

Rcpp: Seamless R and C++

Romain François

romain@r-enthusiasts.com

Joint work with Dirk Eddelbuettel

RMetrics, June 2010, MeielSalp.

Fine for Indiana Jones



Le viaduc de Millau



Plat du jour

- 1 Appetizers : Some background on R and C++
- 2 Main course : The Rcpp API
- 3 Desert : Rcpp sugar
- 4 Coffee : Rcpp modules

R support for C/C++

- R is a C program
- R supports C++ out of the box, just use a .cpp file extension
- R exposes a API based on low level C functions and MACROS.
- R provides several calling conventions to invoke compiled code.

```
SEXP foo( SEXP x1, SEXP x2 ){  
    ...  
}
```

```
> .Call( "foo", 1:10, rnorm(10) )
```

.Call example

```
#include <R.h>
#include <Rdefines.h>
extern "C" SEXP convolve2(SEXP a, SEXP b){
  int i, j, na, nb, nab;
  double *xa, *xb, *xab; SEXP ab;
  PROTECT(a = AS_NUMERIC(a));
  PROTECT(b = AS_NUMERIC(b));
  na=LENGTH(a); nb=LENGTH(b); nab=na+nb-1;
  PROTECT(ab = NEW_NUMERIC(nab));
  xa=NUMERIC_POINTER(a); xb=NUMERIC_POINTER(b);
  xab=NUMERIC_POINTER(ab);
  for(i=0; i<nab; i++) xab[i] = 0.0;
  for(i=0; i<na; i++) for(j=0; j<nb; j++)
    xab[i+j] += xa[i]*xb[j];
  UNPROTECT(3);
  return (ab);
}
```

.Call example: character vectors

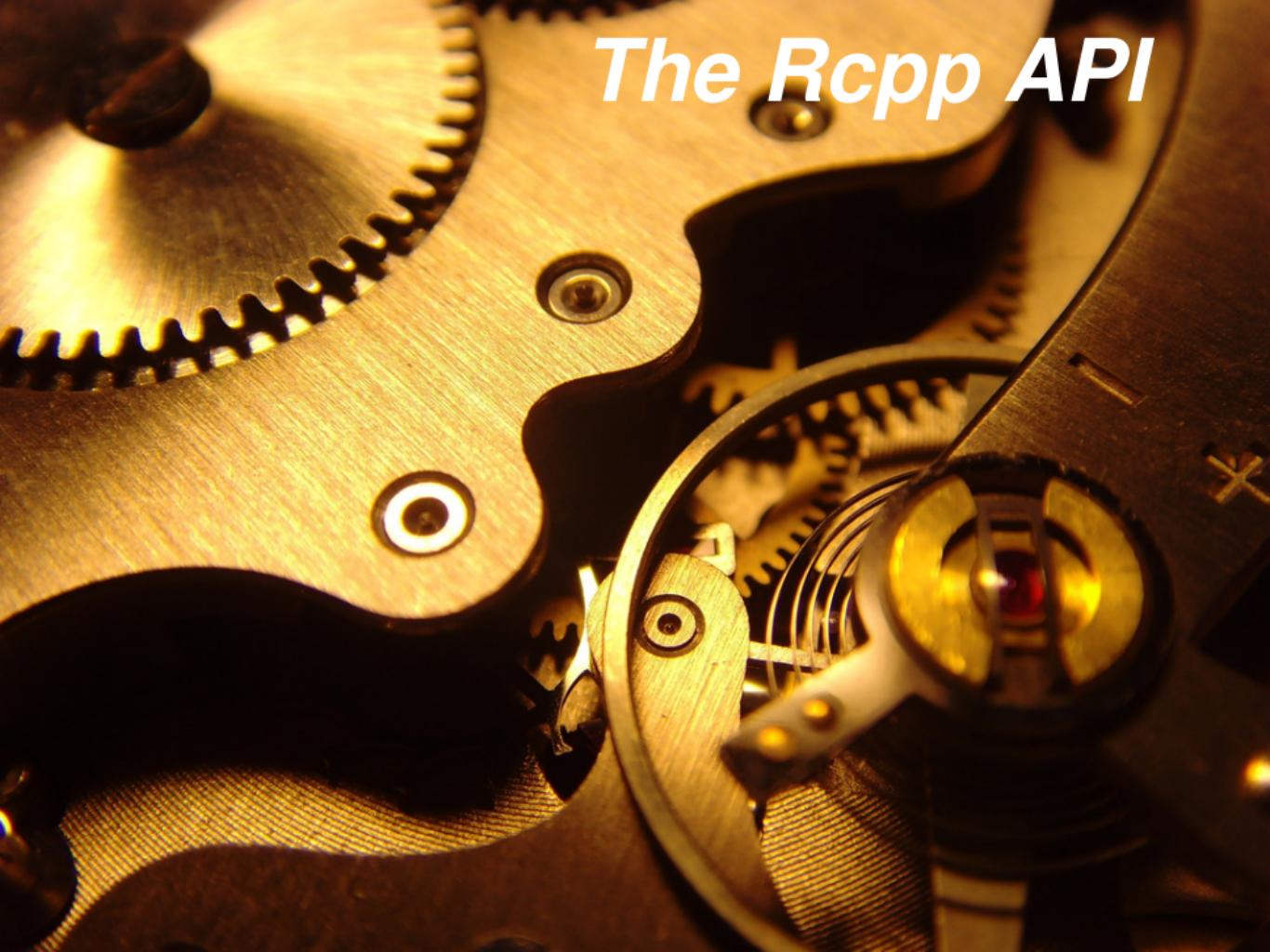
```
> c("foo", "bar")
```

```
#include <R.h>
#include <Rdefines.h>
extern "C" SEXP foobar(){
  SEXP res = PROTECT(allocaVector(STRSXP, 2));
  SET_STRING_ELT(res, 0, mkChar("foo"));
  SET_STRING_ELT(res, 1, mkChar("bar"));
  UNPROTECT(1);
  return res;
}
```

