

# Package ‘Rcpp’

August 4, 2009

**Title** Rcpp R/C++ interface package

**Version** 0.6.6

**Date** \$Date: 2009-08-03 15:06:57 -0500 (Mon, 03 Aug 2009) \$

**Author** Dirk Eddelbuettel with contributions by Simon Urbanek and David Reiss

**Maintainer** Dirk Eddelbuettel <edd@debian.org>

**Description** R/C++ interface classes and examples The Rcpp library maps data types between R and C++, and includes support for R types real, integer, character, vector, matrix, Date, datetime (i.e. POSIXct) at microsecond resolution, data frame, and function. It also supports calling R functions from C++.

**Depends** R (>= 2.0.0)

**SystemRequirements** None

**License** LGPL (>= 2.1)

**Repository** CRAN

**Date/Publication** 2009-08-04 14:21:57

## R topics documented:

Rcpp-package . . . . .	2
RcppDate . . . . .	3
RcppExample . . . . .	5
RcppParams . . . . .	7
RcppResultSet . . . . .	9
RcppVector . . . . .	11
RcppVersion . . . . .	12

<b>Index</b>	<b>13</b>
--------------	-----------

## Description

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

Detailed information is provided in the vignette ‘RcppAPI’ that can be invoked from R via `vignette("RcppAPI")`.

## Overview

**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 structure without having to worry about translation or transfer 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 to return data from C++ to R. The following two sections list supported data types.

### Transfer from R to C++

Standard R datatypes that are understood are

1. named lists containing numeric (i.e. floating point), integer, character, logical (i.e. boolean) or Date and Datetime (i.e. POSIXct) arguments;
2. data frames containing numeric, integer, logical, character, Date, Datetime or Factor columns;
3. named vectors containing numeric or integer values,
4. vectors and matrices of different values
5. character strings

### Transfer from C++ to R

Standard C++ datatypes can be returned to R in a named list, the most general data type in R. Permissible components of the returns list are the following C++ types:

1. double (scalar as well as vectors and vectors of vectors),
2. int (scalar as well as vectors and vectors of vectors),
3. string,
4. STL vector types and `vector<vector>` types of int and double
5. STL vector of strings
6. internal Rcpp types `RcppDate`, `RcppDateVector`, `RcppDatetime`, `RcppDatetimeVector`, `RcppStringVector`, `RcppVector` of int or double, `RcppMatrix` of int or double, `RcppFrame`

### Usage for package building

**Rcpp** provides a header file 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")`. For Windows, it will be a static library 'Rcpp.a'. For Linux and Mac OS X, it will be 'libRcpp.so'.

To use **Rcpp** in another package, you need to include the header during compilation. This typically requires use of the `-I` to provide the location as in `-I/usr/local/lib/R/site-library/Rcpp/lib`. Similarly, for linking we need to provide the location of the library via `-L` as well as the actual library. An example for Linux would be `-L/usr/local/lib/R/site-library/Rcpp/lib -lRcpp`.

In order to make it more convenient to use **Rcpp**, functions providing these arguments were added. To use these, simply use a file `src/Makevars` such as this (which was taken from the `emdL1` package)

```
# compile flag providing header directory containing Rcpp.h
PKG_CXXFLAGS=`Rscript -e 'Rcpp::CxxFlags()'`

# link flag providing library as well as path to library, and optionally rpath
PKG_LIBS=`Rscript -e 'Rcpp::LdFlags()'`
```

Alternatively, one can also encode the test for these variables using the GNU `autoconf` system. The `RQuantLib` package provides an example of that.

Lastly, on Linux, and during development, it can be convenient to simply provide these files via soft links (or copies) from the usual locations like `/usr/local/include` and `/usr/local/lib` which alleviates the need for explicit location flags like `-I` or `-L`. Remember to also run `ldconfig` after creating the link for the library. Alternatively, you can hardcode the dynamic library location using the `rpath` linker directive:

Use on Windows is identical to the use in Linux. However only a static library is provided. The two [Rcpp](#) functions detailed above provide the relevant content.

### Author(s)

Dominick Samperi wrote most of **Rcpp** during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

---

RcppDate

*C++ classes for receiving date and datetime R objects in C++*

---

### Description

`RcppDate`, `RcppDatetime`, `RcppDateVector` and `RcppDatetimeVector` are C++ classes defined in `Rcpp.h` that can pass scalars and vectors of **R** objects of types `Date` and `POSIXct`, respectively, to C++ via the `.Call()` function interface.

Member functions are provided to query the dimension of the vector or matrix object, convert it in a corresponding C representation.

R objects of type `Date`, and hence the `RcppDate` and `RcppDateVector` objects, are internally represented as an integer counting days since the epoch, i.e. January 1, 1970. Similarly, R objects of type `POSIXct` and the `RcppDatetime` and `RcppDatetimeVector` objects, are internally represented as seconds since the epoch. However, R extends the POSIX standard by using a double leading to microsecond precision in timestamps. This is fully supported by Rcpp as well.

