Frequently Asked Questions about **Rcpp**

Dirk Eddelbuettel Romain François

Rcpp version 0.9.10 as of February 16, 2012

Abstract

This document attempts to answer the most Frequently Asked Questions (FAQ) regarding the **Rcpp** (Eddelbuettel and François, 2011a,b) package.

1 Getting started

1.1 How do I get started ?

Please execute the following command in R

```
> vignette( "Rcpp-introduction" )
```

to access the vignette which provides a detailed introduction.

1.2 What do I need ?

Obviously, R must be installed. **Rcpp** provides a C++ API as an extension to the R system. As such, it is bound by the choices made by R and is also influenced by how R is configured.

In general, the standard environment for building a CRAN package from source (particularly when it contains C or C++ code) is required. This means one needs:

- a development environment with a suitable compiler (see below), header files and required libraries;
- R should be built in a way that permits linking and possibly embedding of R; this is typically ensured by the `-enable-shared-lib` option;
- standard development tools such as `make` etc.

1.3 What compiler can I use ?

On almost all platforms, the GNU Compiler Collection (or `gcc`, which is also the name of its C language compiler) has to be used along with the corresponding `g++` compiler for the C++ language. A minimal suitable version is a final 4.2.* release; earlier 4.2.* were lacking some C++ features. Generally speaking, and as of early 2011, the default compilers on all the common platforms are suitable.

Specific per-platform notes:

Windows users need the `Rtools` package from the site maintained by Duncan Murdoch which contains all the required tools in a single package; complete instructions specific to Windows are in the 'R Administration' manual (R Development Core Team, 2011a, Appendix D).

OS X users, as noted in the 'R Administration' manual (R Development Core Team, 2011a, Appendix C.4), need to install the Apple Developer Tools (*e.g.*, `Xcode`) (as well as `gfortran` if R or Fortran-using packages are to be built); also see FAQ 2.8 below.

Linux user need to install the standard developement packages. Some distributions provide helper packages which pull in all the required packages; the `r-base-dev` package on Debian and Ubuntu is an example.

1.4 What other packages are useful ?

Additional packages that we have found useful are

inline which is invaluable for direct compilation, linking and loading of short code snippets;

RUnit is used for unit testing; the package is recommended and will be needed to re-run some of our tests but it is not strictly required;

rbenchmark to run simple timing comparisons and benchmarks; it is also recommended but not required.

2 Compiling and Linking

2.1 How do I use Rcpp in my package ?

Rcpp has been specifically designed to be used by other packages. Making a package that uses **Rcpp** depends on the same mechanics that are involved in making any R package that use compiled code — so reading the *Writing R Extensions* manual (R Development Core Team, 2011b) is a required first step.

Further steps, specific to **Rcpp**, are described in a separate vignette.

```
> vignette( "Rcpp-package" )
```

2.2 How do I quickly prototype my code ?

The **inline** package (Sklyar, Murdoch, Smith, Eddelbuettel, and François, 2010) provides the functions `cfunction` and `cxxfunction`. Below is a simple function that uses `accumulate` from the (C++) Standard Template Library to sum the elements of a numeric vector.

```
> fx <- cxxfunction(signature( x = "numeric" ),
+     'NumericVector xx(x);
     return wrap( std::accumulate( xx.begin(), xx.end(), 0.0));',
+     plugin = "Rcpp")
> res <- fx( seq( 1, 10, by = 0.5 ) )
> res
[1] 104.5
```

Rcpp uses **inline** to power its entire unit test suite. Consult the `unitTests` directory of **Rcpp** for over five hundred further examples.

```
> list.files( system.file( "unitTests", package = "Rcpp" ), pattern = "^runit[.]" )
```

One might want to use code that lives in a C++ file instead of writing the code in a character string in R. This is easily achieved by using `readLines` :

```
> fx <- cxxfunction( signature(),
+         paste( readLines( "myfile.cpp"), collapse = "\n" ),
+         plugin = "Rcpp" )
```

The `verbose` argument of `cxxfunction` is very useful as it shows how **inline** runs the show.