.Call example: calling an R function

```
> eval( call( "rnorm", 3L, 10.0, 20.0 ) )
```

```
#include <R.h>
#include <Rdefines.h>
extern "C" SEXP callback(){
    SEXP call = PROTECT( LCONS( install("rnorm"),
        CONS( ScalarInteger( 3 ),
            CONS( ScalarReal( 10.0 ),
                CONS( ScalarReal( 20.0 ), R_NilValue )
            )
        )
    );
    SEXP res = PROTECT(eval(call, R_GlobalEnv)) ;
    UNPROTECT(2) ;
    return res ;
}
```

A close-up photograph of a mechanical watch movement. The image shows several brass-colored gears with intricate tooth profiles. Some gears have small circular holes with metallic covers, likely rubies or jewels, which catch the light and appear as bright spots. The movement is set against a dark, textured background, possibly the inner case of the watch. The lighting is dramatic, highlighting the metallic surfaces and the precision engineering of the machinery.

The Rcpp API

The Rcpp API

- Encapsulation of R objects (SEXP) into C++ classes:
NumericVector, IntegerVector, ..., Function,
Environment, Language, ...
- Conversion from R to C++ : `as`
- Conversion from C++ to R : `wrap`
- Interoperability with the Standard Template Library (STL)

The Rcpp API : classes

Rcpp class	R <code>typeof</code>
<code>Integer(Vector Matrix)</code>	<code>integer</code> vectors and matrices
<code>Numeric(Vector Matrix)</code>	<code>numeric</code> ...
<code>Logical(Vector Matrix)</code>	<code>logical</code> ...
<code>Character(Vector Matrix)</code>	<code>character</code> ...
<code>Raw(Vector Matrix)</code>	<code>raw</code> ...
<code>Complex(Vector Matrix)</code>	<code>complex</code> ...
<code>List</code>	<code>list</code> (aka generic vectors) ...
<code>Expression(Vector Matrix)</code>	<code>expression</code> ...
<code>Environment</code>	<code>environment</code>
<code>Function</code>	<code>function</code>
<code>XPtr</code>	<code>externalptr</code>
<code>Language</code>	<code>language</code>
<code>S4</code>	<code>S4</code>
...	...

