Packages and their Management in R 2.1.0

by Brian D. Ripley

R 2.1.0 introduces changes that make packages and their management considerably more flexible and less tied to CRAN.

What is a package?

The idea of an R *package* was based quite closely on that of an S *library section*: a collection of R functions with associated documentation and perhaps compiled code that is loaded from a *library*[1] by the `library()` command. The Help Desk article (Ligges, 2003) provides more background.

Such packages still exist, but as from R 2.1.0 there are other types of R extensions, distinguished by the `Type:` field in the 'DESCRIPTION' file: the classic package has `Type: Package` or no `Type:` field. The `Type:` field tells `R CMD INSTALL` and `install.packages()` what to do with the extension, which is still distributed as a tarball or `zip` file containing a directory of the same name as the package.

This allows new types of extensions to be added in the future, but R 2.1.0 knows what to do with two other types.

Translation packages

with `Type: Translation`. These are used to add or update translations of the diagnostic messages produced by R and the standard packages. The convention is that languages are known by their ISO 639 two-letter (or less commonly three-letter) codes, possibly with a 'country or territory' modifier. So package **Translation-sl** should contain translations into Slovenian, and **Translation-zhTW** translations into traditional Chinese as used in Taiwan. *Writing R Extensions* describes how to prepare such a package: it contains compiled translations and so all `R CMD INSTALL` and `install.packages()` need to do is to create the appropriate subdirectories and copy the compiled translations into place.

Frontend packages

with `Type: Frontend`. These are only supported as source packages on Unix, and are used to install alternative front-ends such as the GNOME console, previously part of the R distribution and now in CRAN package **gnomeGUI**. For this type of package the installation runs `configure` and the `make` and so allows maximal flexibility for the package writer.

[1] a directory containing subdirectories of installed packages

Source *vs* binary

Initially R packages were distributed in source form as tarballs. Then the Windows binary packages came along (distributed as Zip files) and later Classic MacOS and MacOS X packages. As from R 2.1.0 the package management functions all have a `type` argument defaulting to the value of `getOption("pkgType")`, with known values `"source"`, `"win.binary"` and `"mac.binary"`. These distributions have different file extensions: `.tar.gz`, `.zip` and `.tgz` respectively.

This allows packages of each type to be manipulated on any platform: for example, whereas one cannot install Windows binary packages on Linux, one can download them on Linux or Solaris, or find out if a MacOS X binary distribution is available.

Note that `install.packages` on MacOS X can now install either `"source"` or `"mac.binary"` package files, the latter being the default. Similarly, `install.packages` on Windows can now install either `"source"` or `"win.binary"` package files. The only difference between the platforms is the default for `getOption("pkgType")`.

Repositories

The package management functions such as `install.packages`, `update.packages` and `packageStatus` need to know where to look for packages. Function `install.packages` was originally written to access a CRAN mirror, but other *repositories* of packages have been created, notably that of Bioconductor. Function `packageStatus` was the first design to allow multiple repositories.

We encourage people with collections of packages to set up a repository. Apart from the CRAN mirrors and Bioconductor there is an Omegahat repository, and I have a repository of hard-to-compile Windows packages to supplement those made automatically. We anticipate university departments setting up repositories of support software for their courses. A repository is just a tree of files based at a URL: details of the format are in *Writing R Extensions*. It can include one, some or all of `"source"`, `"win.binary"` and `"mac.binary"` types in separate subtrees.

Specifying repositories

Specifying repositories has become quite complicated: the (sometimes conflicting) aims were to be intuitive to new users, very flexible and as far as possible backwards compatible.

As before, the place(s) to look for files are specified by the `contriburl` argument. This is a character vector of URLs, and can include `file://` URLs if packages are stored locally, for example on a distribution CD-ROM. Different URLs in the vector can be of different types, so one can mix a distribution CD-ROM with a CRAN mirror.

