

# Rcpp by Examples

Dr. Dirk Eddelbuettel

`dirk@eddelbuettel.com`

`@eddelbuettel`

Workshop preceding **R/Finance** 2014

University of Illinois at Chicago

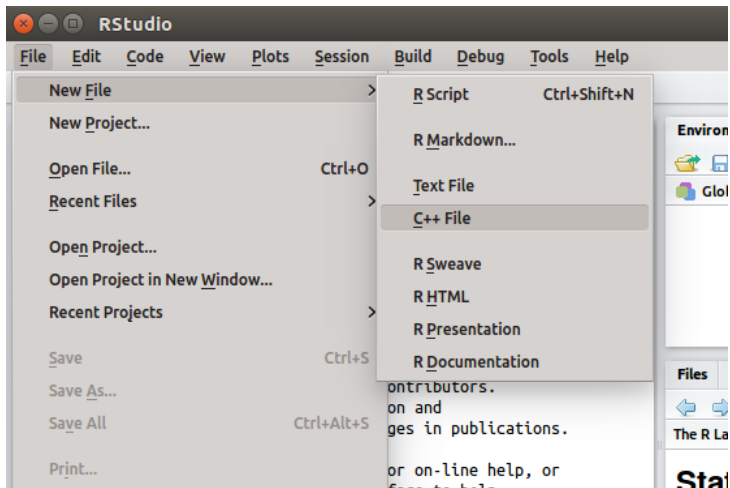
17 May 2013

# Outline

- 1 **Intro**
- 2 Usage
- 3 Sugar
- 4 Examples
- 5 More

# Jumping right in

RStudio makes starting very easy:



# A First Example: Cont'd

The following file gets created:

```
#include <Rcpp.h>
using namespace Rcpp;

// Below is a simple example of exporting a C++ function to R.
// You can source this function into an R session using the
// Rcpp::sourceCpp function (or via the Source button on the
// editor toolbar)

// For more on using Rcpp click the Help button on the editor
// toolbar

// [[Rcpp::export]]
int timesTwo(int x) {
    return x * 2;
}
```

# A First Example: Cont'd

We can easily deploy the file ("press the button") and call the resulting function:

```
Rcpp::sourceCpp('files/timesTwo.cpp')  
timesTwo(21)  
  
## [1] 42
```

# A First Example: Cont'd

So what just happened?

- We defined a simple C++ function
- It operates on a single integer argument
- We asked Rcpp to 'source it' for us
- Behind the scenes Rcpp creates a wrapper
- Rcpp then compiles, links, and loads the wrapper
- The function is available in R under its C++ name

# A First Example: Related

Two related functions related to `sourceCpp()`:

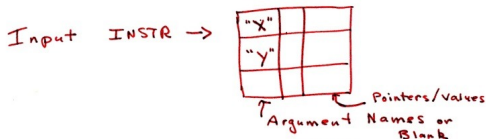
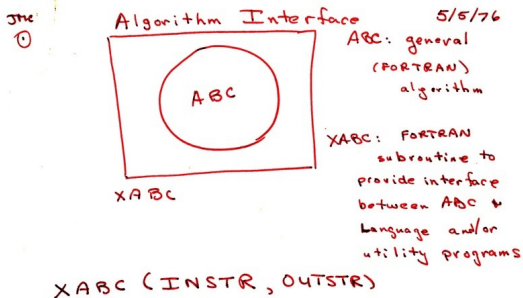
```
evalCpp("2 * 2")
```

```
## [1] 4
```

```
cppFunction("int times2(int x) { return 2*x;}")  
times2(123)
```

```
## [1] 246
```

# A "vision" from Bell Labs from 1976



Source: John Chambers' talk at Stanford in October 2010; personal correspondence.