The Rcpp API : example

```
SEXP foo( SEXP x, SEXP y ){
    Rcpp::NumericVector xx(x), yy(y) ;
    int n = xx.size() ;
    Rcpp::NumericVector res( n ) ;
    double x_ = 0.0, y_ = 0.0 ;
    for( int i=0; i<n; i++) {
        x_ = xx[i] ; y_ = yy[i] ;
        if( x_ < y_ ){
            res[i] = x_ * x_ ;
        } else {
            res[i] = -( y_ * y_ ) ;
        }
    }
    return res ;
}
```

The Rcpp API : example

```
using namespace Rcpp ;
SEXP bar(){
    std::vector<double> z(10) ;
    List res = List::create(
        _["foo"] = NumericVector::create(1,2),
        _["bar"] = 3,
        _["bla"] = "yada yada",
        _["blo"] = z
    ) ;
    res.attr("class") = "myclass" ;
    return res ;
}
```

The Rcpp API : example

```
using namespace Rcpp ;
SEXP bar(){
    Language call( "rnorm",
        _["n"] = 100, _["mean"] = 10 ) ;
    NumericVector res = call.eval() ;
    double sum = std::accumulate(
        res.begin(), res.end(),
        0.0 ) ;
    return wrap( sum ) ;
}
```

The Rcpp API : conversion from R to C++

Rcpp:::as<T> handles conversion from SEXP to T.

```
template <typename T> T as( SEXP m_sexp)
    throw(not_compatible) ;
```

T can be:

- primitive type : int, double, bool, long, std::string
- any type that has a constructor taking a SEXP
- ... that specializes the as template
- ... that specializes the Exporter class template
- containers from the STL

more details in the Rcpp-extending vignette.

The Rcpp API : conversion from C++ to R

Rcpp:::wrap<T> handles conversion from T to SEXP.

```
template <typename T>
SEXP wrap( const T& object ) ;
```

T can be:

- primitive type : int, double, bool, long, std::string
- any type that has a operator SEXP
- ... that specializes the `wrap` template
- ... that has a nested type called `iterator` and member functions `begin` and `end`
- containers from the STL `vector<T>`, `list<T>`, `map<string, T>`, etc ... (where T is itself wrappable)

more details in the Rcpp-extending vignette.

The Rcpp API : conversion examples

```
typedef std::vector<double> Vec ;
int x_ = as<int>( x ) ;
double y_ = as<double>( y_ ) ;
VEC z_ = as<VEC>( z_ ) ;

wrap( 1 ) ; //INTSXP
wrap( "foo" ) ; //STRSXP

typedef std::map<std::string,Vec> Map ;
Map foo( 10 ) ;
Vec f1(4) ;
Vec f2(10) ;
foo.insert( "x", f1 ) ;
foo.insert( "y", f2 ) ;
wrap( foo ) ; //named list of numeric vectors
```

