Calling C code from R - an Example
By Bahjat F. Qaqish

This document describes a simple example of how to call C code from within R. It is assumed that all the necessary software (R itself, compiler and other tools) have been installed properly.

Our example is a function to which we pass an array $x_1, \cdots, x_n$ and it computes $y_1, \cdots, y_n$ defined by $y_i = x_i^2$. The example shows

1. How to pass arguments from R to C code
2. How to receive computed values from C code
3. How to compile the C code
4. How to load and run the C code from within R
5. Some rules about writing C code to link to R

The R function will be called vecsqr. An example usage is y = vecsqr(x), assuming a vector x has been defined. The C function will be called vecsqr (other names will do as well) and stored in a file called vecsqr.c. Here is that file:

```
#include <R.h>

void vecsqr(double *y, double *x, int *n_p)
/* y[0:n-1] <- x[0:n-1]^2 */
{
  int i, n = *n_p;

  if (n < 1) Rprintf("vecsqr: error n = %d < 1\n", n);

  for (i=0; i < n; i++) y[i] = x[i]*x[i];
}
```

Notice that there is no main function. Also, R.h must be included. Instead of printf, we use Rprintf to send the output to the R console. Functions that will be called from R must have type void. All arguments are passed by reference including single numbers such as n, the number of elements.

The next step is to compile the C code. This is done with the command

```
R CMD SHLIB vecsqr.c
```

The result is a file named vecsqr.so.

We are almost ready to call our C function from R. We can issue the call directly from R. However, it is generally better and more convenient to write an R function that takes care of dynamically loading the compiled code, translating back and forth between R and C, and making sure that the arguments have the required type and are passed in the proper order. We put that function in a file named vecsqr.r. Here is that file:

```
dyn.load("vecsqr.so")    # load the compiled code
vecsqr = function(x)     # our R interface to the compiled code
{
  n = length(x)
  result  = .C("vecsqr",
        y = double(n),
        as.double(x),
        as.integer(n) )
  result[["y"]]
}
```

The function passes arguments in the proper order (y, x, n), forces the correct type using as.double and as.integer, and allocates the required space for the result y. The R function .C performs the actual call to the C function, and returns a list. The return value from the R function vecsqr is the vector y extracted from the list returned by .C via result[["y"]].

Now, in R, we source the R function

```
source("vecsqr.r")
```

and we are ready to call the C function,

```
a1 = 1:10
a2 = vecsqr(a1)
a4 = vecsqr(a2)
a8 = vecsqr(a4)
cbind(a1, a2, a4, a8)
```

To see the error message generated if n is less than 1 we call the compiled code directly via .C

```
x = 1:10
.C("vecsqr", y = double(length(x)), as.double(x), as.integer(-4))
```

A lot more is possible. Several functions can be defined in vecsqr.c and in vecsqr.r. It is possible to call R functions from C code. Further, the *package* mechanism is especially convenient for distributing code to others. See the *Writing R Extensions* manual for further details.