R Help Desk — Accessing the Sources

Uwe Ligges - R News 2006-4

Accessing the Sources

by Uwe Ligges

Introduction

One of the major advantages of open source software such as R is implied by its name: the sources are open (accessible) for everybody.

There are many reasons to look at the sources. One example is that a user might want to know exactly what the software does, but the documentation is not sufficiently explicit about the underlying algorithm. As another example, a user might want to change some code in order to implement a modification of a given (already implemented) method or in order to fix a bug in a contributed package or even in R itself.

How to access different kinds of sources (in order to read or to change them), both in R itself and in packages, is described in the following sections.

It is always a good idea to look into appropriate manuals for your current R version, if working on the sources is required. Almost all manuals contain relevant information for this topic: 'An Introduction to R', 'Writing R Extensions', 'R Installation and Administration', and 'The R Language Definition' (Venables et al., 2006; R Development Core Team, 2006a,b,c).

R Code Sources

In most cases, it is sufficient to read some R code of the function in question and look at how other functions are called or how the data are manipulated within a function. The fastest and easiest way to do so for simple functions is to type the function's name into R and let R print the object. For example, here is how to view the source code for the function `matrix()`:

```
> matrix
function (data = NA, nrow = 1, ncol = 1,
          byrow = FALSE, dimnames = NULL)
{
    data <- as.vector(data)
    if (missing(nrow))
        nrow <- ceiling(length(data)/ncol)
    else if (missing(ncol))
        ncol <- ceiling(length(data)/nrow)
    x <- .Internal(matrix(data, nrow, ncol,
                          byrow))
    dimnames(x) <- dimnames
    x
}
<environment: namespace:base>
```

Unfortunately, comments in the code may have been removed from the printed output, because they were already removed in the loaded or installed package in order to save memory. This is *in principle* controlled by the arguments `keep.source` (for R) and keep.source.pkgs (for packages) of the options() function. If these arguments are set to TRUE comments are not removed, but there are also a lot of circumstances, e.g., lazy-loading and saved images of packages (Ripley, 2004), where setting these arguments to TRUE does not help to keep comments in the sources.

Therefore, it makes more sense to look into the original sources, i.e., those from which the package or R have been installed. Both R sources and sources of contributed packages are available from the CRAN mirrors (see http://cran.r-project.org/mirrors.html for a current list of mirrors).

After unpacking a source package, the R source code can be found in the directory PackageName/R/. Since the files therein are plain text files, they can be opened in the user's favorite text editor. For R's base packages, the R code is available in R's subdirectory $R_HOME/src/library/PakageName/R/. For package bundles, replace PackageName by BundleName/PackageName in the paths given above.

Code Hidden in a Namespace

In some cases, a seemingly missing function is called within another function. Such a function might simply be hidden in a namespace (Tierney, 2003). Type getAnywhere("FunctionName") in order to find it. This function reports which namespace a function comes from, and one can then look into the sources of the corresponding package. This is particularly true for S3 methods such as, for example, plot.factor:

```
R> plot.factor
Error: object "plot.factor" not found

R> getAnywhere("plot.factor")
A single object matching 'plot.factor' was found
It was found in the following places
  registered S3 method for plot from namespace
    graphics
  namespace:graphics
with value
### [function code omitted] ###
```

The file that contains the code of plot.factor is '$R_HOME/src/library/graphics/R/plot.R'.

S3 and S4

As another example, suppose that we have the object lmObj, which results from a call to lm(), and we would like to find out what happens when the object is printed. In that case, a new user probably types

```
R> print
```

in order to see the code for printing. The frustrating result of the call is simply:

```
function (x, ...)
UseMethod("print")
<environment: namespace:base>
```

The more experienced user knows a call to UseMethod() indicates that print() is an S3 generic and calls the specific method function that is appropriate for the object of class class(x). It is possible to ask for available methods with methods(print). The function of interest is the S3 method print.lm() from namespace **stats** (the generic itself is in the **base** package namespace). A method hidden in a namespace can be accessed (and therefore printed) directly using the ::: operator as in stats:::print.lm. The file containing the source code for printing the lmObj object is available from package **stats**' sources: '$R_HOME/src/library/stats/R/lm.R'.