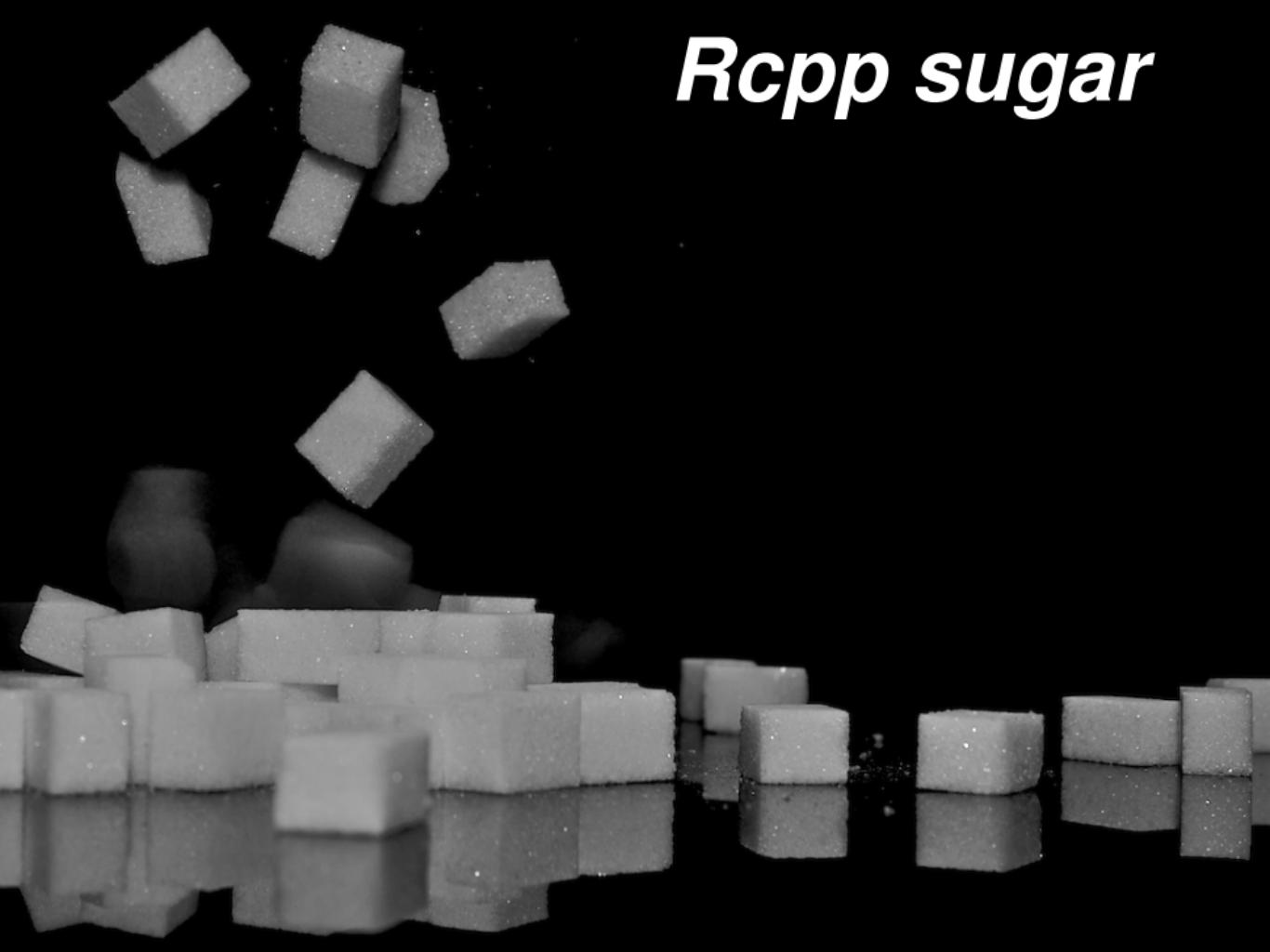
The Rcpp API : *implicit* conversion examples

```
Environment env = ... ;
List list = ... ;
Function rnorm( "rnorm" ) ;

// implicit calls to as
int x = env[ "x" ] ;
double y = list[ "y" ] ;

// implicit calls to wrap
rnorm( 100, _[ "mean" ] = 10 ) ;
env[ "x" ] = 3;
env[ "y" ] = "foo" ;
List::create( 1, "foo", 10.0, false ) ;
```

Rcpp sugar



Sugar : motivation

```
int n = x.size() ;
NumericVector res1( n ) ;
double x_ = 0.0, y_ = 0.0 ;
for( int i=0; i<n; i++) {
    x_ = x[i] ;y_ = y[i] ;
    if( R_IsNA(x_) || R_IsNA(y_) ){
        res1[i] = NA_REAL;
    } else if( x_ < y_ ){
        res1[i] = x_ * x_ ;
    } else {
        res1[i] = -( y_ * y_ ) ;
    }
}
```

Sugar : motivation

We missed the R syntax :

```
> ifelse( x < y, x*x, -(y*y) )
```

Sugar : motivation

We missed the R syntax :

```
> ifelse( x < y, x*x, -(y*y) )
```

sugar brings it into C++

```
SEXP foo( SEXP xx, SEXP yy){  
    NumericVector x(xx), y(yy) ;  
    return ifelse( x < y, x*x, -(y*y) ) ;  
}
```

Sugar : another example

```
double square( double x){  
    return x*x ;  
}  
  
SEXP foo( SEXP xx ){  
    NumericVector x(xx) ;  
    return sapply( x, square ) ;  
}
```

Sugar : contents

- **logical operators:** <, >, <=, >=, ==, !=
- **arithmetic operators:** +, -, *, /
- **functions:** abs, all, any, ceiling, diff, exp, ifelse, is_na, lapply, pmin, pmax, pow, sapply, seq_along, seq_len, sign

Sugar uses Expression Templates (Blitz++, Armadillo, ...) to achieve lazy evaluation of expressions.