## Details

Usage of the `RcppDate`, `RcppDatetime` (and their vector extensions) in C++ is fully defined in `Rcpp.h`.

As example, consider a call from R to C++ such as

```
# an R example passing one type of each class to a function
# someFunction in package somePackage
val <- .Call("someFunction",
             Sys.Date(),           # current date
             Sys.time(),          # current timestamp
             as.Date("2000-02-25")
             + 0:5,               # date vector
             ISOdatetime(1999,12,31,23,59,0)
             + (0:5)*0.250,      # datetime vector
             PACKAGE="somePackage")
```

At the C++ level, the corresponding code to assign these parameter to C++ objects is can be as follows::

```
SEXP someFunction(SEXP ds, SEXP dts,
                 SEXP dvs, SEXP dtvs) {

    RcppDate          d(ds);
    RcppDatetime      dt(dts);
    RcppDateVector    dv(dvs);
    RcppDatetimeVector dtv(dtvs);
}
```

Standard accessor functions are defined, see `Rcpp.h` for details.

Objects of these types can also be returned via `RcppResultSet`.

## Author(s)

Dominick Samperi wrote most of Rcpp during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

## See Also

`RcppResultSet`, the vignette “RcppAPI”.

**Examples**

```
# set up date and datetime vectors
dvec <- Sys.Date() + -2:2
dtvec <- Sys.time() + (-2:2)*0.5

# call the underlying C++ function
result <- RcppDateExample(dvec, dtvec)

# inspect returned object
result
```

---

RcppExample	<i>Rcpp R/C++ interface example</i>
-------------	-------------------------------------

---

**Description**

RcppExample illustrates how the Rcpp R/C++ interface class library is used.

**Usage**

```
RcppExample(params, nlist, numvec, nummat, df, datevec, stringvec,
            fnvec, fnlist)
## S3 method for class 'RcppExample':
print(x, ...)
```

**Arguments**

params	A heterogeneous list specifying method (string), tolerance (double), maxIter (int).
nlist	a list of named numeric values (double or int).
numvec	a numeric 1D vector (double or int).
nummat	a numeric 2D matrix (double or int).
df	a data frame.
datevec	a vector of Date's.
stringvec	a vector of strings.
fnvec	an R function with numeric vector argument.
fnlist	an R function with list argument.
x	Object of type RcppExample.
...	Extra named parameters.

## Details

The C++ representation of data frames are not passed back to R in a form that R recognizes as a data frame, but it is a simple matter to do the conversion. For example, the return value named `PreDF` (see return values below) is not seen as a data frame on the R side (thus the name "pre-data frame"), but it can be converted to a data frame using `df <- data.frame(result$PreDF)`.

The `print.RcppExample()` function is defined so that we can control what gets printed when a variable assigned the return value is entered on a line by itself. It is defined to simply list the names of the fields returned (see `RcppExample.R`).

The source files for this example and for the `Rcpp` class library can be found in the `RcppTemplate` package source archive (the `.tar.gz` file).

## Value

`RcppExample` returns a list containing:

<code>method</code>	string input paramter
<code>tolerance</code>	double input paramter
<code>maxIter</code>	int input parameter
<code>n1FirstName</code>	first name in nlist
<code>n1FirstValue</code>	first value in nlist
<code>matD</code>	R matrix from an <code>RcppMatrix&lt;double&gt;</code> object
<code>stlvec</code>	R vector from a <code>vector&lt;double&gt;</code> object
<code>stlmat</code>	R matrix from a <code>vector&lt;vector&lt;double&gt;&gt;</code> object
<code>a</code>	R matrix from C/C++ matrix
<code>v</code>	R vector from C/C++ vector
<code>strings</code>	R vector of strings from <code>vector&lt;string&gt;</code> object
<code>InputDF</code>	a data frame passed in from R
<code>PreDF</code>	a data frame created on C++ side to be passed back to R
<code>params</code>	input parameter list (this is redundant because we returned the input parameters above)

## Author(s)

Dominick Samperi wrote most of `Rcpp` during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

## References

Samperi, D., (2006) *Rcpp: R/C++ Interface Classes: Using C++ Libraries from R*, available as a package vignette, or in the `RcppTemplate` package `doc` subdirectory (`RcppAPI.pdf`).

*Writing R Extensions*, available at <http://www.r-project.org>.