2.3 How do I convert my prototyped code to a package ?

Since release 0.3.5 of **inline**, one can combine FAQ 2.2 and FAQ 2.1. See `help("package.skeleton-methods")` once **inline** is loaded and use the skeleton-generating functionality to transform a prototyped function into the minimal structure of a package. After that you can proceed with working on the package in the spirit of FAQ 2.1.

2.4 But I want to compile my code with R CMD SHLIB !

The recommended way is to create a package and follow FAQ 2.1. The alternate recommendation is to use **inline** and follow FAQ 2.2 because it takes care of all the details.

However, some people have shown that they prefer not to follow recommended guidelines and compile their code using the traditional `R CMD SHLIB`. To do this, we need to help `SHLIB` and let it know about the header files that **Rcpp** provides and the C++ library the code must link against.

```
$ export PKG_LIBS=`Rscript -e "Rcpp:::LdFlags()"`
$ export PKG_CXXFLAGS=`Rscript -e "Rcpp:::CxxFlags()"`
$ R CMD SHLIB myfile.cpp
```

This approach corresponds to the very earliest ways of building programs and can still be found in some deprecated documents (as *e.g.* some of Dirk's older 'Intro to HPC with R' tutorial slides). It is still not recommended as there are tools and automation mechanisms that can do the work for you.

An alternative, which might work better on Windows is to use the unexported function `Rcpp:::SHLIB` :

```
$ Rscript -e "Rcpp:::SHLIB('myfile.cpp')"
```

2.5 What about `LinkingTo` ?

R has only limited support for cross-package linkage.

We now employ the `LinkingTo` field of the `DESCRIPTION` file of packages using **Rcpp**. But this only helps in having R compute the location of the header files for us.

The actual library location and argument still needs to be provided by the user. How to do so has been shown above, and we recommned you use either FAQ 2.1 or FAQ 2.2 both which use the **Rcpp** function `Rcpp:::LdFlags()`.

If and when `LinkingTo` changes and lives up to its name, we will be sure to adapt **Rcpp** as well.

2.6 Does Rcpp work on windows ?

Yes of course. See the Windows binaries provided by CRAN.

2.7 Can I use Rcpp with Visual Studio ?

Not a chance.

And that is not because we are meanies but because R and Visual Studio simply do not get along. As **Rcpp** is all about extending R with C++ interfaces, we are bound by the available toolchain. And R simply does not compile with Visual Studio. Go complain to its vendor if you are still upset.

2.8 I am having problems building Rcpp on OS X, any help ?

OS X is a little more conservative with compiler versions, so it pays to get the latest of whatever Apple releases which may already be a little behind what is used on Linux or Windows.

At the time of writing this paragraph (in the spring of 2011), **Rcpp** (just like CRAN) supports all OS X releases greater or equal to 10.5. However, building **Rcpp** from source (or building packages using **Rcpp**) also requires a recent-enough version of Xcode. For the *Leopard* release of OS X, the current version is 3.1.4 which can be downloaded free of charge from the Apple Developer site. Users may have to manually select `g++-4.2` via the symbolic link `/usr/bin/g++`. The *Snow Leopard* release already comes with Xcode 3.2.x and work as is.

2.9 Does Rcpp work on solaris/suncc ?

Yes.

2.10 Does Rcpp work with Revolution R ?

We have not tested it yet. **Rcpp** might need a few tweaks to work with the compilers used by Revolution R.

2.11 Is it related to CXXR ?

CXXR is an ambitious project that aims to totally refactor the R interpreter in C++. There are a few similaritites with **Rcpp** but the projects are unrelated.

CXXR and **Rcpp** both want R to make more use of C++ but they do it in very different ways.

3 Examples

The following questions were asked on the `rcpp-devel` mailing list, which is generally the best place to ask questions.

