Packaging, Documentation and Testing

Kurt Hornik

Packaging

What are packages?

Packages are not libraries

Packages are not collections of R code (files)

Packages are standardized units for extending R

Package metadata

Packages must provide the following info to R:

- name, version
- license, description, title, author, maintainer

Currently via a 'DESCRIPTION' file in DCF (Debian Control File) format.

Example DESCRIPTION file

```
Package: pkgname
Version: 0.5-1
Date: 2004-01-01
Title: My first collection of functions
Author: Joe Developer <J.Developer@some.domain.net>, with
  contributions from A. User <A.User@whereever.net>.
Maintainer: Joe Developer <Joe.Developer@some.domain.net>
Depends: R (>= 1.8.0), nlme
Suggests: MASS
Description: A short (one paragraph) description of what
  the package does and why it may be useful.
License: GPL version 2 or newer
URL: http://www.r-project.org, http://www.another.url
```

Package structure

Packages (in source form) have a standardized hierarchical file system representation:

- A directory (with the same name as the package)
- Certain top-level files in that directory: ‘DESCRIPTION’ (must!), ‘COPYING’, ‘NAMESPACE’, ‘configure’/‘cleanup’, . . .
- Certain subdirectories of that directory: ‘R’, ‘man’, ‘data’, ‘src’, . . .

Package structure: details

- Subdirs ‘R’ and ‘man’ are for R code and documentation files
- Subdir ‘src’ is for foreign code that needs to be compiled/dynloaded.

  R can automatically create “shared libraries” for dynloading (DLLs) from C, C++ and Fortran code using Make

  Developers can prepend (Makevars) or append (Makefile) to the default mechanism

Package structure: details

- Subdir ‘data’ is for R “system data sets”, i.e., (collections of) data objects to be made available via `data()` (i.e., via `load/source/read.table`), not for “all data”
- Subdir ‘exec’ is for foreign code that needs to be interpreted (Shell, Perl, Tcl, . . .)
- Subdir ‘inst’ is “taken as is”

Installing packages

To be available for extending R, packages must be installed to libraries (directories where R knows to find installed packages), using e.g. `R CMD INSTALL` or `install.packages()`

Installing from source does several things, such as

- Preformat documentation objects in plain text and HTML formats;
- Create DLLs from foreign function code
- Maybe create a binary image of the R code
- Set up data structures with package index information

Currently, only minimal library-level install-time computing

Using packages

Function `library()` "loads" a package and adds it to the search path.

Important distinction: some functions (`help`, `data`) use only packages in the search path, others (`help.search`, `vignette`) use all *available* packages

Coming up: `usePackage()`

Creating packages

- "Just put everything where it should be"

- When starting from "just" R code, convenience function `package.skeleton()` as a starting point

Building packages

Packages are distributed as single file archiving their contents; creating this archive is called "building" the package (argh!)

Utility `R CMD build`

- Performs necessary cleanups and adds front-matter information (NOTE: 'Built' tag in DESCRIPTION, eventually checksums, ...)
- Creates the archive using a canonical file name

Can also have "binary packages" (more argh!)

Repositories

Packages can be distributed (over the web) via repositories (suitably indexed collections of packages).

Available repositories include CRAN, Bioconductor, Omegahat

Current package management tools in R

- allow for directly installing packages from repositories
- allow for automatically updating installed packages when newer versions are made available
- can do some package dependency handling

Package dependencies and reposTools

Dependencies are package metadata and registered in 'DESCRIPTION'. Currently, two categories:

Depends includes in particular everything needed to successfully load the package

Suggests needed "not necessarily" (e.g. suitably conditionalized code in examples or vignettes), but in order to exploit the full functionality of the package

Package reposTools from Bioconductor provides next generation client and server side tools for package/repository management

Packages, bundles and themes

Bundles are collections of packages to be distributed in a single archive (recommended packages MASS class nnet spatial come as the VR bundle)

Themes represent an ongoing Bioconductor development to create collections of packages (comparable to e.g. Debian's tasks)

Packages and frameworks

Currently, packages "cannot have a C-level API"

Other packages and R cannot find package headers and dynamic libraries (to link against, may be different from something dynloadable).

Well . . . let's just say impossibly hard.

D T Lang is working on frameworks

Documentation

Documentation

- R documentation objects
- Package vignettes
- Other

R documentation objects

"Things" providing in particular object reference documentation in a simple structured form

- Currently serialized files in R documentation (Rd) format ("Rd files"), with a syntax superficially resembling (La)TeX
- Mandatory entries

  `\name \alias \title \description \keyword`
- Other important entries

  `\usage \arguments \value \concept \examples`

Aliases

Aliases register the "topics documented" by an Rd object.

Correspondence: `help("foo")` (or `?"foo"`) looks for `\alias{foo}` entries.

I.e., for R functions or variables, the alias is just their name.

