Package: dotCall64 (via r-universe)

September 8, 2024

Type Package

Title Enhanced Foreign Function Interface Supporting Long Vectors

Version 1.1-1

Date 2023-11-28

Description Provides .C64(), which is an enhanced version of .C() and .Fortran() from the foreign function interface. .C64() supports long vectors, arguments of type 64-bit integer, and provides a mechanism to avoid unnecessary copies of read-only and write-only arguments. This makes it a convenient and fast interface to C/C++ and Fortran code.

License GPL (>= 2)

URL https://git.math.uzh.ch/reinhard.furrer/dotCall64

BugReports https://git.math.uzh.ch/reinhard.furrer/dotCall64/-/issues

Depends R (>= 3.1)

Suggests microbenchmark, RhpcBLASctl, RColorBrewer, roxygen2, spam, testthat,

Collate 'vector_dc.R' 'dotCall64.R' 'zzz.R'

RoxygenNote 7.1.1

NeedsCompilation yes

Author Kaspar Moesinger [aut], Florian Gerber [aut] (<https://orcid.org/0000-0001-8545-5263>), Reinhard Furrer [cre, ctb] (<https://orcid.org/0000-0002-6319-2332>)

Maintainer Reinhard Furrer <reinhard.furrer@math.uzh.ch>

Date/Publication 2023-11-28 11:30:02 UTC

Repository https://reinhardfurrer.r-universe.dev

RemoteUrl https://github.com/cran/dotCall64

RemoteRef HEAD

RemoteSha baa4573b25984cb7df24df8dbac6eafa10fc60bb

Contents

---

dotCall64 *dotCall64 - Extended Foreign Function Interface*

Description

`.C64` can be used to call compiled and loaded C/C++ functions and Fortran subroutines. `.C64` is similar to `.C` and `.Fortran`, but

1. supports long vectors, i.e., vectors with more than `2^31-1` elements
2. does the necessary castings to expose the R representation of "64-bit integers" (numeric vectors) to 64-bit integer arguments of the compiled function. The latter are int64_t in C code and integer (kind = 8) in Fortran code
3. provides a mechanism the control duplication of the R objects exposed to the compiled code
4. checks if the provided R objects are of the expected types and coerces them if necessary

Compared to `.C`, `.C64` has the additional arguments `SIGNATURE`, `INTENT` and `VERBOSE`. `SIGNATURE` specifies the types of the arguments of the compiled function. `INTENT` indicates whether the compiled function "reads", "writes", or "read and writes" to the R objects passed to the compiled function. This information is then used to duplicate R objects if and only if necessary.

Usage

```
.C64(
  .NAME,
  SIGNATURE,
  ...,
  INTENT = NULL,
  NAOK = FALSE,
  PACKAGE = "",
  VERBOSE = getOption("dotCall64.verbose")
)
```

Arguments

`.NAME`	character vector of length 1. Name of the compiled function to be called.
`SIGNATURE`	character vector of the same length as the number of arguments of the compiled function. Accepted strings are `"double"`, `"integer"`, and `"int64"`. They describe the signature of each argument of the compiled function.
`...`	arguments passed to the compiled function. One R object for each argument. Up to 65 arguments are supported.

INTENT — character vector of the same length as the number of arguments of the compiled function. Accepted strings are "rw", "r", and "w", which indicate whether the intent of the argument is "read and write", "read", or "write", respectively. If the INTENT of an argument is "rw", the R object is copied and the compiled function receives a pointer to that copy. If the INTENT of an R object is "r", the compiled function receives a pointer to the R object itself. While this avoids copying and hence is more efficient in terms of speed and memory usage, it is absolutely necessary that the compiled function does not alter the object, since this corrupts the R object in the current R session. When the INTENT is "w", the corresponding input argument can be specified with the function vector_dc or its shortcuts integer_dc and numeric_dc. This avoids copying the passed R objects and hence is more efficient in terms of speed and memory usage. By default, all arguments have INTENT "rw".

