Use R!

Dirk Eddelbuettel

Seamless R and C++ Integration with Rcpp

Springer

Use R!

Series Editors:
Robert Gentleman Kurt Hornik Giovanni G. Parmigiani

For further volumes:
http://www.springer.com/series/6991

Dirk Eddelbuettel

Seamless R and C++ Integration with Rcpp

Springer

Dirk Eddelbuettel
River Forest
Illinois, USA

ISBN 978-1-4614-6867-7 ISBN 978-1-4614-6868-4 (eBook)
DOI 10.1007/978-1-4614-6868-4
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013933242

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Lisa, Anna and Julia

Preface

Rcpp is an R add-on package which facilitates extending R with C++ functions.

It is being used for anything from small and quickly constructed add-on functions written either to fluidly experiment with something new or to accelerate computing by replacing an R function with its C++ equivalent to large-scale bindings for existing libraries, or as a building block in entirely new research computing environments.

While still relatively new as a project, **Rcpp** has already become widely deployed among users and developers in the R community. **Rcpp** is now the most popular language extension for the R system and used by over 100 CRAN packages as well as ten BioConductor packages.

This books aims to provide a solid introduction to **Rcpp**.

Target Audience

This book is for R users who would like to extend R with C++ code. Some familiarity with R is certainly helpful; a number of other books can provide refreshers or specific introductions. C++ knowledge is also helpful, though not strictly required. An appendix provides a very brief introduction for C++ to those familiar only with the R language.

The book should also be helpful to those coming to R with more of a C++ programming background. However, additional background reading may be required to obtain a firmer grounding in R itself. Chambers (2008) is a good introduction to the philosophy behind the R system and a helpful source in order to acquire a deeper understanding.

There may also be some readers who would like to see how **Rcpp** works internally. Covering that aspect, however, requires a fairly substantial C++ content and is not what this book is trying to provide. The focus of this book is clearly on how to *use* **Rcpp**.

Historical Context

Rcpp first appeared in 2005 as a (fairly small when compared to its current size) contribution by Dominick Samperi to the **RQuantLib** package started by Eddelbuettel in 2002 (Eddelbuettel and Nguyen 2012). **Rcpp** became a CRAN package in its own name in early 2006. Several releases (all provided by Samperi) followed in quick succession under the name **Rcpp**. The package was then renamed to **RcppTemplate**; several more releases followed during 2006 under the new name. However, no new releases were made during 2007, 2008, or most of 2009. Following a few updates in late 2009, the **RcppTemplate** package has since been archived on CRAN for lack of active maintenance.

Given the continued use of the package, Eddelbuettel decided to revitalize it. New releases, using the original name **Rcpp**, started in November 2008. These included an improved build and distribution process, additional documentation, and new functionality—while retaining the existing "classic **Rcpp**" interface. While not described here, this API will continue to be provided and supported via the **RcppClassic** package (Eddelbuettel and François 2012c).

Reflecting evolving C++ coding standards (see Meyers 2005), Eddelbuettel and François started a significant redesign of the code base in 2009. This added numerous new features, many of which are described in the package via different vignettes. This redesigned version of **Rcpp** (Eddelbuettel and François 2012a) has become widely used with over ninety CRAN packages depending on it as of November 2012. It is also the version described in this book.

Rcpp continues to be under active development, and extensions are being added. The content described here shall remain valid and supported.

Related Work

Integration of C++ and R has been addressed by several authors; the earliest published reference is probably Bates and DebRoy (2001). The "Writing R Extensions" manual (R Development Core Team 2012d) has also been mentioning C++ and R integration since around that time. An unpublished paper by Java et al. (2007) expresses several ideas that are close to some of our approaches, though not yet fully fleshed out. The **Rserve** package (Urbanek 2003, 2012) acts as a socket server for R. On the server side, **Rserve** translates R data structures into a binary serialization format and uses TCP/IP for transfer. On the client side, objects are reconstructed as instances of Java or C++ classes that emulate the structure of R objects.

The packages **rcppbind** (Liang 2008), **RAbstraction** (Armstrong 2009a), and **RObjects** (Armstrong 2009b) are all implemented using C++ templates. None of them have matured to the point of a CRAN release. **CXXR** (Runnalls 2009) approaches this topic from the other direction: its aim is to completely refactor R on a stronger C++ foundation. **CXXR** is therefore concerned with all aspects of the R interpreter, read-eval-print loop (REPL), and threading; object interchange be-

tween R and C++ is but one part. A similar approach is discussed by Temple Lang (2009a) who suggests making low-level internals extensible by package developers in order to facilitate extending R. Temple Lang (2009b), using compiler output for references on the code in order to add bindings and wrappers, offers a slightly different angle. Lastly, the **rdyncall** package (Adler 2012) provides a direct interface from R into C language APIs. This can be of interest if R programmers want to access lower-level programming interfaces directly. However, it does not aim for the same object-level interchange that is possible via C++ interfaces, and which we focus on with **Rcpp**.

Typographic Convention

The typesetting follows the usage exemplified both by the publisher, and by the *Journal of Statistical Software*. We use

- Sans-serif for programming language such as R or C++
- Boldface for (CRAN or other) software packages such as **Rcpp** or **inline**
- Courier for short segments of code or variables such as `x <- y + z`

We make use of a specific environment for the short pieces of source code interwoven with the main text.

River Forest, IL, USA Dirk Eddelbuettel

Acknowledgements

Rcpp is the work of many contributors, and a few words of thanks are in order.

Dominick Samperi contributed the original code which, while much more limited in scope than the current **Rcpp**, pointed clearly in the right direction of using C++ templates to convert between R and C++ types.

Romain François has shown impeccable taste in designing and implementing very large parts of **Rcpp** as it is today. The power of the current design owes a lot to his work and boundless energy. Key components such as modules and sugar, as well as lot a of template "magic," are his contributions. This started as an aside to make object interchange easier for our **RProtoBuf** package—and it has taken us down a completely different, but very exciting road. It has been a pleasure to work with Romain, I remain in awe of his work, and I look forward to many more advances with **Rcpp**.

Doug Bates has been a help from the very beginning: had it not been for some simple macros to pick list components out of `SEXP` types, I may never have started **RQuantLib** a decade ago. Doug later joined this project and has been instrumental in a few key decisions regarding **Rcpp** and **RcppArmadillo** and has taken charge of the **RcppEigen** project.

John Chambers become a key supporter right when *Rcpp modules* started and contributed several important pieces at the gory intersection between R and C++. It was very flattering for Romain and me to hear from John how **Rcpp** is so close to an original design vision of a whole-object interchange between systems which was already present on a hand-drawn Bell Labs designs from the 1970s.

JJ Allaire has become a very important contributor to **Rcpp** and a key supporter of the same idea of an almost natural pairing between R and C++. The *Rcpp attributes* which he contributed are showing a lot of promise, and we expect great things to be built on top of this.

Several other members of the R Core team—notably Kurt Hornik, Uwe Ligges, Martyn Plummer, Brian Ripley, Luke Tierney, and Simon Urbanek—have helped at various points with anything from build issues and portability to finer points of R internals. Last but not least, there would of course be no **Rcpp** if there was no R system to build upon and to extend.

Finally, many members of the R and **Rcpp** communities have been very supportive at different workshops, conference presentations, and via the mailing lists. Numerous good questions and suggestions have come this way. And, of course, it is seeing this work being used so actively which motivates us and keeps us moving forward with **Rcpp**.

Contents

List of Tables

List of Figures

List of Listings