# An Introductory Example

Consider a function defined as

$$f(n) \text{ such that } \begin{cases} n & \text{when } n < 2 \\ f(n-1) + f(n-2) & \text{when } n \geq 2 \end{cases}$$

# An Introductory Example: Simple R Implementation

R implementation and use:

```
f <- function(n) {  
  if (n < 2) return(n)  
  return(f(n-1) + f(n-2))  
}  
  
## Using it on first 11 arguments  
sapply(0:10, f)  
  
## [1] 0 1 1 2 3 5 8 13 21 34 55
```

# An Introductory Example: Timing R Implementation

Timing:

```
library(rbenchmark)
benchmark(f(10), f(15), f(20))[,1:4]

##      test replications elapsed relative
## 1 f(10)           100    0.020         1.0
## 2 f(15)           100    0.224        11.2
## 3 f(20)           100    2.327       116.3
```

# An Introductory Example: C++ Implementation

```
int g(int n) {  
    if (n < 2) return(n);  
    return(g(n-1) + g(n-2));  
}
```

Deployed as:

```
library(Rcpp)  
cppFunction('int g(int n) {  
    if (n < 2) return(n);  
    return(g(n-1) + g(n-2)); }')  
## Using it on first 11 arguments  
sapply(0:10, g)  
  
## [1] 0 1 1 2 3 5 8 13 21 34 55
```

# An Introductory Example: Comparing timing

Timing:

```
library(rbenchmark)
benchmark(f(20), g(20))[,1:4]

##      test replications elapsed relative
## 1 f(20)           100    2.381    216.5
## 2 g(20)           100    0.011     1.0
```

A nice gain of a few orders of magnitude.

# Well-known packages using Rcpp

`Amelia` by G King et al

`lme4` by D Bates, M Maechler et al

`forecast` by R Hyndman et al

`RStan` by A Gelman et al

`rugarch` by A Ghalanos

`plyr` by H Wickham (plus **roxygen2**, **dplyr**, ...)

`httpuv` by J Cheng / RStudio

`MTS` by R Tsay

Rcpp is currently used by 212 CRAN packages, and a further 27 BioConductor packages.

# Type mapping

Standard R types (integer, numeric, list, function, ... and compound objects) are mapped to corresponding C++ types using extensive template meta-programming – it just works:

```
library(Rcpp)
cppFunction("NumericVector la(NumericVector x){
  return log(abs(x));
}")
la(seq(-5, 5, by=2))

## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```

Also note: vectorized C++!

# Type mapping also with C++ STL types

Use of `std::vector<double>` and STL algorithms:

```
#include <Rcpp.h>
using namespace Rcpp;

inline double f(double x) { return ::log(::fabs(x)); }

// [[Rcpp::export]]
std::vector<double> logabs2(std::vector<double> x) {
    std::transform(x.begin(), x.end(), x.begin(), f);
    return x;
}
```



# Type mapping also with C++ STL types

Used via

```
library(Rcpp)
sourceCpp("code/logabs2.cpp")
logabs2(seq(-5, 5, by=2))

## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```

# Type mapping is seamless

Simple outer product of a column vector (using Armadillo / RcppArmadillo):

```
cppFunction("arma::mat v(arma::colvec a) {
    return a*a.t();}",
    depends="RcppArmadillo")
v(1:5)
```

##	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]
## [1, ]	1	2	3	4	5
## [2, ]	2	4	6	8	10
## [3, ]	3	6	9	12	15
## [4, ]	4	8	12	16	20
## [5, ]	5	10	15	20	25

This uses implicit conversion via `as<>` and `wrap` – cf package vignette `Rcpp-extending`.

# C++11: lambdas, auto, and much more

We can simplify the `log(abs(...))` example further:

```
#include <Rcpp.h>

// [[Rcpp::plugins(cpp11)]]

using namespace Rcpp;

// [[Rcpp::export]]
std::vector<double> logabs3(std::vector<double> x) {
    std::transform(x.begin(), x.end(), x.begin(),
                  [](double x) {
                      return ::log(::fabs(x));
                  } );
    return x;
}
```

# C++11: lambdas, auto, and much more

Used via

```
library(Rcpp)
sourceCpp("code/logabs3.cpp")
logabs3(seq(-5, 5, by=2))

## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```

# Outline

- 1 Intro
- 2 Usage
- 3 Sugar
- 4 Examples
- 5 More

# Basic Usage: evalCpp

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

```
evalCpp("2 + 2")           # simple test  
  
## [1] 4  
  
evalCpp("std::numeric_limits<double>::max()")  
  
## [1] 1.798e+308
```

# Basic Usage: `cppFunction()`

`cppFunction()` creates, compiles and links a C++ file, and creates an R function to access it.