S4-related extensions of the question mark operator:

`class ? foo`	docs for class "foo"
`methods ? bar`	docs for methods for generic "bar"
`? baz(`*ARGLIST*`)`	docs about method to be used in call
`method ? baz(`*SIGLIST*`)`	docs about method with given siglist

Corresponding aliases for S4 classes and methods are of the form

```
CLASS-class
GENERIC,SIGLIST-method
```

Class-style vs methods-style documentation

Documentation shells can be generated using `prompt()`, `promptClass()` and `promptMethods()`.

Concepts and keywords

Keywords classify Rd objects according to a predefined tree (i.e., similar to “subject classifications” in maths/stats)

What about “real keywords”? R 1.8 has added the `\concept` markup for “concept index entries”, e.g.

```
\concept{Kendall correlation coefficient}
\concept{Pearson correlation coefficient}
\concept{Spearman correlation coefficient}
```

(cf. e.g. Texinfo concept vs function/variable index)

Searching the R documentation system/objects

`help()` and `?()` look for Rd objects with an alias matching exactly the topic determined from their arguments, by default in the packages in the search path.

`help.search()` looks for Rd objects with alias, concept or title (or name or keyword, or combinations of all these) matching a given char string, using either fuzzy matching or regular expression matching, by default in all available packages.

Vignettes

A vignette is an *integrated text document* for which R knows how to extract certain metadata (e.g., title and keywords) and the R code inside them.

Allows for various computations, such as e.g. inserting terminal or graphics output from running the code into the document.

Performed by the Sweave system in package utils, which currently knows at least about documents combining LaTeX and R using noweb or LaTeX syntax.

Applications: dynamic report generation, “live” textbooks for computing with R based on Bioconductor’s `vExplorer()`, ...

Package vignettes, vignette packages and compendia.

Other documentation

Every file in 'inst/doc' will be installed to 'doc'

But why not write a vignette?

Testing

Testing packages

> "Many thanks for your submission but did you run R CMD check on it?"
>
> "Hmm, why does your version of R CMD check show all these problems that mine does not know about?"

Can "test" packages using `R CMD check`

R CMD check

If possible, verifies installability as the basic test; then check

- availability and correctness of meta-information
- R code (syntactic correctness, common coding problems, consistency of S3 generics and methods, . . .)
- R documentation, including correctness, consistency, and completeness
- run-time behavior

etc., creating a standardized report

undoc

Early principle:

All user-level "objects" in a package must be documented.

I.e., everything shown by `objects()` must have (at least) an alias.

Trickier with "system" data sets, name spaces (can we have exported functions without a usage entry?), S4 classes and methods (how do we refer to the latter?) . . .

How to avoid the need for documenting

"I have 25 undocumented functions, but that's ok, they're really for internal use only . . ."

- Add a name space
- Use a leading dot for the names of these functions (`.foo`)
- Create a '*package*-internal.Rd' file

codoc

Find inconsistencies between actual and documented "structure" of R objects in a package.

- `codoc()` compares names and optionally also corresponding positions and default values of the arguments of functions (in general between code and `\usage` entries in the docs).

- `codocClasses()` and `codocData()` compare slot names of S4 classes and variable names of data sets, respectively.

Some unpleasant codoc details

As of R 1.9, documented default values are compared by default.

Special markup for indicating S3 and S4 methods:

`\method{`*GENERIC*`}{`*CLASS*`}(`*ARGLIST*`)`

(also works for replacement methods) and

`\S4method{`*GENERIC*`}{`*SIGLIST*`}(`*ARGLIST*`)`

if really needed . . . ("surprising arguments")

Did you know about `\synopsis`? If so . . .

Package computing vs reporting

Most of `R CMD check` is based on R code in package tools.

`R CMD check` is an inflexible tool for providing reports on package QA status—the underlying R code provide a flexible and extensible toolbox for (some aspects of) package computation.

> "Everything is an object."
>
> "What you see is less than what you get."

E.g., `codoc()` also provides info on usage entries which are not valid R syntax

Run-time tests

- Running all code in the package Rd examples
- Running all code in the package vignettes
- Package-specific tests

Running all code in the package Rd examples

Early principle:

All examples must be executable.

Clear tension between providing "example usage" and comprehensive unit testing.

Currently: by default, examples are both shown and run; selection via `\dontrun{}` and `\dontshow{}`, respectively.

Package-specific tests

If there is a subdir 'tests', then by default

- '.Rin' files are used to create '.R' files
- '.R' files are run through R creating '.Rout' files (or, '.Rout.fail' and exit in case of an error)
- '.Rout.save' files are compared to the corresponding '.Rout' ones

Extremely powerful and substantially underused mechanism!

Repository QA

CRAN operating principles: all packages must “pass” R CMD check relative to the current release version of R at both time of package submission and when a new (major/minor) version of R is released.

Daily check process on CRAN, summary and timings

Summary & Outlook

Contact

Kurt Hornik
Abteilung für Computational Statistics
Institut für Statistik und Mathematik
Wirtschaftsuniversität Wien
Augasse 2–6, A-1090 Wien
Austria

email: Kurt.Hornik@wu-wien.ac.at