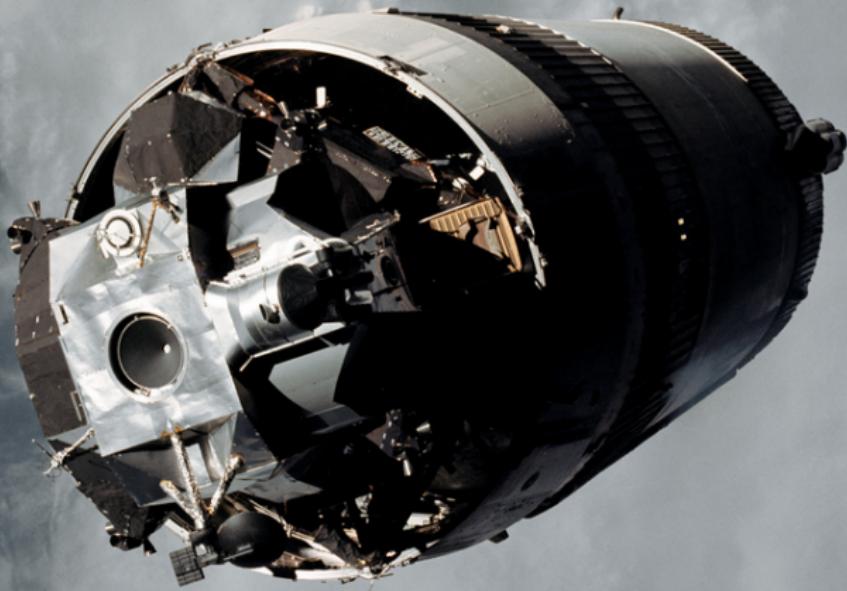
More information in the `Rcpp-sugar` vignette.

Sugar : benchmarks

expression	sugar	R	R / sugar
any (x*y<0)	0.0008771	29.58	33721
ifelse(x<y, x*x, -(y*y))	5.217	35.88	6.879
sapply(x, square)	0.6322	259.4	410.2

Benchmarks performed on fedora 12 / R 2.12.0 (under development) on a 2 years old dell inspiron 1525 laptop.

Rcpp modules



Modules: expose C++ to R

```
const char* hello( const std::string& who ) {
    std::string result( "hello " );
    result += who;
    return result.c_str();
}

RCPP_MODULE(yada) {
    using namespace Rcpp;
    function( "hello", &hello );
}
```

```
> yada <- Module( "yada" )
> yada$hello( "world" )
```

Modules: expose C++ classes to R

```
class World {  
public:  
    World() : msg("hello") {}  
    void set(std::string msg) {  
        this->msg = msg;  
    }  
    std::string greet() {  
        return msg;  
    }  
private:  
    std::string msg;  
};  
  
void clearWorld( World* w) {  
    w->set( "") ;  
}
```

Modules: expose C++ classes to R

```
RCPP_MODULE(yada) {
    using namespace Rcpp ;

    class_<World>( "World" )
        .method( "greet", &World::greet )
        .method( "set", &World::set )
        .method( "clear", &clearWorld )
    ;
}

}
```

Modules: on the R side

```
> w <- new( yada$World )
> w$greet()
[1] "hello"

> w$set( "hello world")
> w$greet()
[1] "hello world"

> w$clear()
> w$greet()
[1] ""
```

Want to learn more ?

- Check the vignettes
- Questions on the Rcpp-devel mailing list
- Hands-on training courses
- Commercial support

Romain François
Dirk Eddelbuettel

romain@r-enthusiasts.com
edd@debian.org