```
cppFunction ("
  int useCpp11() {
    auto x = 10;
    return x;
  }", plugins=c("cpp11"))
useCpp11() # same identifier as C++ function

## [1] 10
```

## Basic Usage: `sourceCpp()`

`sourceCpp()` is the actual workhorse behind `evalCpp()` and `cppFunction()`. It is described in more detail in the package vignette `Rcpp-attributes`.

`sourceCpp()` builds on and extends `cxxfunction()` from package `inline`, but provides even more ease-of-use, control and helpers – freeing us from boilerplate scaffolding.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf `RcppArmadillo`, `RcppEigen`, `RcppGSL`).



# Basic Usage: Packages

Package are *the* standard unit of R code organization.

Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette [Rcpp-package](#) has fuller details.

As of early May 2014, there are 208 packages on CRAN which use Rcpp, and a further 27 on BioConductor — with working, tested, and reviewed examples.

# Packages and Rcpp

Best way to organize R code with Rcpp is via a package:

The screenshot shows the RStudio interface. The main editor displays a C++ file named `foo.cpp` with the following code:

```

1 #include <Rcpp.h>
2 using namespace Rcpp;
3
4 // Below is a simple example of exporting a C++ function to R. You
5 // source this function into an R session using the Rcpp::sourceCpp()
6 // function (or via the R console).
7
8 // For more on using Rcpp, see the Rcpp website:
9 // http://www.Rcpp.org
10 // [[Rcpp::export]]
11 int timesTwo(int x)
12 {
13   return x * 2;
14 }

```

The console window shows the following output:

```

> sourceCpp("files/timesTwoA.cpp")
Error: file not found: 'files/timesTwoA.cpp'
In addition: Warning message:
In normalizePath(file, winslash = "/") :
path[1]='files/timesTwoA.cpp': No such file or directory
> getwd()
[1] "/home/edd"
>

```

The 'Create R Package' dialog box is open, showing the following options:

- Type:** Package w/ Rcpp
- Package name:** (empty field)
- Create package based on source files:** (empty list with 'Add...' and 'Remove' buttons)
- Create project as subdirectory of:** (empty field with 'Browse...' button)
- Create a git repository for this project
- Open in new window
- Buttons:** Create Project, Cancel

The background shows the RStudio Environment and History panes, and a 'Reference' section with links to R documentation:

- [An Introduction to R](#)
- [The R Language Definition](#)
- [Writing R Extensions](#)
- [R Installation and Administration](#)
- [R Data Import/Export](#)
- [R Internals](#)

# Packages and Rcpp

`Rcpp.package.skeleton()` and its derivatives. e.g. `RcppArmadillo.package.skeleton()` create working packages.

```
// another simple example: outer product of a vector,  
// returning a matrix  
//  
// [[Rcpp::export]]  
arma::mat rcpparma_outerproduct(const arma::colvec & x) {  
    arma::mat m = x * x.t();  
    return m;  
}  
  
// and the inner product returns a scalar  
//  
// [[Rcpp::export]]  
double rcpparma_innerproduct(const arma::colvec & x) {  
    double v = arma::as_scalar(x.t() * x);  
    return v;  
}
```

# Outline

- 1 Intro
- 2 Usage
- 3 Sugar**
- 4 Examples
- 5 More

# Syntactic 'sugar': Simulating $\pi$ in R

Basic idea: for point  $(x, y)$ , compute distance to origin. Do so repeatedly, and ratio of points below one to number  $N$  of simulations will approach  $\pi/4$  as we fill the area of one quarter of the unit circle.