3.1 Can I use templates with Rcpp and inline ?

> *I'm curious whether one can provide a class definition inline in an R script and then initialize an instance of the class and call a method on the class, all inline in R.*

Most certainly, consider this simple example of a templated class which squares its argument:

```
inc <- 'template <typename T>
        class square : public std::unary_function<T,T> {
        public:
            T operator()( T t) const { return t*t ;}
        };
       '

src <- '
       double x = Rcpp::as<double>(xs);
       int i = Rcpp::as<int>(is);
       square<double> sqdbl;
       square<int> sqint;
       return Rcpp::DataFrame::create(Rcpp::Named("x", sqdbl(x)),
                                      Rcpp::Named("i", sqint(i)));
       '
fun <- cxxfunction(signature(xs="numeric", is="integer"),
                   body=src, include=inc, plugin="Rcpp")

fun(2.2, 3L)
```

3.2 Can I do matrix algebra with Rcpp ?

> ***Rcpp*** *allows element-wise operations on vector and matrices through operator overloading and STL interface, but what if I want to multiply a matrix by a vector, etc ...*

Currently, **Rcpp** does not provide binary operators to allow operations involving entire objects. Adding operators to **Rcpp** would be a major project (if done right) involving advanced techniques such as expression templates. We currently do not plan to go in this direction, but we would welcome external help. Please send us a design document.

However, we have developed the **RcppArmadillo** package (François, Eddelbuettel, and Bates, 2011) that provides a bridge between **Rcpp** and **Armadillo** (Sanderson, 2010). **Armadillo** supports binary operators on its types in a way that takes full advantage of expression templates to remove temporaries and allow chaining of operations. That is a mouthful of words meaning that it makes the code go faster by using fiendishly clever ways available via the so-called template meta programming, an advanced C++ technique. Also, the **RcppEigen** package provides an alternative using the http://eigen.tuxfamily.org template library.

The following example is adapted from the examples available at the project page of Armadillo. It calculates $x' \times Y^{-1} \times z$

```
// copy the data to armadillo structures
arma::colvec x = Rcpp::as<arma::colvec> (x_);
arma::mat Y = Rcpp::as<arma::mat>( Y_ ) ;
arma::colvec z = Rcpp::as<arma::colvec>( z_ ) ;

// calculate the result
double result = arma::as_scalar(
    arma::trans(x) * arma::inv(Y) * z
    );

// return it to R
return Rcpp::wrap( result );
```

```
> fx <- cxxfunction(
+          signature(x_ = "numeric", Y_ = "matrix", z_ = "numeric" ),
+          paste( readLines( "myfile.cpp" ), collapse = "\n" ),
+          plugin = "RcppArmadillo" )
> fx( 1:4, diag( 4 ), 1:4 )
[1] 30
```

The focus is on the code `arma::trans(x) * arma::inv(Y) * z`, which performs the same operation as the R code `t(x) %*% solve(Y) %*% z`, although Armadillo turns it into only one operation, which makes it quite fast. Armadillo benchmarks against other C++ matrix algebra libraries are provided on the Armadillo website.

It should be noted that code below depends on the version `0.3.5` of **inline** and the version `0.2.2` of **RcppArmadillo**

3.3 Can I use code from the Rmath header and library with Rcpp ?

Can I call functions defined in the Rmath header file and the standalone math library for R–as for example the random number generators?

Yes, of course. This math library exports a subset of R, but **Rcpp** has access to much more. Here is another simple example. Note how we have to use and instance of the `RNGScope` class to set and re-set the random-number generator. This also illustrates Rcpp sugar as we are using a vectorised call to `rnorm`. Moreover, because the RNG is reset, the two calls result in the same random draws. If we wanted to control the draws, we could explicitly set the seed after the `RNGScope` object has been instantiated.

```
> fx <- cxxfunction(signature(),
+                   'RNGScope();
                   return rnorm(5, 0, 100);',
+                    plugin="Rcpp")
> fx()
[1] 144.26077 -44.57712 168.34426 -30.34574 -90.67051
> fx()
[1] 144.26077 -44.57712 168.34426 -30.34574 -90.67051
```

3.4 Can I use NA and Inf with Rcpp ?

R knows about NA and Inf. How do I use them from C++?

