

An introduction to the R package mechanism

October 22, 2002

Introduction

The R package mechanism was designed to provide developers with a coherent and manageable way to provide code, data and documentation to users. A *package* should be thought of as a coherent extensible object. Operationally a package is a collection of files and directories (or folders). This document details the procedures and conventions that are used when creating a package.

We will first present some semi-automatic methods for creating an R package and then consider various details and extensions. The simplest R package to create is one that contains only interpreted R functions. We begin by explaining how to perform that task. Adding programs written in other languages is harder and is discussed later.

The simplest approach

In this section we consider creating an R package when only data and interpreted R programs are involved. The process is quite simple. Start R, create the R functions and data objects that you want to include in the R session. Make sure that there are no other objects (use `ls` to check) in your workspace.

Now, executing the command, `package.skeleton("myPkgName")`, will create a package framework. You will need to edit many of the files in the package skeleton in order to get a working package but most of the framework will be created by `package.skeleton`.

In order to be explicit we will consider creating a package that has one function,

```
foo <- function(x) x
```

and one data set,

```
mydata <- 1:10
```

The first thing to do after running `package.skeleton` is to open the `DESCRIPTION` file in the package and edit the entries appropriately.

The R code should be fine and is unlikely to need any further processing. Similarly the data should be fine. The manual pages will in general need some editing. These files are created by calling one of the `prompt` family of functions. The creation of the help files can be customized by editing the appropriate `prompt` function.

The default version of `prompt` indicates lines that should be replaced by the author by starting them with two tildes `~~`. You should identify these lines and replace them with your own text or remove that part of the file if it is not appropriate.

You should create examples for each function that test whether the function is working correctly. These examples will form the basis of your quality control procedures for your package. Careful selection of examples at this point will make future development easier.

R manual pages are currently created using `Rd` format. This is a LaTeX like format and is well described in the `R Extensions Manual`. It is worth examining that document to find out how to properly interact with the documentation format.

You should consider writing a *vignette*. Function documentation should describe explicitly what a function does. Vignettes should provide a task-oriented view of what your package does. Vignettes are written using `Sweave` which is in the R library `tools`. Details on writing vignettes are given in the `R Extensions manual`.

Package Creation Checklist

1. Load all data and R code into your R session.
2. Run, `package.skeleton` to create the form of your package.
3. Edit the `DESCRIPTION` file.
4. Edit the manual pages.
5. Run `R CMD check`. Fix any errors (and any warnings).
6. Run `R CMD build`. Your package is now ready for distribution.

Alternatives

An alternative to using `package.skeleton` is to use the `RPackGen` package from V. Carey <http://biosun1.harvard.edu/~carey/>.

The anatomy of an R package

The current implementation of packages imposes certain conventions on the package author. Some of these are detailed in this section.

A package is a collection of files and directories. One of the most important files is the `DESCRIPTION` file that is found in the top directory of the package. This file has a particular syntax (Debian control format, or DCF) that is described in more detail in an appendix.

There can be a number of directories that further organize the code.

R A directory containing interpreted code.

data A directory containing data objects.

man A directory containing manual pages. There should be one manual page for each object (either data or an R function).

src A directory containing source code, typically in some other language that will be called from R.

inst A directory containing various subdirectories that will be copied when the package is installed.

The `R` and `man` directories can contain subdirectories named `unix`, `windows` and `mac` which will be used only on the appropriate machine.

The following, optional directories, receive special treatment.

demo Scripts that demonstrate the functionality of the package can be placed in this directory. They will be available to and may be run by the `demo` function in R.

inst/doc If this directory is present and contains any vignettes they will be processed by the `R CMD check` process.

tests If present any files containing R code will be run as part of the `R CMD check` procedure.

exec This directory can contain additional executable files (such as shell scripts) that are used by the package.

R Tools for package management

R provides a number of tools to help developers with package management. These include the following tools:

BATCH Run R in batch mode.

COMPILE Compile code.

INSTALL Install the specified package(s).

REMOVE Remove (uninstall) the specified packages.

SHLIB Create a shared library.

They are invoked by R directly. On Unix systems the syntax is either `R CMD toolname` or by `R toolname`. On Windows systems one must use `Rcmd toolname`.

The two tools, `build` and `check`, are perhaps the most important. The first *builds* the package. This is really an assemblage into a format that can be distributed.

The second *checks* the package. The check process is intended to provide developers with essential information regarding any problems with their package. This tool should be run after all modifications to a package.

Another set of tools provides text processing and conversion functionality for the R function documentation system.

Rdconv Convert Rd format to various other formats.

Rd2dvi Convert Rd format to DVI/PDF

Rd2txt Convert Rd format to pretty text

Rdindex Extract index information from Rd files

Sd2Rd Convert S documentation to Rd format

A final selection of tools provides special functionality,

LINK Front-end for creating executable programs

Rprof Post-process R profiling files

config Obtain configuration information about R

Package Maintenance

The package *check* tool provides developers with an invaluable quality assurance tool. The specific details of the check mechanism are described in the R Extension manual.

One of the processes that takes place is to run every example in every manual page in the package. This means that package developers can (and should) place specific functionality checks in to the help pages for their functions.

If there are any *vignettes* in the `inst/doc` directory they will also be run and tested at this time.

Since these checks are run automatically package writers should be careful not to use examples that rely on interactive input. If the developer wants to provide such examples then the example should be enclosed in an `if` statement that checks to see if R is in interactive mode or not.

It is also important to be somewhat realistic in setting up examples. It is not hard to devise examples that will run for hours; this should be avoided if possible.

Appendices

Debian Control Format

In several different places in the package management and creation system files are created with the Debian control format. The syntax is quite straight forward. The files contain tag–value pairs. The tag is considered to be anything from the first position on a line until the first colon `:` is encountered. The value is the entire text from the colon until the end of the line. Any line that starts with white space is considered to be a continuation of the previous line. Text on continuation lines is appended to the value from the previous line. There is no limit to the number of continuation lines.

DESCRIPTION files

The following fields in a DESCRIPTION file are required by the R package mechanism.

Package The name of the package. Should be shortish.

Title A description, not too long, of what the package does.

Version An R package version number. This can be any sequence of integers separated by either dots, `.` or dashes `-`.

Author The author of the package.

Description A longer description of what the package does.

Maintainer The person who maintains the code. You should provide an email address in the form `<yourfault@somewhere.net>`

License The license that your software is distributed under.

The `title` field should be less than one line long. All email addresses should be enclosed in angle brackets, `<` and `>`.

Optional fields include:

Date The creation date.

Depends Which packages this package depends on.

URL A URL that can be used to find out more about the package.

The `depends` field should be a comma separated list of packages (and relations) that this package depends on. A relation is of the form `(>= version number)`. The parentheses are mandatory. There are two comparison operators, either `<=` or `+?=?` can be used. For example, to specify a dependency on R version 1.5.0 or larger you would say:

```
Depends: R (>= 1.5.0)
```

The `depends` field only lists dependencies on R packages. There are other mechanisms available for specifying package dependencies on system utilities or other software libraries such as HDF5.