```
piR <- function(N) {  
  x <- runif(N)  
  y <- runif(N)  
  d <- sqrt(x^2 + y^2)  
  return(4 * sum(d <= 1.0) / N)  
}  
  
set.seed(5)  
sapply(10^(3:6), piR)  
  
## [1] 3.156 3.155 3.139 3.141
```

# Syntactic 'sugar': Simulating $\pi$ in C++

The neat thing about Rcpp sugar enables us to write C++ code that looks almost as compact.

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double piSugar(const int N) {
  RNGScope scope; // ensure RNG gets set/reset
  NumericVector x = runif(N);
  NumericVector y = runif(N);
  NumericVector d = sqrt(x*x + y*y);
  return 4.0 * sum(d <= 1.0) / N;
}
```

Apart from RNG set/reset, the code is essentially identical.

# Syntactic 'sugar': Simulating $\pi$

And by using the same RNG, so are the results.

```
sourceCpp("code/piSugar.cpp")
set.seed(42); a <- piR(1.0e7)
set.seed(42); b <- piSugar(1.0e7)
identical(a,b)

## [1] TRUE

print(c(a,b), digits=7)

## [1] 3.140899 3.140899
```

# Syntactic 'sugar': Simulating $\pi$

The performance is close with a small gain for C++ as R is already vectorised:

```
library(rbenchmark)
benchmark(piR(1.0e6), piSugar(1.0e6))[,1:4]

##           test replications elapsed relative
## 1      piR(1e+06)           100    20.35      1.000
## 2 piSugar(1e+06)           100    31.89      1.567
```

More about Sugar is in the [package vignette Rcpp-sugar](#).



# Outline

- 1 Intro
- 2 Usage
- 3 Sugar
- 4 Examples**
- 5 More

# Cumulative Sum

<http://gallery.rcpp.org/articles/vector-cumulative-sum/>

A basic looped version:

```
#include <Rcpp.h>
#include <numeric>      // for std::partial_sum
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector cumsum1(NumericVector x){
  // initialize an accumulator variable
  double acc = 0;

  // initialize the result vector
  NumericVector res(x.size());

  for(int i = 0; i < x.size(); i++){
    acc += x[i];
    res[i] = acc;
  }
  return res;
}
```

# Cumulative Sum

See <http://gallery.rcpp.org/articles/vector-cumulative-sum/>

An STL variant:

```
// [[Rcpp::export]]
NumericVector cumsum2(NumericVector x){
  // initialize the result vector
  NumericVector res(x.size());
  std::partial_sum(x.begin(), x.end(), res.begin());
  return res;
}
```

# Cumulative Sum

<http://gallery.rcpp.org/articles/vector-cumulative-sum/>

Or just Rcpp sugar:

```
// [[Rcpp::export]]
NumericVector cumsum_sug(NumericVector x){
  return cumsum(x); // compute + return result vector
}
```

Of course, all results are the same.

```
cppFunction('NumericVector cumsum2(NumericVector x) {
  return cumsum(x); }')
x <- 1:10
all.equal(cumsum(x), cumsum2(x))

## [1] TRUE
```

# Calling an R function from C++

<http://gallery.rcpp.org/articles/r-function-from-c++/>

```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector callFunction(NumericVector x,
                           Function f) {
    NumericVector res = f(x);
    return res;
}

/** R
callFunction(x, fivenum)
*/
```

# Using Boost via BH

<http://gallery.rcpp.org/articles/using-boost-with-bh/>

```
// [[Rcpp::depends(BH)]]
#include <Rcpp.h>

// One include file from Boost
#include <boost/date_time/gregorian/gregorian_types.hpp>

using namespace boost::gregorian;

// [[Rcpp::export]]
Rcpp::Date getIMMDate(int mon, int year) {
  // compute third Wednesday of given month / year
  date d = nth_day_of_the_week_in_month(
    nth_day_of_the_week_in_month::third,
    Wednesday, mon).get_date(year);
  date::ymd_type ymd = d.year_month_day();
  return Rcpp::wrap(Rcpp::Date(ymd.year, ymd.month, ymd.day));
}
```