NAOK — logical vector of length 1. If FALSE (default), the presence of NA, NaN, and Inf in the R objects passed through ... results in an error. If TRUE, NA, NaN, and Inf values are passed to the compiled function. The used time to check arguments (if FALSE) is considerable for large vectors.

PACKAGE — character vector of length 1. Specifies where to search for the function given in .NAME. This is intended to add safety for packages, which can use this argument to ensure that no other package can override their external symbols, and also speeds up the search.

VERBOSE — numeric vector of length 1. If 0, no warnings are printed. If 1, warnings are printed, which help to improve the performance of the call. If 2, additional debug information is given as warnings. The default value can be changed via the dotCall64.verbose option, which is set to 0 by default.

Value

list of R objects similar to the list of arguments specified as ... arguments. The objects of the list reflect the changes made by the compiled C or Fortran function.

References

F. Gerber, K. Moesinger, R. Furrer (2018), dotCall64: An R package providing an efficient interface to compiled C, C++, and Fortran code supporting long vectors, SoftwareX 7, 217-221, https://doi.org/10.1016/j.softx.2018.06.002.

F. Gerber, K. Moesinger, and R. Furrer (2017), Extending R packages to support 64-bit compiled code: An illustration with spam64 and GIMMS NDVI3g data, Computer & Geoscience 104, 109-119, https://doi.org/10.1016/j.cageo.2016.11.015.

Examples

```
## Consider the following C function, which is included
## in the dotCall64 package:
## void get_c(double *input, int *index, double *output) {
##     output[0] = input[index[0] - 1];
## }
##
```

```
## We can use .C64() to call get_c() from R:
.C64("get_c", SIGNATURE = c("double", "integer", "double"),
     input = 1:10, index = 9, output = double(1))$output

## Not run:
## 'input' can be a long vector
x_long <- double(2^31) ## requires 16 GB RAM
x_long[9] <- 9; x_long[2^31] <- -1
.C64("get_c", SIGNATURE = c("double", "integer", "double"),
     input = x_long, index = 9, output = double(1))$output

## Since 'index' is of type 'signed int' (a 32-bit integer),
## it can only address the first 2^31-1 elements of 'input'.
## To also address elements beyond 2^31-1, we change the
## definition of the C function as follows:
## #include <stdint.h>  //  for int64_t
## void get64_c(double *input, int64_t *index, double *output) {
##     output[0] = input[index[0] - 1];
## }

## Now, we can use .C64() to call get64_c() from R.
.C64("get64_c", SIGNATURE = c("double", "int64", "double"),
     input = x_long, index = 2^31, output = double(1))$output
## Note that 2^31 is of type double and .C64() casts it into an
## int64_t type before calling the C function get64_c().

## The performance of the previous call can be improved by
## setting additional arguments:
.C64("get64_c", SIGNATURE = c("double", "int64", "double"),
     x = x_long, i = 2^31, r = numeric_dc(1), INTENT = c("r", "r", "w"),
     NAOK = TRUE, PACKAGE = "dotCall64", VERBOSE = 0)$r


## Consider the same function defined in Fortran:
##      subroutine get64_f(input, index, output)
##      double precision :: input(*), output(*)
##      integer (kind = 8) :: index  ! specific to GFortran
##      output(1) = input(index)
##      end

## The function is provided in dotCall64 and can be called with
.C64("get64_f", SIGNATURE = c("double", "int64", "double"),
     input = x_long, index = 2^31, output = double(1))$output

## End(Not run)
```

---

vector_dc *Allocate vectors in .C64()*

Description

vector_dc and its shortcuts numeric_dc and integer_dc are helper functions used in calls to .C64. They return an R object of class c("vector_dc", "list"), which contains information on the type and length of the vector to allocate. Using vector_dc together with INTENT = "w" argument of .C64 leads to performance gains by avoiding unnecessary castings and copies.

Usage

```
vector_dc(mode = "logical", length = 0L)

numeric_dc(length = 0)

integer_dc(length = 0)
```

Arguments

mode	character vector of length 1. Storage mode of the vector.
length	numeric vector of length 1. Length of the vector.

Value

object of class vector_dc and list.

Examples

```
vector_dc("integer", 20)
```

Index