Writing a package that uses Rcpp

Dirk Eddelbuettel\(^*\) and Romain François\(^*\)

\(^*\)http://dirk.eddelbuettel.com; \(^*\)https://romain.rbind.io/

This document was compiled on January 11, 2022

This document provides a short overview of how to use Rcpp (Eddelbuettel et al., 2022; Eddelbuettel and François, 2011; Eddelbuettel, 2013) when writing an R package. It shows how usage of the function \texttt{Rcpp.package.skeleton} which creates a complete and self-sufficient example package using Rcpp. All components of the directory tree created by \texttt{Rcpp.package.skeleton} are discussed in detail. This document thereby complements the Writing R Extensions manual (R Core Team, 2021) which is the authoritative source on how to extend R in general.

\begin{verbatim}
Rcpp | package | R | C++
\end{verbatim}

1. Introduction

Rcpp (Eddelbuettel et al., 2022; Eddelbuettel and François, 2011; Eddelbuettel, 2013) is an extension package for R which offers an easy-to-use yet featureful interface between C++ and R. However, it is somewhat different from a traditional R package because its key component is a C++ library. A client package that wants to make use of the \texttt{Rcpp} features must link against the library provided by \texttt{Rcpp}.

It should be noted that R has only limited support for C++-level dependencies between packages (R Core Team, 2021). The LinkingTo declaration in the package DESCRIPTION file allows the client package to retrieve the headers of the target package (here \texttt{Rcpp}), but support for linking against a library is not provided by R and has to be added manually.

This document follows the steps of the \texttt{Rcpp.package.skeleton} function to illustrate a recommended way of using \texttt{Rcpp} from a client package. We illustrate this using a simple C++ function which will be called by an R function.

We strongly encourage the reader to become familiar with the material in the Writing R Extensions manual (R Core Team, 2021), as well as with other documents on C++ material in the \texttt{Rcpp.package.skeleton}.

2. Using \texttt{Rcpp.package.skeleton}

2.1. Overview. Rcpp provides a function \texttt{Rcpp.package.skeleton}, modeled after the base R function \texttt{package.skeleton}, which facilitates creation of a skeleton package using \texttt{Rcpp}.

\texttt{Rcpp.package.skeleton} has a number of arguments documented on its help page (and similar to those of \texttt{package.skeleton}). The main argument is the first one which provides the name of the package one aims to create by invoking the function. An illustration of a call using an argument \texttt{mypackage} is provided below.

\begin{verbatim}
Rcpp.package.skeleton("mypackage")
\end{verbatim}

\begin{verbatim}
$ ls -lR mypackage/
DESCRIPTION
\end{verbatim}

Using \texttt{Rcpp.package.skeleton} is by far the simplest approach as it fulfills two roles. It creates the complete set of files needed for a package, and it also includes the different components needed for using \texttt{Rcpp} that we discuss in the following sections.

2.2. C++ code. If the \texttt{attributes} argument is set to TRUE\(^1\), the following C++ file is included in the src/ directory:

\begin{verbatim}
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
List rcpp_hello_world() {

  CharacterVector x = CharacterVector::create("foo", "bar");
  NumericVector y = NumericVector::create( 0.0, 1.0 );
  List z = List::create( x, y );

  return z ;
}
\end{verbatim}

The file defines the simple \texttt{rcpp_hello_world} function that uses a few \texttt{Rcpp} classes and returns a \texttt{List}.

This function is preceded by the \texttt{Rcpp::export} attribute to automatically handle argument conversion because R has to be taught how to e.g. handle the \texttt{List} class.

\texttt{Rcpp.package.skeleton} then invokes \texttt{compileAttributes} on the package, which generates the \texttt{RcppExports.cpp} file (where we indented the first two lines for the more compact display here):

\begin{verbatim}
$ ls -lR mypackage/
DESCRIPTION
\end{verbatim}

\(^1\)Setting \texttt{attributes} to TRUE is the default. This document does not cover the behavior of \texttt{Rcpp.package.skeleton} when \texttt{attributes} is set to FALSE as we try to encourage package developers to use attributes.
The **Imports** declaration indicates R-level dependency between the client package and **Rcpp**; code from the latter is being imported into the package described here. The **LinkingTo** declaration indicates that the client package needs to use header files exposed by **Rcpp**.

### 2.5. Now optional: Makevars and Makevars.win

This behaviour changed with **Rcpp** release 0.11.0. These files used to be mandatory, now they are merely optional.

We will describe the old setting first as it was in use for a few years. The new standard, however, is much easier and is described below.