Yes, see the following example:

```
> src <- 'Rcpp::NumericVector v(4);
        v[0] = R_NegInf;  // -Inf
        v[1] = NA_REAL;   // NA
        v[2] = R_PosInf;  // Inf
        v[3] = 42;        // see the Hitchhiker Guide
        return Rcpp::wrap(v);'
> fun <- cxxfunction(signature(), src, plugin="Rcpp")
> fun()
[1] -Inf   NA  Inf   42
```

3.5 Can I easily multiply matrices ?

Can I multiply matrices easily?

Yes, via the **RcppArmadillo** package which builds upon **Rcpp** and the wonderful Armadillo library at `http://arma.sf.net`:

```
> txt <- 'arma::mat Am = Rcpp::as< arma::mat >(A);
        arma::mat Bm = Rcpp::as< arma::mat >(B);
        return Rcpp::wrap( Am * Bm );'
> mmult <- cxxfunction(signature(A="numeric", B="numeric"),
+                      body=txt, plugin="RcppArmadillo")
> A <- matrix(1:9, 3, 3)
> B <- matrix(9:1, 3, 3)
> C <- mmult(A, B)
```

Armadillo supports a full range of common linear algebra operations.
The **RcppEigen** package provides an alternative using the http://eigen.tuxfamily.org template library.

3.6 How do I write a plugin for inline ?

*How can I create my own plugin for use by the **inline** package?*

Here is an example which shows how to it using GSL libraries as an example. This is merely for demonstration, it is also not perfectly general as we do not detect locations first—but it serves as an example:

```
> ## simple example of seeding RNG and drawing one random number
> gslrng <- '
int seed = Rcpp::as<int>(par) ;
gsl_rng_env_setup();
gsl_rng *r = gsl_rng_alloc (gsl_rng_default);
gsl_rng_set (r, (unsigned long) seed);
double v = gsl_rng_get (r);
gsl_rng_free(r);
return Rcpp::wrap(v);'
> plug <- Rcpp:::Rcpp.plugin.maker(
+     include.before = "#include <gsl/gsl_rng.h>",
+     libs = paste("-L/usr/local/lib/R/site-library/Rcpp/lib -lRcpp",
+                  "-Wl,-rpath,/usr/local/lib/R/site-library/Rcpp/lib",
+                  "-L/usr/lib -lgsl -lgslcblas -lm"))
> registerPlugin("gslDemo", plug )
> fun <- cxxfunction(signature(par="numeric"), gslrng, plugin="gslDemo")
> fun(0)
```

Here the **Rcpp** function `Rcpp.plugin.maker` is used to create a plugin 'plug' which is then registered, and subsequently used by **inline**.

3.7 How can I pass one additional flag to the compiler?

How can I pass another flag to the `g++` compiler without writing a new plugin?

The quickest way is to modify the return value from an existing plugin. Here we use the default one from **Rcpp** itself in order to pass the new flag `-std=c++0x`. As it does not set the `PKG_CXXFLAGS` variable, we simply assign this. For other plugins, one may need to append to the existing values instead.

```
> myplugin <- getPlugin("Rcpp")
> myplugin$env$PKG_CXXFLAGS <- "-std=c++0x"
> f <- cxxfunction(signature(), settings=myplugin, body='
+    std::vector<double> x = { 1.0, 2.0, 3.0 };  // fails without -std=c++0x
+    return Rcpp::wrap(x);
+ ')
> f()
```

3.8 How can I set matrix row and column names ?

Ok, I can create a matrix, but how do I set its row and columns names?

Pretty much the same way as in R itself: We define a list with two character vectors, one each for row and column names, and assign this to the `dimnames` attribute:

```
> src <- '
  Rcpp::NumericMatrix x(2,2);
  x.fill(42);                           // or more interesting values
  Rcpp::List dimnms =                   // two vec. with static names
      Rcpp::List::create(Rcpp::CharacterVector::create("cc", "dd"),
                         Rcpp::CharacterVector::create("ee", "ff"));
  // and assign it
  x.attr("dimnames") = dimnms;
  return(x);
'
> fun <- cxxfunction(signature(), body=src, plugin="Rcpp")
> fun()
```

