

Building R packages for Windows

Rob J Hyndman

29 June 2008



MONASH University

Outline

- 1 **Installing the required tools**
- 2 Creating the package
- 3 Putting your package on CRAN
- 4 More advanced features

Installing the required tools

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

Links and detailed instructions:

www.murdoch-sutherland.com/Rtools

- Rtools (*essential*)
- Microsoft HTML Help Workshop (*optional*)
- MikTeX (*optional*)

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.

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 from:

www.murdoch-sutherland.com/Rtools/

Run latest version of Rtools. Choose default “Package authoring installation”.

Optional: Microsoft HTML Help Workshop

- 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 go.microsoft.com/fwlink/?LinkId=14188

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 already.
- Download from

www.miktex.org

Essential: Setting PATH variable

- The PATH variable tells Windows where to find the relevant programs.
- The path variable 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 check 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>
```

Essential: Setting PATH variable

- Precise directories will depend where you have installed the various tools.

Essential: Setting PATH variable

- Precise directories will depend where you have installed the various tools.
- The `htmlhelp` part of the `PATH` can be omitted if you did not install the HTML help workshop.

Essential: Setting PATH variable

- Precise directories will depend where you have installed the various tools.
- The htmlhelp part of the PATH 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`).

Essential: Setting PATH variable

- Precise directories will depend where you have installed the various tools.
- The `htmlhelp` part of the `PATH` 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`).
- If there are problems, please read the `Rtools.txt` file *carefully*.

Outline

- 1 Installing the required tools
- 2 Creating the package**
- 3 Putting your package on CRAN
- 4 More advanced features

Creating the package

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

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

This document should be read!

Use `package.skeleton()`

Simplest way to create a package

- First create an R workspace containing all relevant functions and data sets for package.

Use `package.skeleton()`

Simplest way to create a package

- First create an R workspace containing all relevant functions and data sets for package.
- Delete anything that you do not want to include in the package.

Use `package.skeleton()`

Simplest way to create a package

- First create an R workspace containing all relevant functions and data sets for package.
- Delete anything that you do not want to include in the package.
- To create a package called `fred`, use R command

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

Use `package.skeleton()`

Simplest way to create a package

- First create an R workspace containing all relevant functions and data sets for package.
- Delete anything that you do not want to include in the package.
- To create a package called `fred`, use R command

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

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

Editing the files

- A package consists of a directory containing a file `DESCRIPTION` and usually has the subdirectories `R`, `data` and `man`.

Editing the files

- A package consists of a directory containing a file `DESCRIPTION` and usually has the subdirectories `R`, `data` and `man`.
- Package directory should be given same name as package.

Editing the files

- A package consists of a directory containing a file `DESCRIPTION` and usually has the subdirectories `R`, `data` and `man`.
- Package directory should be given same name as package.
- The `package.skeleton()` command will have created these files for you.

Editing the files

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

DESCRIPTION file

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>

Rd files

- Help files for each function or data set in the `man` subdirectory under `fred`
- In a simple markup language resembling \LaTeX .
- Detailed instructions for writing R documentation:

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

Rd files

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

\arguments{
\item{object}{Object created by \link[stats]{decompose}
or \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)
}
```

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

`fred/src`

Compiling the package for Windows

- To compile the package into a zip file, go to a DOS prompt in the directory containing your package.

Compiling the package for Windows

- To compile the package into a zip file, go to a DOS prompt in the directory containing your package.
- Then type
`Rcmd build --binary fred`

Compiling the package for Windows

- To compile the package into a zip file, go to a DOS prompt in the directory containing your package.
- 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.

Checking the package

- To check that the package satisfies the requirements for a CRAN package, use `Rcmd check fred`

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.

Building a package for other operating systems

- To build a package for something other than a Windows computer, use
`Rcmd build fred`

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.

Outline

- 1 Installing the required tools
- 2 Creating the package
- 3 Putting your package on CRAN**
- 4 More advanced features

Uploading to CRAN

- 1 Run `Rcmd check fred`.
Packages must pass without warnings to be admitted to the main CRAN package area.

Uploading to 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.

Uploading to 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.

Uploading to 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.

Outline

- 1 Installing the required tools
- 2 Creating the package
- 3 Putting your package on CRAN
- 4 More advanced features**

Package namespaces

- Namespaces allow some functions and other objects to be hidden from the user.

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)
```

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.

S3 methods

Example

```
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)
}
```

S3 methods

Example

```
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)))`

S3 methods

Add to your NAMESPACE file:

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

S3 methods

Add to your NAMESPACE file:

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

For new S3 methods, add to an R file:

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

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)
```

Conclusion

There is a lot more information in

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