Package 'cpp11'

October 14, 2020

Title A C++11 Interface for R's C Interface

Version 0.2.3

Description Provides a header only, C++11 interface to R's C interface. Compared to other approaches 'cpp11' strives to be safe against long jumps from the C API as well as C++ exceptions, conform to normal R function semantics and supports interaction with 'ALTREP' vectors.

License MIT + file LICENSE

URL https://github.com/r-lib/cpp11

BugReports https://github.com/r-lib/cpp11/issues

Suggests bench, brio, callr, cli, covr, decor, desc, ggplot2, glue, knitr, lobstr, mockery, progress, rmarkdown, scales, testthat, tibble, utils, vctrs, withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/cpp11/cpp_register brio, cli, decor, desc, glue, tibble, vctrs

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

SystemRequirements C++11

NeedsCompilation no

Author Jim Hester [aut, cre] (<https://orcid.org/0000-0002-2739-7082>), Romain François [ctb], Benjamin Kietzman [ctb], RStudio [cph, fnd]

Maintainer Jim Hester <jim.hester@rstudio.com>

Repository CRAN

Date/Publication 2020-10-14 05:30:11 UTC

R topics documented:

`cpp_register` *Generates wrappers for registered C++ functions*

Description

Functions decorated with [[cpp11::register]] in files ending in `.cc`, `.cpp`, `.h` or `.hpp` will be wrapped in generated code and registered to be called from R.

Usage

```
cpp_register(path = ".", quiet = FALSE)
```

Arguments

`path`	The path to the package root directory
`quiet`	If `TRUE` suppresses output from this function

Details

In order to use `cpp_register()` the `cli`, `decor`, `desc`, `glue`, `tibble` and `vctrs` packages must also be installed.

Value

The paths to the generated R and C++ source files (in that order).

Examples

```
# create a minimal package
dir <- tempfile()
dir.create(dir)

writeLines("Package: testPkg", file.path(dir, "DESCRIPTION"))
writeLines("useDynLib(testPkg, .registration = TRUE)", file.path(dir, "NAMESPACE"))

# create a C++ file with a decorated function
dir.create(file.path(dir, "src"))
writeLines("[[cpp11::register]] int one() { return 1; }", file.path(dir, "src", "one.cpp"))

# register the functions in the package
cpp_register(dir)
```

```
# Files generated by registration
file.exists(file.path(dir, "R", "cpp11.R"))
file.exists(file.path(dir, "src", "cpp11.cpp"))

# cleanup
unlink(dir, recursive = TRUE)
```

---

cpp_source *Compile C++ code*

Description

cpp_source() compiles and loads a single C++ file for use in R. cpp_function() compiles and loads a single function for use in R. cpp_eval() evaluates a single C++ expression and returns the result.

Usage

```
cpp_source(file, code = NULL, env = parent.frame(), clean = TRUE, quiet = TRUE)

cpp_function(code, env = parent.frame(), clean = TRUE, quiet = TRUE)

cpp_eval(code, env = parent.frame(), clean = TRUE, quiet = TRUE)
```

Arguments

file	A file containing C++ code to compile
code	If non-null, the C++ code to compile
env	The R environment where the R wrapping functions should be defined.
clean	If TRUE, cleanup the files after sourcing
quiet	If 'TRUE', do not show compiler output

Details

Within C++ code you can use [[cpp11::linking_to("pkgxyz")]] to link to external packages. This is equivalent to putting those packages in the LinkingTo field in a package DESCRIPTION.

Value

For cpp_source() and [cpp_function()] the results of dyn.load() (invisibly). For [cpp_eval()] the results of the evaluated expression.

Examples

```
## Not run:
cpp_source(
  code = '#include "cpp11/integers.hpp"

  [[cpp11::register]]
  int num_odd(cpp11::integers x) {
    int total = 0;
    for (int val : x) {
      if ((val % 2) == 1) {
        ++total;
      }
    }
    return total;
  }
  ')

num_odd(as.integer(c(1:10, 15, 23)))

if (require("progress")) {

cpp_source(
  code = '
#include <cpp11/R.hpp>
#include <RProgress.h>

[[cpp11::linking_to("progress")]]

[[cpp11::register]] void
show_progress() {
  RProgress::RProgress pb("Downloading [:bar] ETA: :eta");

  pb.tick(0);
  for (int i = 0; i < 100; i++) {
    usleep(2.0 / 100 * 1000000);
    pb.tick();
  }
}
')

show_progress()
}

## End(Not run)
```

---

cpp_vendor *Vendor the cpp11 dependency*

Description

Vendoring is the act of making your own copy of the 3rd party packages your project is using. It is often used in the go language community.

Usage

```
cpp_vendor(path = ".")
```

Arguments

path	The path to the package root directory

Details

This function vendors cpp11 into your package by copying the cpp11 headers into the `inst/include` folder of your package and adding 'cpp11 version: XYZ' to the top of the files, where XYZ is the version of cpp11 currently installed on your machine.

If you choose to vendor the headers you should *remove* `LinkingTo: cpp11` from your DESCRIPTION.

Note: vendoring places the responsibility of updating the code on **you**. Bugfixes and new features in cpp11 will not be available for your code until you run `vector_cpp11()` again.

Value

The file path to the vendored code (invisibly).

Examples

```
# create a new directory
dir <- tempfile()
dir.create(dir)

# vendor the cpp11 headers into the directory
cpp_vendor(dir)

list.files(file.path(dir, "inst", "include", "cpp11"))

# cleanup
unlink(dir, recursive = TRUE)
```

Index