Package 'callme'

May 8, 2026

Type Package

Title Easily Compile and Call Inline 'C' Functions

Version 0.1.11

Maintainer Mike Cheng <mikefc@coolbutuseless.com>

Description Compile inline 'C' code and easily call with automatically generated wrapper functions. By allowing user-defined headers and compilation flags (preprocessor, compiler and linking flags) the user can configure optimization options and linking to third party libraries. Multiple functions may be defined in a single block of code - which may be defined in a string or a path to a source file.

License MIT + file LICENSE

URL https://github.com/coolbutuseless/callme, https://coolbutuseless.github.io/package/callme/

BugReports https://github.com/coolbutuseless/callme/issues

Imports methods

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

SystemRequirements Rtools (suitable for the installed R) for Windows, Xcode for Mac

NeedsCompilation no

Author Mike Cheng [aut, cre, cph]

Repository CRAN

Date/Publication 2024-11-28 05:30:01 UTC

Contents

`compile` *Compile C code and create wrapper functions to call from R*

Description

This function uses the `R CMD SHLIB` process to compile C code into a linked library. This library is then loaded, and appropriate functions created in R to call into this library. See also: `?SHLIB`

Usage

```
compile(
  code,
  CFLAGS = NULL,
  PKG_CPPFLAGS = NULL,
  PKG_LIBS = NULL,
  env = parent.frame(),
  overwrite = "callme",
  verbosity = 0,
  invisible = FALSE
)
```

Arguments

`code`	C code following the `.Call()` conventions, or a filename containing this code. This code must also include any `#include` statements - include `<R.h>` and `<Rinternals.h>` at the very least.
`CFLAGS`	character string of flags for the C compiler. e.g. "-O3" Default: NULL. If specified this value will *replace* the default `CFLAGS` R would normally use. To see these default flags use `maketools::cc_info()$flags`.
`PKG_CPPFLAGS`	character string of flags for the C pre-processor. Flags such as "-I", "-D" and "-U" go here. Default: NULL e.g. `PKG_CPPFLAGS = "-I/opt/homebrew/include"` to add the include path for homebrew to the compilation step.
`PKG_LIBS`	character string of flags for linking. "-L" and "-l" flags go here. Default: NULL. e.g. `PKG_LIBS = "-L/opt/homebrew/lib -lzstd"` to include the homebrew libraries in the linker search path and to link to the `zstd` library installed there.
`env`	environment into which to assign the R wrapper functions. Default: `parent.frame()`. If `NULL` then no assignment takes place and the (invisible) return value should be assigned to a variable to access the compiled code.
`overwrite`	Which existing variables can be overwritten when wrapper functions are created in the given environment? An error will be raised if the name of the wrapper function already exists in the environment and permission has not been given to overwrite.
`verbosity`	Level of output: Default: 0. Max level: 4

`invisible` Should the R wrapper function return the result invisibly? Default: FALSE. Set this to `TRUE` if the code is only run for its side-effect e.g. just printing data and not returning anything.

"callme" (Default) Only functions created by this package can be overwritten

"all" All objects can be overwritten

"functions" Only functions can be overwritten

"none" No existing objects can be overwritten

Value

Invisibly returns a named list of R functions. Each R function calls to the equivalent C function. If `env` is specified, then these wrapper functions are assigned in the given environment.

Examples

```
code <- "
#include <R.h>
#include <Rinternals.h>

// Add 2 numbers
SEXP add(SEXP val1, SEXP val2) {
  return ScalarReal(asReal(val1) + asReal(val2));
}

// Multiply 2 numbers
SEXP mul(SEXP val1, SEXP val2) {
  return ScalarReal(asReal(val1) * asReal(val2));
}

// sqrt elements in a vector
SEXP new_sqrt(SEXP vec) {
  SEXP res = PROTECT(allocVector(REALSXP, length(vec)));
  double *res_ptr = REAL(res);
  double *vec_ptr = REAL(vec);
  for (int i = 0; i < length(vec); i++) {
    res_ptr[i] = sqrt(vec_ptr[i]);
  }

  UNPROTECT(1);
  return res;
}
"

# compile the code and load into R
compile(code)

# Call the functions
add(99.5, 0.5)
mul(99.5, 0.5)
new_sqrt(c(1, 10, 100, 1000))
```

Index