**Examples**

```

params <- list(method='BFGS',
              tolerance=1.0e-8,
              maxIter=1000,
              startDate=as.Date('2006-7-15'))

nlist <- list(ibm = 80.50, hp = 53.64, c = 45.41)

numvec <- seq(1,5) # numerical vector

nummat <- matrix(seq(1,20),4,5) # numerical matrix

stringvec <- c("hello", "world", "fractal") # string vector

datestr <- c('2006-6-10', '2006-7-12', '2006-8-10')
datevec <- as.Date(datestr, "%Y-%m-%d") # date vector

df <- data.frame(a=c(TRUE, TRUE, FALSE), b=I(c('a','b','c')),
               c=c('beta', 'beta', 'gamma'), dates=datevec)

fnvec <- function(x) { sum(x) } # Add up components of vector

fnlist <- function(l) { # Return vector with 1 added to each component
  vec <- c(l$alpha + 1, l$beta + 1, l$gamma + 1)
  vec
}

result <- RcppExample(params, nlist, numvec, nummat, df, datevec,
                    stringvec, fnvec, fnlist)

result

```

---

RcppParams

*C++ class for receiving (scalar) parameters from R*


---

**Description**

RcppParams is a C++ class defined in Rcpp.h that receive any number of scalar parameters of types in a single named list object from R through the .Call() function.

The parameters can be of different types that are limited to the R types numeric, integer, character, logical or Date. These types are mapped into, respectively, the corresponding C++ types double, int, string, bool and Date (a custom class defined by Rcpp).

**Usage**

```
val <- RcppParamsExample(params)
```

**Arguments**

`params` A heterogeneous list specifying `method` (string), `tolerance` (double), `maxIter` (int) and `startDate` (Date in R, RcppDate in C++).

**Details**

Usage of RcppParams from R via `.Call()` is as follows:

```
# an R example passing one type of each class to a function
# someFunction in package somePackage
val <- .Call("someFunction",
             list(pie=3.1415, magicianswer=42, sometext="foo",
                 yesno=true, today=Sys.date()),
             PACKAGE="somePackage")
```

At the C++ level, the corresponding code to assign these parameter to C++ objects is

```
SEXP someFunction(SEXP params) {
    RcppParams par(params);
    double p = RcppParams.getDoubleValue("pie");
    int magic = RcppParams.getIntValue("magicianswer");
    string txt = RcppParams.getStringValue("sometext");
    bool yn = RcppParams.getBoolValue("yesno");
    RcppDate d = RcppParams.getDateValue("today");
    // some calculations ...
    // some return values ...
}
```

As the lookup is driven by the names given at the R level, order is not important. It is however important that the types match. Errors are typically caught and an exception is thrown.

The class member function `checkNames` can be used to verify that the SEXP object passed to the function contains a given set of named object.

**Value**

RcppExample returns a list containing:

<code>method</code>	string input paramter
<code>tolerance</code>	double input paramter
<code>maxIter</code>	int input parameter
<code>startDate</code>	Date type with starting date
<code>params</code>	input parameter list (this is redundant because we returned the input parameters above)

**Author(s)**

Dominick Samperi wrote most of Rcpp during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

**See Also**

RcppExample, the vignette “RcppAPI”.

**Examples**

```
# set up some value
params <- list(method='BFGS',
               tolerance=1.0e-5,
               maxIter=100,
               startDate=as.Date('2006-7-15'))

# call the underlying C++ function
result <- RcppParamsExample(params)

# inspect returned object
result
```

---

RcppResultSet      *C++ class for sending C++ objects back to R*

---

**Description**

RcppResultSet is a C++ class defined in Rcpp.h that can assign any number of C++ objects to R in a single named list object as the SEXP return value of a .Call() function call.

The C++ objects can be of different types that are limited to types double, int, string, vectors of double or int (with explicit dimensions), matrices of double or int (with explicit dimensions), STL vectors of double, int or string, STL ‘vector of vectors’ of types double or int (all with implicit dimensions), the internal types RcppDate, RcppDateVector, RcppStringVector, RcppVector of types double or int, RcppMatrix of types double or int as well RcppFrame, a type that can be converted into a data.frame, and the R type SEXP.

Where applicable, the C++ types are automatically converted to the corresponding R types structures around types numeric, integer, or character. The C++ code can all be retrieved in R as elements of a named list object.

**Details**

Usage of RcppResultSet from C++ is fully defined in Rcpp.h. An example for returning data to R at the end of a .Call() call follows.