# Using Boost via BH

<http://gallery.rcpp.org/articles/boost-foreach/>

```
#include <Rcpp.h>
#include <boost/foreach.hpp>
using namespace Rcpp;
// [[Rcpp::depends(BH)]]

// the C-style upper-case macro name is a bit ugly
#define foreach BOOST_FOREACH

// [[Rcpp::export]]
NumericVector square( NumericVector x ) {

  // elem is a reference to each element in x
  // we can re-assign to these elements as well
  foreach( double& elem, x ) {
    elem = elem*elem;
  }
  return x;
}
```

C++11 now has something similar in a smarter `for` loop.

# Vector Subsetting

<http://gallery.rcpp.org/articles/subsetting/>

## New in **Rcpp** 0.11.1:

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector positives(NumericVector x) {
    return x[x > 0];
}

// [[Rcpp::export]]
List first_three(List x) {
    IntegerVector idx = IntegerVector::create(0, 1, 2);
    return x[idx];
}

// [[Rcpp::export]]
List with_names(List x, CharacterVector y) {
    return x[y];
}
```



# Armadillo Eigenvalues

<http://gallery.rcpp.org/articles/armadillo-eigenvalues/>

```
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::vec getEigenValues(arma::mat M) {
    return arma::eig_sym(M);
}
```

```
set.seed(42)
X <- matrix(rnorm(4*4), 4, 4)
Z <- X %*% t(X)
getEigenValues(Z)

# R gets the same results (in reverse)
# and also returns the eigenvectors.
```

# Converting C to C++: A plyr example

<http://gallery.rcpp.org/articles/plyr-c-to-rcpp/>

*The job of `split_indices()` is simple: given a vector `x` of integers, it returns a list where the  $i$ -th element of the list is an integer vector containing the positions of `x` equal to  $i$ .*

I will spare you the C API version.

# Converting C to C++: A plyr example

<http://gallery.rcpp.org/articles/plyr-c-to-rcpp/>

```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::exports]]
std::vector<std::vector<int> >
split_indices(IntegerVector x, int n = 0) {
  if (n < 0) stop("n must be a pos. int.");

  std::vector<std::vector<int> > ids(n);

  int nx = x.size();
  for (int i = 0; i < nx; ++i) {
    if (x[i] > n) {
      ids.resize(x[i]);
    }
    ids[x[i] - 1].push_back(i + 1);
  }
  return ids;
}
```

# Creating xts objects in C++

<http://gallery.rcpp.org/articles/creating-xts-from-c++/>

```
#include <Rcpp.h>
using namespace Rcpp;

NumericVector createXts(int sv, int ev) {
  IntegerVector ind = seq(sv, ev);      // values

  NumericVector dv(ind);                // date(time)s == reals
  dv = dv * 86400;                       // scaled to days
  dv.attr("tzone") = "UTC";            // index has attributes
  dv.attr("tclass") = "Date";

  NumericVector xv(ind);                // data has same index
  xv.attr("dim") = IntegerVector::create(ev-sv+1, 1);
  xv.attr("index") = dv;
  CharacterVector cls = CharacterVector::create("xts", "zoo");
  xv.attr("class") = cls;
  xv.attr(".indexCLASS") = "Date";
  // ... some more attributes ...

  return xv;
}
```

# Passing user-defined C(++) functions R to C++

<http://gallery.rcpp.org/articles/passing-cpp-function-pointers/>

```
// [[Rcpp::depends(RcppArmadillo)]]
#include <RcppArmadillo.h>

// [[Rcpp::export]]
arma::vec fun_cpp(const arma::vec& x) { return(10*x); }

typedef arma::vec (*funcPtr)(const arma::vec& x);

// [[Rcpp::export]]
Rcpp::XPtr<funcPtr> putFunPtrInXPtr() {
    return(Rcpp::XPtr<funcPtr>(new funcPtr(&fun_cpp)));
}

