Package: cppally (via r-universe)

July 2, 2026

Title A 'C++20' API for R

Version 1.0.0

Maintainer Nick Christofides <nick.christofides.r@gmail.com>

Description A header-only 'C++20' API for manipulating R data structures from 'C++'. Provides 'C++20' concepts specific to R, custom scalar and vector classes with built-in NA handling, automatic object protection, 'SIMD' (single-instruction-multiple-data), parallelisation, and a streamlined system for registering 'C++' functions, including templates, to R. Full API reference and documentation are available at <https://nicchr.github.io/cppally/>.

License MIT + file LICENSE

URL https://nicchr.github.io/cppally/

BugReports https://github.com/NicChr/cppally/issues/

Depends R (>= 4.5.0)

Suggests bench, bit64, brio, callr, cli, cpp11, decor, desc, devtools, fs, glue, knitr, pkgload, purrr, readr, rmarkdown, roxygen2, rstudioapi, stringr, testthat (>= 3.0.0), usethis, vctrs, withr

VignetteBuilder knitr

Config/Needs/cppally/cpp_register brio, cli, decor, desc, glue, purrr, readr, stringr, vctrs, withr

Config/testthat/edition 3

Encoding UTF-8

SystemRequirements C++20

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Nick Christofides [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-9743-7342>), Martin Leitner-Ankerl [cph] (Author of bundled ankerl::unordered_dense library), Malte Skarupke [cph] (Author of bundled ska_sort library), Posit Software, PBC [cph] (SEXP protection mechanism in r_protect.h inspired by cpp11)

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

Date/Publication 2026-07-02 11:20:02 UTC

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

RemoteRef HEAD

RemoteSha a82b1cdad21d31846f32e473583c84a028fcc60f

Contents

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

Description

Register C++ functions to be callable from R. C++ functions decorated with `[[cppally::register]]` will be registered (including template functions).

Usage

```
cpp_register(
  path = ".",
  quiet = !is_interactive(),
  extension = c(".cpp", ".cc")
)
```

Arguments

`path`	Path to package root directory.
`quiet`	If `TRUE` suppresses output from this function.
`extension`	The file extension to use for the generated src/cppally file. Options are either '.cpp' (the default) or '.cc'.

Value

The paths to the generated R and C++ source files.

See Also

cpp_source

`cpp_source` *Compile C++20 code*

Description

cpp11-style helpers to compile cppally code outside of a cppally-linked package context.

`cpp_source()` compiles and loads a single C++ file for use in R, either from an expression or a cpp file. This may include multiple C++ functions.

`cpp_eval()` evaluates a single C++ expression and returns the result. For example `cpp_eval('get_threads()')` will run the C++ function `cppally::get_threads()` and return the number of OMP threads currently set for use. For expressions no return result, the call is evaluated and returns `NULL` invisibly.

Usage

```
cpp_source(
  file,
  code = NULL,
  env = parent.frame(),
  clean = TRUE,
  quiet = TRUE,
  debug = FALSE,
  preserve_altrep = FALSE,
  check_factors = FALSE,
  check_data_frames = FALSE,
  copy_on_modify = FALSE,
  cxx_std = Sys.getenv("CXX_STD", "CXX20"),
  dir = tempfile()
)

cpp_eval(
  code,
  env = curr_env(),
  clean = TRUE,
  quiet = TRUE,
  debug = FALSE,
  preserve_altrep = FALSE,
  check_factors = FALSE,
  check_data_frames = FALSE,
  copy_on_modify = FALSE,
  simplify = TRUE,
  cxx_std = Sys.getenv("CXX_STD", "CXX20")
)
```

Arguments

file	C++ file.
code	For cpp_source() - If file is NULL then a string of C++ code to compile. This can include the contents of a cpp file which can contain multiple [[cppally::register]] tagged functions. For cpp_eval - This can be a character vector of single-line expressions.
env	Environment where R functions should be defined.
clean	Should files be cleaned up after sourcing? Default is TRUE.
quiet	Should compiler output be suppressed? Default is TRUE.
debug	Should C++ code be compiled in a debug build? Default is FALSE.
preserve_altrep	Should ALTREP vectors be preserved by avoiding materialisation where possible? Default is FALSE.
check_factors	Should factor levels be validated when using r_factors objects? Default is FALSE. When TRUE, factor levels are checked once on r_factors construction to ensure they are valid, reducing the chance of R crashing when passing factors with invalid levels.
check_data_frames	Should data frames be validated when constructing r_df objects from SEXP? Default is FALSE.
copy_on_modify	Should copy-on-modify be used everywhere? Default is FALSE.
cxx_std	C++ standard to use. Should be >= C++20.
dir	Directory to store the source files. The default is a temporary directory via tempfile() which is removed when clean = TRUE.
simplify	Applies to cpp_eval. A list of results is returned unless length(code) == 1 and simplify = TRUE.