At the C++ level, the corresponding code to assign these parameter to C++ objects is can be as follows (taken from the C++ source of RcppExample):

```
SEXP rl;
RcppResultSet rs;

rs.add("date", aDate);            // RcppDate
```

```

rs.add("dateVec", dateVec); // RcppDateVec
rs.add("method", method); // string
rs.add("tolerance", tol); // numeric
rs.add("maxIter", maxIter); // int
rs.add("matD", matD); // RcppMatrix
rs.add("stlvec", stlvec); // vector<double> or <int>
rs.add("stlmat", stlmat); // vector< vector <double> >
// or <int>
rs.add("a", a, nrows, ncols); // double** (or int**) with
// two dimension
rs.add("v", v, len); // double* (or int*) with
// one dimension
rs.add("stringVec", strVec); // RcppStringVector
rs.add("strings", svec); // vector<string>
rs.add("InputDF", inframe); // RcppFrame
rs.add("PreDF", frame); // RcppFrame

rl = rs.getReturnList();
return(rl);

```

As the R level, we assign the returned object a list variables from which we select each list element by its name. lookup is driven by the names given at the R level, order is not important. It is however important that the types match. Errors are typically caught and an exception is thrown.

The class member function `checkNames` can be used to verify that the SEXP object passed to the function contains a given set of named object.

### Author(s)

Dominick Samperi wrote most of Rcpp during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

### See Also

`RcppExample`, the vignette “RcppAPI”.

### Examples

```

# example from RcppDate
# set up date and datetime vectors
dvec <- Sys.Date() + -2:2
dtvec <- Sys.time() + (-2:2)*0.5

# call the underlying C++ function
result <- RcppDateExample(dvec, dtvec)

# inspect returned object
result

```

---

RcppVector                      C++ classes for receiving R object in C++

---

## Description

RcppVector, RcppMatrix and RcppStringVector are C++ classes defined in Rcpp.h that can pass vectors (matrices) of R objects of appropriate types to C++ via the .Call() function interface.

The vector and matrix types are templated and can operate on R types integer and numeric.

Member functions are provided to query the dimension of the vector or matrix object, convert it in a corresponding C representation, and also to convert it into a corresponding STL object.

## Details

Usage of RcppVector, RcppMatrix and RcppStringVector in C++ is fully defined in Rcpp.h.

As example, consider a call from R to C++ such as

```
# an R example passing one type of each class to a function
# someFunction in package somePackage
val <- .Call("someFunction",
             rnorm(100),           # numeric vector
             sample(1:10, 5, TRUE) # int vector
             search(),            # character vector
             as.matrix(rnorm(100),10,10), # matrix
             PACKAGE="somePackage")
```

At the C++ level, the corresponding code to assign these parameter to C++ objects is can be as follows (taken from the C++ source of RcppExample):

```
SEXP someFunction(SEXP nvec, SEXP ivec,
                  SEXP svec, SEXP nmat) {

    RcppVector<double> nv(nvec);
    RcppVector<int>    iv(ivec);
    RcppStringVector  sv(svec);
    RcppMatrix<double> nm(nmat);
}
```

These C++ objects could then be queried via

```
int n = nv.size();
int d1 = nm.dim1(), d2 = nm.dim2();
```

to retrieve, respectively, vector length and matrix dimensions.

Moreover, the stlVector() and stlMatrix() member functions can be used to convert the objects into STL objects:

```
vector<int> ivstl = iv.stlVector();  
vector< vector< double > > = nm.stlMatrix();
```

**Author(s)**

Dominick Samperi wrote most of Rcpp during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

**See Also**

RcppExample, the vignette “RcppAPI”.

**Examples**

```
# set up some value  
vector <- (seq(1,9))^2  
  
# call the underlying C++ function  
result <- RcppVectorExample(vector)  
  
# inspect returned object  
result
```

---

RcppVersion

*Display version info for package and for Rcpp API*

---

**Description**

RcppVersion displays version information about the package, along with the version of Rcpp that was used to build the package.

**Usage**

```
RcppVersion()
```

**Author(s)**

Dominick Samperi wrote most of Rcpp during 2005 and 2006. Dirk Eddelbuettel made some additions, and became maintainer in 2008.

**Examples**

```
RcppVersion()
```

# Index

## \*Topic **interface**

- Rcpp-package, 1
- RcppDate, 3
- RcppExample, 5
- RcppParams, 7
- RcppResultSet, 9
- RcppVector, 10
- RcppVersion, 12

## \*Topic **programming**

- Rcpp-package, 1
- RcppDate, 3
- RcppExample, 5
- RcppParams, 7
- RcppResultSet, 9
- RcppVector, 10

`print.RcppExample (RcppExample), 5`

`Rcpp, 3`

`Rcpp (Rcpp-package), 1`

`Rcpp-package, 1`

`RcppDate, 3`

`RcppDateExample (RcppDate), 3`

`RcppDatetime (RcppDate), 3`

`RcppDatetimeVector (RcppDate), 3`

`RcppDateVector (RcppDate), 3`

`RcppExample, 5`

`RcppMatrix (RcppVector), 10`

`RcppParams, 7`

`RcppParamsExample (RcppParams), 7`

`RcppResultSet, 9`

`RcppStringVector (RcppVector), 10`

`RcppVector, 10`

`RcppVectorExample (RcppVector), 10`

`RcppVersion, 12`