Guide to the **SEXPtools** Package

Drew Schmidt

January 30, 2014

Contents

© 2013, Drew Schmidt

Permission is granted to make and distribute verbatim copies of this vignette and its source provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Formatted or processed versions of this manual, if unaccompanied by the source, must acknowledge the copyright and authors of this work.

This manual may be incorrect or out-of-date. The author(s) assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

This publication was typeset using LaTeX.

Acknowledgement

We thank Christian Heckendorf and Wei-Chen Chen for helping with the package's configuration issues, especially in regard to the building of the static library, as well as in helping with portability of the package.

1 Introduction

1.1 Purpose

This package is intended to serve somewhat the same purpose as the very (deservedly!) popular package **Rcpp**. However, this package is not meant to be a competitor to **Rcpp**. Rather, it is meant to fill a very small niche that **Rcpp** does not fill.

1.2 License

The **SEXPtools** package is licensed under the very permissive 2-clause BSD license, commonly referred to as the FreeBSD license. For a copy of the license, see the file named `COPYING` in the root directory of the package source.

1.3 Installation

The package should install without issue from the command line via the usual commands:

Shell Command

```
R CMD INSTALL SEXPtools_0.1-0.tar.gz
```

2 Linking with SEXPtools

2.1 Configuring a Package to use SEXPtools

In your `configure.ac` and/or `src/Makevars` file(s), you can get the package linking and include information via:

```
R_SCMD="${R_HOME}/bin/Rscript -e"

SEXPTOOLS_LDFLAGS=`${R_SCMD} "SEXPtools:::ldflags()"`
SEXPTOOLS_CPPFLAGS=`${R_SCMD} "SEXPtools:::cppflags()"`
```

and adding `$(SEXPTOOLS_LDFLAGS)` to `PKG_LIBS` and `$(SEXPTOOLS_CPPFLAGS)` to `PKG_CPPFLAGS`. See the Writing R Extensions manual for more information. You can also see the **pbdBASE** and **pbdDMAT** packages as examples.

2.2 Testing the Configuration

To ensure that the package configuration is correct, you can use this test code. Include this C file:

sexptools_test.c

```
#include <SEXPtools.h>

SEXP sexptools_test(SEXP mat)
```

```
{
        PRINT(mat);

        return RNULL;
}
```

and this R file:

sexptools_test.r

```
sexptools_test <- function() .Call("sexptools_test", matrix(1:30, 10))
```

Then build your package with the usual `R CMD INSTALL` and test by loading R and running:

```
library(<my package>)

sexptools_test()
```

3 Specification

```
newRlist(x, n);
newRvec(x, n, type);
newRmat(x, m, n, type);
```

4 Example Usage

5 Q&A

This section is a set of frequently asked questions (FAQ), with frequency uniformly equal to zero.

5.1 Why make this?

Probably my biggest motivator was fun; I just wanted to make something like this. Another, more pragmatic reason is that part of my workload (for very non-standard reasons not worth getting into) prevents me from using **Rcpp**. This leaves me stuck with the native C interface for R. And I don't like R's native C interface. This is my attempt to make that interface (slightly) more friendly and convenient to work with.

5.2 Why the strange name?

Every R object (underneath, in the C interface) is an `SEXP` (short for S-expression) object, which is a struct pointer. This is explained in the R Internals manual. This package is a collection of tools for more easily managing SEXP objects.

5.3 Is this now, or will this ever be a competitor to Rcpp?

In terms of features, no. In some other respects yes; see Section 5.4 for details.

5.4 How does this differ from Rcpp?

Each of these packages makes an attempt at solving a serious problem with utilizing compiled code from R: the native interface for C code in R sucks. There are huge differences between the two packages, however. In short, **Rcpp** is *much* a much more comprehensive solution. If you are new to using compiled code with R, frankly this package probably is not for you; you would likely be much better served by **Rcpp**. However, if for some combination of reasons you either cannot or prefer to not use **Rcpp**, then this package may be of interest to you.

Beyond the scope and ease of use of each project (where **Rcpp** handily wins), there are some other critical differences between the projects. A few of note are:

1. **SEXPtools** is more permissively licensed than **Rcpp** (BSD rather than GPL)
2. **SEXPtools** is pure C while **Rcpp** is C++.

These things may not matter in the least to you. If that's the case, then you may well be better served by **Rcpp**.

5.5 Why would I want to use it?

You may well not. But it is an option available to you.

5.6 How would I use SEXPtools in a package?

Assuming that you have some compiled code you have or want to create to use with a package, you simply link with **SEXPtools** and then wrap that compiled code with the utilities provided by **SEXPtools**. For the former, see Section 2, and for the latter, see Section 3 and Section 4.

Philosophically, you should never have the bulk of the work of a function (of any importance) be handled by the R interface (including **SEXPtools**'s version of it). If you do, then your code can never (easily) have a life outside of R. That may sound fine to you now, but if you ever decide to perform some of your work outside of R, then you can't take your compiled code with you. This is just bad practice and shortsightedness.