy (say) R\win-library\3.0 to R\win-library\3.1 before running update.packages(checkBuilt=TRUE, ask=FALSE).

*proposed text*:-

2.8 What’s the best way to upgrade?

That’s a matter of taste. Different versions of R are quite deliberately installed in parallel folders, under Program Files/R, or possibly Program Files (x86)/R for 32 bit versions, so you can keep old versions around if you wish. Or, you can select "Add or Remove Programs" in Settings and select the version of R you wish to Uninstall. If you have installed any packages directly into the library or site-library folders (see ?.Library) of an older installation of R, then copy the package folders to the corresponding library folder in the new installation. There is no need to copy the package folders that are part of the installation.

Typically in Windows you will install additional packages and updates into a personal or shared library. This is because Windows is restrictive of install privileges, which are invoked when installing R, but generally not for additions and updates. Run update.packages(checkBuilt=TRUE, ask=FALSE). By default, this will not put updates in Program Files/R/R-x.y.z/library, and will prompt for a personal library instead.

See .libPaths() for a list of library folders, and .libPaths(new=...) to add a site-library.

The personal library defaults to Documents/R/win-library/x.y, with separate folders under each <package>/libs for i386 and x64 32- and 64-bit versions. For those with a personal library, you will need to update that too when the minor version of R changes (e.g. from 4.0.1 to 4.1.0). A simple way to do so is to copy (say) R/win-library/4.0 to R/win-library/4.1 before running update.packages(checkBuilt=TRUE, ask=FALSE).

**New Function Arguments**

*noDupl* skip duplicate packages, default TRUE

argument to old.packages and update.packages (which passes the argument to old.packages)

the default TRUE is the same for both packages, and likewise for other arguments

*noInstallRHOME* do not attempt to install in R\_HOME/library, default TRUE

argument to update.packages (passes to install.packages) and

install.packages (passes to .libPaths)

*noRHOME* do not include R\_HOME/library in .libPaths(), default FALSE

argument noInstallRHOME is passed as noRHOME from install.packages

*latest* return the latest library with given name,

otherwise return the first library in libPaths, default TRUE

argument to find.package() and library (which passes the argument to find.package)

*showVersion* return all versions of requested packages, default FALSE

argument to find.package

*saveOld* save the previous version of the library, in subfolder of 01SAVE/

argument to install.packages and update.packages (passes to install.packages)

**Issues**

internal objects cannot be referenced in an edited function (special compilation needed?). I haven't dipped into source code (yet).

(1) new .libPaths with additional argument noRHOME calls base:::.libPaths() to access .lib.loc

(2) installed.packages() references .instPkgFields etc, no edits provided

(3) install.packages(), update.packages() now reference utils:::simplifyRepos(), utils:::getDependencies()

(4) no digging into .install.winbinary() called from install.packages()

**Sample Calls**

base:::.libPaths()

.libPaths() # accesses .lib.loc through base:::.libPaths()

.libPaths(noRHOME=T)

.libPaths(new=...) # as before

args(old.packages)

old.packages()

old.packages(noDupl=F) # previous default

args(find.package)

find.package("class",verbose=T)

find.package("nlme",verbose=T) # two versions

find.package("nlme",verbose=T,latest=F)

find.package(c("class","nlme"),verbose=T,latest=T,showVersion=T)

find.package(c("class","nlme"),verbose=T,latest=F,showVersion=T)

update.packages()