Building R packages for Windows

Rob J Hyndman 29 June 2008

---

1 Installing the required tools

To build an R package in Windows, you will need to install some additional software tools. These are summarized at

`http://www.murdoch-sutherland.com/Rtools`

1.1 Essential: Rtools

Contains

- Perl.
- Some unix-like tools that can be run from the DOS command prompt.
- MinGW compilers for compiling Fortran and C code.

Download the most recent version from

`http://www.murdoch-sutherland.com/Rtools/`

Run it, choosing the default "Package authoring installation" to build add-on packages.

1.2 Optional: Microsoft HTML Help Workshop

This is used for producing compiled html help files. You can produce an R package without it, but the package will not contain chm files. Download from

`http://go.microsoft.com/fwlink/?LinkId=14188`

Install it in `C:\Program Files\HTML Help Workshop`.

1.3 Optional: MikTeX

MikTeX is used for producing the pdf help files. You can produce an R package without it, but the package will not contain pdf help files. You may have this installed anyway. Download from

`http://www.miktex.org`

1.4 Essential: Setting PATH variable

The PATH variable tells Windows where to find the relevant programs. It may have already been fixed when installing Rtools.

To add a directory to your PATH on Windows XP select

`Control Panel -> System -> Advanced -> Environment Variables`

You should edit your Path variable so that it looks something like this:

```
C:\Rtools\bin;C:\Rtools\perl\bin;C:\Rtools\MinGW\bin;
     C:\Program files\R\R-2.7.0\bin;C:\Program Files\HTML Help Workshop;<others>
```

- The precise directories will depend where you have installed the various tools. The above path should work if you have followed the default installation procedure.
- The htmlhelp part can be omitted if you did not install the HTML help workshop.

If you have not installed HTML Help workshop, you will need to set `WINHELP=NO` in `MkRules` (in the directory `C:\Program files\R\R-2.7.0\src\gnuwin32`).

I have assumed R2.7.0. For later versions, simply change the above paths to the relevant R version. It will probably then still work.

2 Creating the package

Information about creating packages is provided in the document "Writing R extensions" (available under the R Help menu) or at

`http://cran.r-project.org/doc/manuals/R-exts.html`

The main items are summarized below. But you will almost certainly need to consult this document if you are to successfully compile a package.

2.1 Use `package.skeleton`

The simplest way to create a package is to first create a workspace containing all the relevant functions and data sets that you want to include in the package. Delete anything from the workspace that you do not want to include in the package.

Make sure the current directory is set to whereever you want create the package. Use `setwd("C:\My Documents\Rpackages")` for example. Then, to create a package called `"fred"`, use the command

`package.skeleton(name="fred",list=ls()).`

This will generate a directory `fred` and several sub-directories in the required structure.

2.2 Editing the files

A package consists of a directory containing a file 'DESCRIPTION' and usually has the subdirectories 'R', 'data' and 'man'. The package directory should be given the same name as the package. The `package.skeleton` command above will have created these files for you. You now need to edit them so they contain the right information.

2.3 DESCRIPTION file

The DESCRIPTION file contains basic information about the package in the following format:

```
Package: fred
Version: 0.5
Date: 2008-06-29
Title: My first collection of functions
Author: Joe Developer <Joe.Developer@some.domain.net>,
  with contributions from A. User <A.User@whereever.net>.
Maintainer: Joe Developer <Joe.Developer@some.domain.net>
Depends: R (>= 2.2.0), forecast
Suggests: tseries
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.another.url
```

2.4 Rd files

The help files for each function and data set are given in "R documentation" (Rd) files in the `man` subdirectory. These are in a simple markup language closely resembling LaTeX, which can be processed into a variety of formats, including LaTeX, HTML and plain text.

As an example, here is the file which documents the function `seasadj` in the `forecast` package.

```
\name{seasadj}
\alias{seasadj}
\title{Seasonal adjustment}
\usage{
seasadj(object)
}

\arguments{
\item{object}{Object created by \code{\link[stats]{decompose}}
or \code{\link[stats]{stl}}.}
}

\description{Returns seasonally adjusted data constructed
by removing the seasonal component.}

\value{Univariate time series.}

\seealso{\code{\link[stats]{stl}}, \code{\link[stats]{decompose}}}

\author{Rob J Hyndman}

\examples{
plot(AirPassengers)
lines(seasadj(decompose(AirPassengers,"multiplicative")),col=4)
}

\keyword{ts}
```

Detailed instructions for writing R documentation are at

`http://cran.r-project.org/doc/manuals/R-exts.html#Writing-R-documentation-files`

2.5 Including C or Fortran code

If your R code calls C or Fortran functions, the source code for these functions needs to be placed in the subdirectory `src` under `fred`.

2.6 Compiling the package for Windows

To compile the package into a zip file, go to a DOS prompt in the directory containing your package. (i.e., the directory `"C:\My Documents\Rpackages"` in the above example. Then type

```
Rcmd build --binary fred
```

This will compile all the necessary information and create a zip file which should be ready to load in R.

2.7 Checking the package

To check that the package satisfies the requirements for a CRAN package, use

`Rcmd check fred`

The checks are quite strict. A package will often work ok even if it doesn't pass these tests. But it is good practice to build packages that do satisfy these tests as it may save problems later.

2.8 Building a package for other operating systems

To build a package for something other than a Windows computer, use

`Rcmd build fred`

This creates a tar.gz file which can then be installed on a non-Windows computer. It can also be uploaded to CRAN provided it satisfies the above tests.

3 Putting your package on CRAN

1. Run `Rcmd check fred`.
   Packages must pass without warnings to be admitted to the main CRAN package area.
2. Run `Rcmd build fred` to make the tar.gz file.
3. Upload the tar.gz file to `ftp://CRAN.R-project.org/incoming/` using 'anonymous' as log-in name and your e-mail address as password.
4. Send a message to `CRAN@R-project.org` about it.

4 More advanced features

4.1 Package namespaces

- Namespaces allow some functions and other objects to be hidden from the user.
- You specify which objects will be visible and only provide help files for those objects.
  `export(average, nonsense)`
- Save this in a textfile called "NAMESPACE" in the top level package directory.

4.2 S3 methods

```
average <- function(x) {
    ave <- sum(x)/length(x)
    structure(list(ave=ave,x=x),
        class="average")
}

plot.average <- function(object, ...) {
    boxplot(object$x)
    abline(h=object$ave,col=2,lwd=2)
}
```

Usage: `plot(average(rnorm(20)))`

Add to your NAMESPACE file:

```
S3method(plot,average)
```

For new S3 methods, add to an R file:

```
forecast <- function(object,...)
  UseMethod("forecast")
```

4.3 C code

- Put C code in the "src" subdirectory.
- In an R file:
  ```
  f <- function(x)
      .Call("foo", x, PACKAGE="fred")
  ```
- In the "NAMESPACE" file
  ```
  useDynLib(fred)
  export(f)
  ```