CHAPTER 19

Making an R Package

One of the strengths of R is the ability to share software as packages. Packages give users a reliable, convenient, and standardized way to access R functions, data, and documentation. Package authors have a means of communicating with users and a way to organize software for the purpose of sharing it with others and reusing it themselves.

In this chapter, you learn the mechanics of making an R package. Although you may not create your own package, you still need to understand how packages work if you need to modify an existing one or if you have problems installing one. There are four good reasons to take the time and effort to learn how to create an R package. The first, and most important, is that it gives you the ability to share your code, data, and documentation with others, including perhaps the R community at large. The second reason is that this will force you to document your code, which will help both you and others, as we have discussed previously. The third is that it will force you to test your code to make sure it works properly, and the fourth is that building a package makes it easier for you to use your own R functions and data.

As of this writing, R has been updated to version 2.15.1, and there are currently 3960 contributed packages at the CRAN repository. There are undoubtedly other packages elsewhere. All of the packages at CRAN, and many of those in other places, are free of charge. Though they are distributed in an “as is” condition, most of the packages contributed to CRAN are quite good. The R Core Team, of course, is vigilant about keeping the language itself and the base version of R as clean and efficient as possible.

The Concept of a Package

The purpose of a package, in general, is to make it easier to share and distribute software. This idea has certainly taken off in the R community; thousands of contributed packages are available for a very wide variety of purposes. An R package is a collection of related source code and other files. When the package is correctly installed by R, you can access the functions and other items in the package by a call to the `library()` function. While the package is under development, it is wise to put all the related files into a single source directory. When you are ready to create the package, you store the source as a compressed file. R provides tools for installing and testing packages. Users of Linux and Mac operating systems will find the R tools sufficient. Windows users, on the other hand, will have to use some additional utilities. I will discuss these, as we have been using R in the Windows environment throughout this book.

We will illustrate the construction of a package with a simple example. Although we will not include code from other languages, you should know that an R package can contain other kinds of software code in addition to R code. If you include code written in other languages, of course, you will also need to have compiler tools for those languages. Remember R is a command-line interpreted language and does not have (or need) a compiler, though there are discussions of an R compiler being developed one day. We will include only R code, documentation, and data in our package. In the process of building our package, we will tie together many loose ends, and you will see how the various strands we have explored separately in this book come together to make a nice (and perhaps even pretty) tapestry.
Let us define several key terms (adapted from the tutorial on creating R packages by R Core Team member Friedrich Leish). The tutorial is available at the CRAN web site.

**Package**: An extension of the R base system with code, data, and documentation in a standardized format

**Library**: A directory containing installed packages

**Repository**: A web site providing packages for installation

**Source**: The original version of a package with text and code

**Binary**: A compiled version of a package

**Base packages**: Part of the R source tree, maintained by the R Core Team

**Recommended packages**: Part of every R installation, but not necessarily maintained by R Core Team

**Contributed packages**: All the rest

Following Leish’s lead, let us create a couple of functions and build a package. In the process, you will learn how to do this on a larger scale when desired.

---

**Note**  The R documentation for creating packages is available at the following URL: [http://cran.r-project.org/doc/manuals/R-exts.html](http://cran.r-project.org/doc/manuals/R-exts.html). The R documentation on creating help files in the Rd format is here: [http://cran.r-project.org/doc/manuals/R-exts.html#Writing-R-documentation-files](http://cran.r-project.org/doc/manuals/R-exts.html#Writing-R-documentation-files).

---

**Some Windows Considerations**

R, which has its roots in Unix, uses *paths* to search for files, so you need to let R know where to find the proper tools. You must edit the path in your Windows environment to include the path to the RTools, to the R binary file, and to the program Rcmd (which is under the R directory). To get to the path in Windows 7, use the following procedure. Right-click on the Computer icon or open the Control Panel and locate the **System Properties** (see Figure 19-1). Click on **Environment Variables**.