However, as before, most people will not specify `contriburl` directly but rather the `repos` argument that replaces the `CRAN` argument.[2] The function `contrib.url` is called to convert the base URLs in `repos` into URLs pointing to the correct subtrees of the repositories. For example

```
> contrib.url("http://base.url",
              type = "source")
[1] "http://base.url/src/contrib"
> contrib.url("http://base.url",
              type = "win.binary")
[1] "http://base.url/bin/windows/contrib/2.1"
> contrib.url("http://base.url",
              type = "mac.binary")
[1] "http://base.url/bin/macosx/2.1"
```

This is of course hidden from the average user.

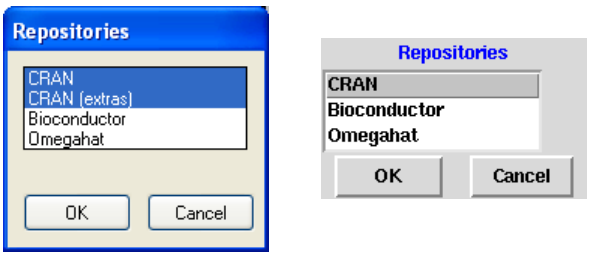

Figure 1: The listboxes from `setRepositories()` on Windows (left) and Unix (right).

The `repos` argument of the package management functions defaults to `getOption("repos")`. This has a platform-specific default, and can be set by calling `setRepositories`. On most systems this uses a listbox widget, as shown in figure 1. Where no GUI is available, a text list is used such as

```
> setRepositories()
--- Please select repositories for use
    in this session ---

1: + CRAN
2: + CRAN (extras)
3:   Bioconductor
4:   Omegahat

Enter one or more numbers separated by spaces
1: 1 4
```

The list of repositories and which are selected by default can be customized for a user or for a site: see `?setRepositories`. We envisaged people distributing a CD-ROM containing R and a repository, with `setRepositories()` customized to include the local repository by default.

Suppose a package exists on more than one repository?

Since multiple repositories are allowed, how is this resolved? The rule is to select the latest version of the package, from the first repository on the list that is offering it. So if a user had

```
> getOption("repos")
[1] "file://d:/R/local"
[2] "http://cran.us.r-project.org"
```

packages would be fetched from the local repository if they were current there, and from a US CRAN mirror if there is an update available there.

Figure 2: Selecting a CRAN mirror on Windows.

No default CRAN

Windows users are already used to selecting a CRAN mirror. Now 'factory-fresh' R does not come with a default CRAN mirror. On Unix the default is in fact

```
> getOption("repos")
    CRAN
"@CRAN@"
```

[2] This still exists but has been deprecated and will be removed in R 2.2.0.

Whenever `@CRAN@` is encountered in a repository specification, the function `chooseCRANmirror` is called to ask for a mirror (and can also be called directly and from a menu item on the Windows GUI). The Windows version is shown in Figure 2: there are MacOS X and Tcl/Tk versions, as well as a text-based version using `menu`.[3]

Experienced users can avoid being asked for a mirror by setting `options("repos")` in their '.Rprofile' files: examples might be

```
options(repos=
  c(CRAN="http://cran.xx.r-project.org"))
```

on Unix-alikes and

```
options(repos=
  c(CRAN="http://cran.xx.r-project.org",
    CRANextra=
      "http://www.stats.ox.ac.uk/pub/RWin"))
```

on Windows.

Although not essential, it is helpful to have a *named* vector of repositories as in this example: for example `setRepositories` looks at the names when setting its initial selection.

Installing packages

Users of the Windows GUI will be used to selecting packages to be installed from a scrollable list box. This now works on all platforms from the command line if `install.packages` is called with no `pkgs` (first) argument. Packages from all the selected repositories are listed in alphabetical order, and the packages from bundles are listed[4] individually in the form `MASS (VR)`.

The `dependencies` argument introduced in `R 2.0.0` can be used to install a package and its dependencies (and their dependencies ...), and R will arrange to install the packages in a suitable order. The `dependencies` argument can be used to select only those dependencies that are essential, or to include those which are suggested (and might only be needed to run some of the examples).

