Dirk Eddelbuettel

Overview

The Rcpp package provides C++ classes that greatly facilitate interfacing C or C++ code in R packages using the `.Call()` interface provided by R.

Rcpp provides matching C++ classes for a large number of basic R data types. Hence, a package author can keep his data in normal R data structures without having to worry about translation or transfering to C++. At the same time, the data structures can be accessed as easily at the C++ level, and used in the normal manner.

The mapping of data types works in both directions. It is as straightforward to pass data from R to C++, as it is it return data from C++ to R. The following two sections list supported data types.

Transfer from R to C++, and from C++ to R

R data types (SEXP) are matched to C++ objects in a class hierarchy. All R types are supported (vectors, functions, environment, etc ...) and each type is mapped to a dedicated class. For example, numeric vectors are represented as instances of the Rcpp::NumericVector class, environments are represented as instances of Rcpp::Environment, functions are represented as Rcpp::Function, etc ...

The underlying C++ library also offers the Rcpp::wrap function which is a templated function that transforms an arbitrary object into a SEXP. This makes it straightforward to implement C++ logic in terms of standard C++ types such as STL containers and then wrap them when they need to be returned to R. Internally, wrap uses advanced template meta programming techniques and currently supports these data types: primitive types (bool, int, double, size_t, Rbyte, Rcomplex, std::string), STL containers (e.g `std::vector`) where T is wrappable, STL maps (e.g `std::map`) where T is wrappable, and arbitrary types that support implicit conversion to SEXP.

The reverse conversion (from R into C++) is performed by the Rcpp::as function template offering a similar degree of flexibility.

New features

Starting with release 0.7.1, a namespace Rcpp is provided. It contains a main class RObject as well as other classes that derive from RObject to deal with environments (ENVSXP) , "Language" for calls (LANGSXP) and the template XPTr for external pointers.

Releases 0.7.2 and later extend this to a number of additional R types along with a number of facilities for automatic conversion thanks to clever use of templates.

Release 0.8.1 adds support for exposing code in C++ directly to R using modules. The corresponding Rcpp-modules vignette has more details.

Release 0.8.3 adds *sugar*: expression templates that allow compact vectorised expression just like in R but at compiled speed; see the Rcpp-sugar vignette.

Release 0.8.6 adds special functions cherished for statistics: d/p/q/r-style for most relevant distribution, in a form that is very close to what we'd use in R.

Release 0.8.7 adds support for ReferenceClasses in R 2.12.0; this now brings S4-based ReferenceClasses in the OO style of Java or C++ to the R language.

Release 0.9.0 splits support for the legacy *classic API* into its own package RcppClassic.

Release 0.10.0 brings *Rcpp attributes*, enhanced modules support and more.

Release 0.11.0 brings simplified builds for packages using Rcpp which no longer need to link against it.

Inline use

As of version 0.7.0, Rcpp also contains a modified function 'cfunction' taken from the excellent 'inline' package by Oleg Sklyar. This allows the user to define the body of a C++ function as a standard R character vector -- which is passed to 'cfunction' along with a few other parameters. The function then builds a complete C++ source file containing a function with the given body --- and then compiles, links and loads it for us. Together with the Rcpp interface classes this makes for very easy use of C++ from R --- as everything can be done from the R prompt without any need for Makefiles, configuration settings etc pp.
As of version 0.8.1, an extended function 'cxxfunction' is used (which requiers inline 0.3.5). This function makes it easier to use C++ code with Rcpp. In particular, it enforces use of the .Call interface, adds the Rcpp amespace, and sets up exception forwarding. It employs the macros BEGIN_RCPP and END_RCPP macros to enclose the user code

Moreover, with cfunction (and cxxfunction), we can even call external libraries and have them linked as well.

Several examples of this are included with the packages; one has also been posted on my blog.

This even works on Windows if you have the working 'R tools' installed along with R. See the R-on-Windows FAQ and additional documentation.

With version 0.10.0, this has been complemented by *Rcpp attributes* which is even easier and more powerful than inline --- see the corresponding Rcpp attributes vignette for details.

Unit testing

As of version 0.12.11, about 1385 unit tests called from over 599 unit test functions are included in the package to ensure that no regressions are introduced in terms of API compatibility. The unit tests also serve as a (arguably somewhat raw) form of examples for usage. A vignette is auto-generated with the results of the unit tests.