In order to understand and change S4 related sources, it is highly advisable to work directly with a package's source files. For a quick look, functions such as getClass(), getGeneric(), and getMethod() are available. The following example prints the code of the show() method for mle objects from the **stats4** package:

```
R> library("stats4")
R> getMethod("show", "mle")
```

Compiled Code Sources

When looking at R source code, sometimes calls to one of the following functions show up: .C(), .Call(), .Fortran(), .External(), or .Internal() and .Primitive(). These functions are calling entry points in compiled code such as shared objects, static libraries or dynamic link libraries. Therefore, it is necessary to look into the sources of the compiled code, if complete understanding of the code is required.

In order to access the sources of compiled code (i.e., C, C++, or Fortran), it is not sufficient to have the binary version of R or a contributed package installed. Rather, it is necessary to download the sources for R or for the package. How to download those sources is described in the previous section.

For contributed packages, source code can be found in the directory PackageName/src/ (again, for package bundles replace PackageName by BundleName/PackageName). Files therein can be searched for the corresponding entry point.

For R and standard R packages, compiled code sources are in subdirectories of $R_HOME/src/. Particularly $R_HOME/src/main/ contains most of the interesting code. The first step is to look up the entry point in file '$R_HOME/src/main/names.c', if the calling R function is either .Primitive() or .Internal(). This is done in the following example for the code implementing the 'simple' R function sum().

Typing sum shows the R code of that function:

```
R> sum
function (..., na.rm = FALSE)
.Internal(sum(..., na.rm = na.rm))
<environment: namespace:base>
```

Obviously, there is only *one* relevant call within sum(), namely the .Internal call to an inner entry point sum(). The next step is to look up that entry point in the file 'names.c', which reveals the following line:

```
/* many lines omitted */
  {"sum", do_summary, 0, 11, -1,
                  {PP_FUNCALL, PREC_FN, 0}},
/* many lines omitted */
```

This line tells us to look for do_summary which itself lives in file 'summary.c' in the same directory. If the filename is not obvious, then it can be found by simply 'grep'ping for the string in R's $R_HOME/src/ path.

The Bleeding Edge

Folks working on the bleeding edge of statistical computing might want to check out the most recent sources, e.g., by looking into the current svn archives of R. To access them via a web browser, visit https://svn.r-project.org/R/. The subdirectory ./trunk/ contains the current R-devel version; other branches (such as R-patched) can be found in ./branches/, and released versions of R can be found in ./tags/.

Summary

It is easy to look at both R and C, C++ or Fortran sources. It is not that difficult to change the sources and to recompile or to reinstall a modified package. This way, users can become developers very easily and contribute bugfixes and new features to existing packages or even to R itself.

Acknowledgments

I would like to thank Roger Bivand, Gabor Grothendieck, Paul Murrell, and David Scott for suggestions to improve this Help Desk article.

Bibliography

R Development Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2006a. URL http://www.R-project.org.

R Development Core Team. *R Installation and Administration*. R Foundation for Statistical Computing, Vienna, Austria, 2006b. URL http://www.R-project.org.

R Development Core Team. *R Language Definition*. R Foundation for Statistical Computing, Vienna, Austria, 2006c. URL http://www.R-project.org.

B. Ripley. Lazy Loading and Packages in R 2.0.0. *R News*, 4(2):2–4, September 2004. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/.

L. Tierney. Name Space Management for R. *R News*, 3(1):2–6, June 2003. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/.

W. N. Venables, D. M. Smith, and the R Development Core Team. *An Introduction to R*. R Foundation for Statistical Computing, Vienna, Austria, 2006. URL http://www.R-project.org.

Uwe Ligges
Fachbereich Statistik, Universität Dortmund, Germany
ligges@statistik.uni-dortmund.de