Installing all packages

System administrators often want to install 'all' packages, or at least as many as can be installed on their system: a standard Linux system lacks the additional software needed to install about 20 and a handful can only be installed on Windows or on Linux.

The new function `new.packages()` is a helpful way to find out which packages are *not* installed of those offered by the repositories selected. If called as `new.packages(ask = "graphics")` a list box is used to allow the user to select packages to be installed.

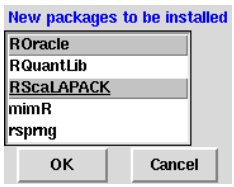

Figure 3: Selecting amongst uninstalled packages on Linux.

Updating packages

The only real change in `update.packages` is the `ask` argument. This defaults to `ask = TRUE` where as before the user is asked about each package (but with a little more information and the option to quit). Other options are `ask = FALSE` to install all updates, and `ask = "graphics"` to use a listbox (similar to figure 3) to de-select packages (as by default all the updates are selected).

Looking at repositories

The function `CRAN.packages` was (as it name implies) designed to look at a single CRAN mirror. It has been superseded by `available.packages` which can look at one or more repositories and by default looks at those specified by `getOption("repos")` for package type (source/binary) specified by `getOption("pkgType")`. This returns a matrix with rather more columns than before, including `"Repository"`, the dependencies and the contents of bundles.

Function `packageStatus` is still available but has been re-implemented on top of functions such as `available.packages`. Its `summary` method allows a quick comparison of the installed packages with those offered by the selected repositories. However, its `print` method is concise: see figure 4.

Bibliography

U. Ligges. R help desk: Package management. *R News*, 3(3):37–39, December 2003. URL `http://CRAN.R-project.org/doc/Rnews/`. 8

Brian D. Ripley
University of Oxford, UK
`ripley@stats.ox.ac.uk`

[3] `menu` has been much enhanced for R 2.1.0.

[4] provided the repository has been set up to supply the information.

```
> (ps <- packageStatus())
--- Please select a CRAN mirror for use in this session ---
Number of installed packages:

                     ok upgrade unavailable
  c:/R/library       34       1           0
  c:/R/rw2010/library 12       0          13

Number of available packages (each package/bundle counted only once):

                                                          installed
  http://www.sourcekeg.co.uk/cran/bin/windows/contrib/2.1          31
  http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.1        4

                                                          not installed
  http://www.sourcekeg.co.uk/cran/bin/windows/contrib/2.1             421
  http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.1            8

> summary(ps)

Installed packages:
-------------------
*** Library c:/R/library
$ok
 [1] "abind"       "acepack"     "akima"       "ash"         "car"
...
$upgrade
[1] "XML"

$unavailable
NULL

*** Library c:/R/rw2010/library
$ok
 [1] "base"      "datasets"  "graphics"  "grDevices" "grid"    "methods"  "splines"   "stats"    "stats4"
[10] "tcltk"     "tools"     "utils"

$upgrade
NULL

$unavailable
 [1] "boot"       "cluster"    "foreign"    "KernSmooth"  "lattice"
...

Available packages:
-------------------
(each package appears only once)

*** Repository http://www.sourcekeg.co.uk/cran/bin/windows/contrib/2.1
$installed
 [1] "abind"       "acepack"     "akima"       "ash"         "car"
...

$"not installed"
  [1] "accuracy"          "adapt"             "ade4"
...
[421] "zicounts"

*** Repository http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.1
$installed
[1] "GLMMGibbs" "xgobi"     "XML"       "yags"

$"not installed"
[1] "gsl"         "hdf5"        "ncdf"        "RDCOMClient"      "rgdal"  "RNetCDF"     "survnnet"    "udunits"

> upgrade(ps)
XML :
0.97-0 at c:/R/library
0.97-3 at http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.1
Update (y/N)?  y
```

Figure 4: `packageStatus` on a Windows machine. The voluminous `summary` has been edited to save space.