Usage for package building

Rcpp provides a main header file `Rcpp.h` and a library inside the installed package in the directory `lib`. From within R, you can compute the directory location via `system.file("lib", "Rcpp.h", package="Rcpp")` --but both are provided for your use via the functions `Rcpp::RcppCxxFlags()` and `Rcpp::RcppLdFlags()` functions. So we can just use the following as a file `src/Makevars` (or `src/Makevars.win` on Windows)

```
PKG_CXXFLAGS=`${R_HOME}/bin/Rscript -e "Rcpp:::CxxFlags()"`

PKG_LIBS=`${R_HOME}/bin/Rscript -e "Rcpp:::LdFlags()"`
```

See the help page for `Rcpp-package` for details.

Also note that starting with version 0.8.0, the 'LinkingTo' argument can also be employed in packages using Rcpp. This will let R determine the location of the header files and users only need to use

`Rcpp::RcppLdFlags()` (as detailed above) to point to the actual library, and this is clearly the *recommended* approach.

Moreover, we added an entire vignette on how to use Rcpp in your package with a detailed discussion. Also note, the vignette for Rcpp attributes detail another approach.

Rcpp book

The book *Seamless R and C++ Integration with Rcpp* (Springer, 2013) provides a thorough and complete documentation for Rcpp, along with many examples. More information is is available here. The book can be ordered directly from Springer as well as from Amazon and other book sellers.

Rcpp Gallery

The Rcpp Gallery regroups about one hundred contributed articles and examples for Rcpp. It is open for user contributions.

Demo package

The RcppExamples package (on CRAN) provides a simple illustration of how to use Rcpp, and can also be used as a framework for deploying Rcpp. This package is however somewhat incomplete in terms of example, so please see below for examples provides by several dozen packages using Rcpp.

Class documentation

We now have Doxygen-generated documentation of all the classes in browseable and searchable html and as a pdf file. We no longer include the Doxygen-generated documentation in the source tarball as it simply too big. But we have zip archives of the html, latex, and man documentation.

Other documentation

Besides the doxygen-generated reference manual we also have these eight vignettes:

- The Rcpp-introduction vignette provides a short overview of Rcpp and an introduction (and has also been published as Volume 40, Issue 8 of theJournal of Statistical Software),
- the Rcpp-package vignette shows how to write your own package using Rcpp,
- the Rcpp-FAQ vignette addresses several frequently asked questions,
- Rcpp-modules vignette discusses how to expose C++ functions and modules with ease using an idea borrowed from Boost::Python,
- the Rcpp-extending vignette details the steps needed to extend Rcpp with user-provided or third-party classes,
- the Rcpp-sugar vignette provides an introduction to the *Rcpp sugar* features inspired by vectorised R code,
- the Rcpp-attributes vignette introduces the *attributes* features for getting C++ into R with ease,
- the Rcpp-quickref vignette provides a quick reference *cheat sheet* (but is still mostly incomplete),
- the Rcpp-attributes vignette details the high-level syntax for declaring C++ functions as callable from R and shows how to automatically generate the code required to invoke them, and
- the Rcpp-unitTests vignette contains a summary of the (by now over two hundred) units tests for Rcpp.

All vignettes are also installed with the package, and available at the CRAN page.

Google Tech Talk

In late October 2010, the *R intergrouplet* at Google was kind enough to invite us for a talk on Rcpp. The resulting talk was recorded and is now available on YouTube

Example usage

A long and growing list of packages using Rcpp is provide on a separate page.

History

Rcpp was initially written by Dominick Samperi to ease contributions to the RQuantLib project, and then released as a project in its own right. During 2006, Dominick made several releases under the RCpp name (versions 1.0 to 1.4) before he changed the name to RCppTemplate and made more releases (1.5 to 5.2). His project saw no public releases for the thirty-five months period from November 2006 to November 2009.

As a user of Rcpp, I (Dirk) chose to adopt Rcpp during 2008, made a first release 0.6.0 in November 2008 and have made a number of new releases since -- see the ChangeLog for details. Rcpp is open for contributions and patches some of which have already been integrated.

Romain Francois joined the effort just before the 0.7.0 release and brought along a lot of energy and new ideas. We now have a mailing list for discussions around Rcpp. If you have ideas or suggested changes, send an email there.

Download

A local archive is available here and at CRAN; SVN access is provided at R-Forge.

License

Rcpp is licensed under the GNU GPL version 2 or later.

Last modified: Tue May 23 06:27:27 CDT 2017