// [[Rcpp::export]]
arma::vec callViaXPtr(const arma::vec x, SEXP xpsexp) {
    Rcpp::XPtr<funcPtr> xpfun(xpsexp);
    funcPtr fun = *xpfun;
    arma::vec y = fun(x);
    return(y);
}
```

# Passing user-defined C(++) functions R to C++

<http://gallery.rcpp.org/articles/passing-cpp-function-pointers/>

Quick illustration:

```
fun <- putFunPtrInXPtr()  
callViaXPtr(1:4, fun)
```

```
##           [,1]  
## [1,]      10  
## [2,]      20  
## [3,]      30  
## [4,]      40
```

# Outline

- 1 [Intro](#)
- 2 [Usage](#)
- 3 [Sugar](#)
- 4 [Examples](#)
- 5 [More](#)

# Documentation

- The package comes with eight pdf vignettes, and numerous help pages.
- The introductory vignettes are now published (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp. Stat. & Data Anal.*).
- The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.
- By now StackOverflow has a fair number of posts too.
- And a number of blog posts introduce/discuss features.



# Rcpp Gallery

Rcpp Gallery - Google Chrome

gallery.rcpp.org

Rcpp Projects Gallery Book Events More

## Featured Articles

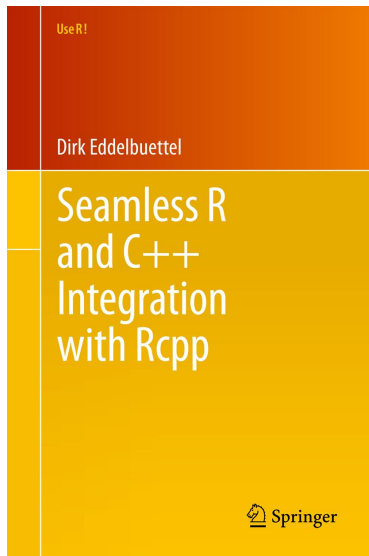
- [Quick conversion of a list of lists into a data frame](#) — John Merrill  
This post shows one method for creating a data frame quickly
- [Passing user-supplied C++ functions](#) — Dirk Eddelbuettel  
This example shows how to select user-supplied C++ functions
- [Using Rcpp to access the C API of xts](#) — Dirk Eddelbuettel  
This post shows how to use the exported API functions of xts
- [Timing normal RNGs](#) — Dirk Eddelbuettel  
This post compares drawing  $N(0,1)$  vectors from R, Boost and C++11
- [A first lambda function with C++11 and Rcpp](#) — Dirk Eddelbuettel  
This post shows how to play with lambda functions in C++11
- [First steps in using C++11 with Rcpp](#) — Dirk Eddelbuettel  
This post shows how to experiment with C++11 features
- [Using Rcout for output synchronised with R](#) — Dirk Eddelbuettel  
This post shows how to use Rcout (and Rcerr) for output
- [Using the Rcpp sugar function clamp](#) — Dirk Eddelbuettel  
This post illustrates the sugar function clamp
- [Using the Rcpp Timer](#) — Dirk Eddelbuettel  
This post shows how to use the Timer class in Rcpp
- [Calling R Functions from C++](#) — Dirk Eddelbuettel  
This post discusses calling R functions from C++

[More »](#)

## Recently Published

- Apr 12, 2013 » [Using the RcppArmadillo-based Implementation of R's sample\(\)](#) — Christian Gunning and Jonathan Olmsted
- Apr 8, 2013 » [Dynamic Wrapping and Recursion with Rcpp](#) — Kevin Ushey
- Mar 14, 2013 » [Using bigmemory with Rcpp](#) — Michael Kane
- Mar 12, 2013 » [Generating a multivariate gaussian distribution using RcppArmadillo](#) — Ahmadou Dicko
- Mar 1, 2013 » [Using Rcpp with Boost.Regex for regular expression](#) — Dirk Eddelbuettel
- Feb 27, 2013 » [Fast factor generation with Rcpp](#) — Kevin Ushey

# The Rcpp book



On sale since June  
2013.