#### 2.6. Releases up until 0.10.6

Unfortunately, the **LinkingTo** declaration in itself was not enough to link to the user C++ library of **Rcpp**. Until more explicit support for libraries is added to R, ones needs to manually add the **Rcpp** library to the **PKG_LIBS** variable in the Makevars and Makevars.win files. (This has now changed with release 0.11.0; see below.) **Rcpp** provides the unexported function **Rcpp:::LdFlags()** to ease the process:

```r
# Use the R_HOME indirection to support installations of multiple R version
#
# NB: No longer needed, see below
PKG_LIBS = "$(R_HOME)/bin/Rscript -e "Rcpp:::LdFlags()"
```

The **Makevars.win** is the equivalent, targeting windows.

```r
# Use the R_HOME indirection to support installations of multiple R version
#
# NB: No longer needed, see below
PKG_LIBS = $(shell "$(R_HOME)/bin/Rscript.exe" -e "Rcpp:::LdFlags()")
```

#### 2.7. Releases since 0.11.0

As of release 0.11.0, this is no longer needed as client packages obtain the required code from **Rcpp** via explicit function registration. The user does not have to do anything.

This means that **PKG_LIBS** can now be empty—unless some client libraries are needed. For example, **RcppCNPy** needs compression support and hence uses **PKG_LIBS** = -lz. Similarly, when a third-party library is required, it can and should be set here.

### 2.8. NAMESPACE

The **Rcpp.package.skeleton** function also creates a file **NAMESPACE**.

```r
useDynLib(mypackage)
exportPattern(~[[alpha:]][+])
importFrom(Rcpp, evalCpp)
```

This file serves three purposes. First, it ensure that the dynamic library contained in the package we are creating via **Rcpp.package.skeleton** will be loaded and thereby made available to the newly created R package.

Second, it declares which functions should be globally visible from the namespace of this package. As a reasonable default, we export all functions.

Third, it instructs R to import a symbol from **Rcpp**. This sets up the import of all registered function and, together with the
Imports: statement in DESCRIPTION, provides what is needed for client packages to access Rcpp functionality.

2.9. Help files. Also created is a directory man containing two help files. One is for the package itself, the other for the (single) R function being provided and exported.

The Writing R Extensions manual (R Core Team, 2021) provides the complete documentation on how to create suitable content for help files.

2.10. mypackage-package.Rd. The help file mypackage-package.Rd can be used to describe the new package (and we once again indented some lines):

```r
\name{mypackage-package}
\alias{mypackage-package}
\alias{mypackage}
\docType{package}
\title{What the package does (short line)}
\description{More about what it does (maybe more than one line) -- A concise (1-5 lines) description of the package --}
\details{\tabular{ll}
Package: \tab mypackage\cr
Type: \tab Package\cr
Version: \tab 1.0\cr
Date: \tab 2013-09-17\cr
License: \tab What license is it under?\cr
\tab -- An overview of how to use the package, including the most important functions --}
\author{Who wrote it}
Maintainer: Who <yourfault@somewhere.net>
\references{
-- Literature or other references for background information --}
\references{
-- Optionally other standard keywords, one per line, from file KEYWORDS in the R documentation directory --
\keyword{package}
\seealso{
-- Optional links to other man pages, e.g. --
-- \code{\link[[\pkg::<pkg>-package]}} --}
\examples{
\dontrun{
rcpp_hello_world()}
examples{
rcpp_hello_world()}}
```

3. Using modules

This document does not cover the use of the module argument of Rcpp.package.skeleton. It is covered in the modules vignette (Eddelbuettel and François, 2022).

4. Further examples

The canonical example of a package that uses Rcpp is the RcppExamples (Eddelbuettel and François, 2019) package. RcppExamples contains various examples of using Rcpp. Hence, the RcppExamples package is provided as a template for employing Rcpp in packages.

Other CRAN packages using the Rcpp package are RcppArmadillo (Eddelbuettel et al., 2021), and minqa (Bates et al., 2014). Several other packages follow older (but still supported and appropriate) instructions. They can serve examples on how to get data to and from C++ routines, but should not be considered templates for how to connect to Rcpp. The full list of packages using Rcpp can be found at the CRAN page of Rcpp.

5. Other compilers

Less experienced R users on the Windows platform frequently ask about using Rcpp with the Visual Studio toolchain. That is simply not possible as R is built with the gcc compiler. Different compilers have different linking conventions. These conventions are particularly hairy when it comes to using C++. In short, it is not possible to simply drop sources (or header files) from Rcpp into a C++ project built with Visual Studio, and this note makes no attempt at claiming otherwise.

Rcpp is fully usable on Windows provided the standard Windows toolchain for R is used. See the Writing R Extensions manual (R Core Team, 2021) for details.

6. Summary

This document described how to use the Rcpp package for R and C++ integration when writing an R extension package. The use of the Rcpp_package.skeleton was shown in detail, and references to further examples were provided.

References


