Frequently Asked Questions about **Rcpp**

Dirk Eddelbuettel Romain François

Rcpp version 0.9.0 as of December 19, 2010

Abstract

This document attempts to answer the most Frequently Asked Questions (FAQ) regarding the **Rcpp** (Eddelbuettel and François, 2010) package.

1 Getting started

1.1 How do I get started ?

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

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, 2010) 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 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)
```

2.7 Does Rcpp work on windows

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

2.8 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.

2.9 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 API

3.1 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, 2010) 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.

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

```
    // 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**

3.2 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.

3.3 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 Support

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.5 Can I have commercial support for Rcpp ?

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

4.6 I want to learn quickly. Do you provide training courses.

Yes. Just send us an email.

References

Dirk Eddelbuettel and Romain François. *Rcpp R/C++ interface package*, 2010. URL `http://CRAN.R-Project.org//package=Rcpp`. R package version 0.8.10.

Romain François, Dirk Eddelbuettel, and Douglas Bates. *RcppArmadillo: Rcpp integration for Armadillo templated linear algebra library*, 2010. URL `http://CRAN.R-Project.org//package=RcppArmadillo`. R package version 0.2.7.

R Development Core Team. *Writing R extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2010. URL `http://CRAN.R-Project.org/doc/manuals/R-exts.html`.

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 0.3.6.