3.9 Why can long long types not be cast correctly?

That is a good and open question. We rely on the basic R types, notably `integer` and `numeric`. These can be cast to and from C++ types without problems. But there are corner cases. The following example, contributed by a user, shows that we cannot reliably cast `long` types (on a 64-bit machines).

```
> BigInts <- cxxfunction(signature(),
+   'std::vector<long> bigints;
   bigints.push_back(12345678901234567LL);
   bigints.push_back(12345678901234568LL);
   Rprintf("Difference of %ld\\n", 12345678901234568LL - 12345678901234567LL);
   return wrap(bigints);', plugin="Rcpp", includes="#include <vector>")
> retval<-BigInts()
> # Unique 64-bit integers were cast to identical lower precision numerics
> # behind my back with no warnings or errors whatsoever.  Error.
>
> stopifnot(length(unique(retval)) == 2)
```

While the difference of one is evident at the C++ level, it is no longer present once cast to R. The 64-bit integer values get cast to a floating point types with a 53-bit mantissa. We do not have a good suggestion or fix for casting 64-bit integer values: 32-bit integer values fit into `integer` types, up to 53 bit precision fits into `numeric` and beyond that truly large integers may have to converted (rather crudely) to text and re-parsed. Using a different representation as for example from the GNU Multiple Precision Arithmetic Library may be an alternative.

4 Support

4.1 Is the API documented ?

You bet. We use doxygen to generate html, latex and man page documentation from the source. The html documentation is available for browsing, as a very large pdf file, and all three formats are also available a zip-archives: html, latex, and man.

4.2 Does it really work ?

We take quality seriously and have developped an extensive unit test suite to cover many possible uses of the **Rcpp** API.

We are always on the look for more coverage in our testing. Please let us know if something has not been tested enough.

4.3 Where can I ask further questions ?

The Rcpp-devel mailing list hosted at R-forge is by far the best place. You may also want to look at the list archives to see if your question has been asked before.

4.4 Where can I read old questions and answers ?

The normal Rcpp-devel mailing list hosting at R-forge contains an archive, which can be searched via swish.

Alternatively, one can also use Gmane on Rcpp-devel as well as Mail-Archive on Rcpp-devel both of which offer web-based interfaces, including searching.

4.5 I like it. How can I help ?

The current list of things to do is available in our `TODO` file. . If you are willing to donate time and have skills in C++, let us know. If you are willing to donate money to sponsor improvements, let us know.

You can also spread the word about **Rcpp**. There are many packages on CRAN that use C++, yet are not using **Rcpp**. You could write a review of **Rcpp** in crantastic, blog about it or get the word out otherwise.

4.6 I don't like it. How can I help ?

It is very generous of you to still want to help. Perhaps you can tell us what it is that you dislike. We are very open to *constructive* criticism.

4.7 Can I have commercial support for Rcpp ?

Sure you can. Just send us an email, and we will be happy to discuss the request..

4.8 I want to learn quickly. Do you provide training courses ?

Yes. Just send us an email.

References

Dirk Eddelbuettel and Romain François. *Rcpp: Seamless R and C++ Integration*, 2011a. URL `http://CRAN.R-Project.org/package=Rcpp`. R package version 0.9.9.

Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011b. URL `http://www.jstatsoft.org/v40/i08/`.

Romain François, Dirk Eddelbuettel, and Douglas Bates. *RcppArmadillo: Rcpp integration for Armadillo templated linear algebra library*, 2011. URL `http://CRAN.R-Project.org/package=RcppArmadillo`. R package version 0.2.34.

R Development Core Team. *R Installation and Administration*. R Foundation for Statistical Computing, Vienna, Austria, 2011a. URL `http://CRAN.R-Project.org/doc/manuals/R-admin.html`. ISBN 3-900051-09-7.

R Development Core Team. *Writing R extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2011b. URL http://CRAN.R-Project.org/doc/manuals/R-exts.html. ISBN 3-900051-11-9.

Conrad Sanderson. Armadillo: An open source C++ algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010. URL http://arma.sf.net.

Oleg Sklyar, Duncan Murdoch, Mike Smith, Dirk Eddelbuettel, and Romain François. *inline: Inline C, C++, Fortran function calls from R*, 2010. URL http://CRAN.R-Project.org/package=inline. R package version 0.3.8.