Value

cpp_source() invisibly compiles the C++ code and registers the [[cppally::register]] tagged functions to R.
cpp_eval() returns the results of the evaluated C++ expressions.

See Also

cpp_register

Examples

```
library(cppally)
library(bit64)

cpp_eval('print("hello world!")')

# Default values of all cppally scalars
cpp_eval(c(
```

```
  'r_lgl()',
  'r_int()',
  'r_dbl()',
  'r_int64()',
  'r_str()',
  'r_raw()',
  'r_cplx()',
  'r_date()',
  'r_psxct()'
))

cpp_source(code = '
  #include <cppally.hpp>
  using namespace cppally;

  [[cppally::register]]
  r_dbl add(r_dbl x, r_dbl y){
    return x + y;
  }
', debug = TRUE)
add(1, 2)
add(2, NA)

### ALTREP ###

# cppally also supports lazy ALTREP materialisation as an opt-in feature.
# To opt-in, set `preserve_altrep = TRUE`

cpp_source(
  code = '
  #include <cppally.hpp>
  using namespace cppally;

  [[cppally::register]]
  r_int last_altrep_unaware(r_vec<r_int> x){
    r_int out;
    r_size_t n = x.length();

    if (n > 0){
      out = x.get(n - 1);
    }
    return out;
  }
', debug = TRUE
)

cpp_source(
  code = '
  #include <cppally.hpp>
  using namespace cppally;

  [[cppally::register]]
  r_int last_altrep_aware(r_vec<r_int> x){
```

```
    r_int out;
    r_size_t n = x.length();

    if (n > 0){
      out = x.get(n - 1);
    }
    return out;
  }
', debug = TRUE,
  preserve_altrep = TRUE
)

library(bench)
mark(last_altrep_aware(1:10^5)) # No materialisation
mark(last_altrep_unaware(1:10^5)) # Materialises full vector

### Copy-on-modify ###

# cppally supports copy-on-modify as an opt-in feature
# It is disabled by default because it incurs a major performance penalty
# and has been deemed not worth it even for the safety benefits
# That being said, if you prefer absolute safety over speed then you can
# enable it globally via `cppally::use_copy_on_modify()` or
# via the arg `copy_on_modify` if  using `cpp_source()`

cpp_source(
  code = '
  #include <cppally.hpp>
  using namespace cppally;

  [[cppally::register]]
  r_vec<r_int> reverse(r_vec<r_int> x){
    x.rev(); // in-place reverse
    return x;
  }
', copy_on_modify = TRUE
)

x <- c(1L, 2L, 3L)
reverse(x)
x # x was preserved and not updated by reference (as expected)

x <- sample.int(10^5)
mark(reverse(x)) # Memory allocated, therefore x was copied before reversing

# The cppally preferred approach is to allocate a fresh vector or copy the
# existing vector
cpp_source(
  code = '
  #include <cppally.hpp>
  using namespace cppally;

  [[cppally::register]]
```

```
  r_vec<r_int> cppally_reverse(r_vec<r_int> x){
    r_vec<r_int> out = shallow_copy(x);
    out.rev();
    return out;
  }
', copy_on_modify = FALSE
)

mark(
  r_reverse = rev(x),
  cppally_copy_on_modify_reverse = reverse(x),
  cppally_no_copy_on_modify_reverse = cppally_reverse(x)
)
```

`document` *A wrapper around* `devtools::document()` *to support cppally package development*

Description

A wrapper around `devtools::document()` to support cppally package development

Usage

```
document(pkg = ".", roclets = NULL, quiet = FALSE)
```

Arguments

`pkg`	See `?devtools::document`
`roclets`	See `?devtools::document`
`quiet`	See `?devtools::document`

Value

Invisibly updates roxygen documentation, compiles C++ code and exports cppally tagged functions to R.

load_all *A wrapper around* devtools::load_all() *specifically for cppally*

Description

A wrapper around devtools::load_all() specifically for cppally

Usage

```
load_all(path = ".", debug = FALSE, ...)
```

Arguments

path	Path to package.
debug	Should package be built without optimisations? Default is FALSE which builds with optimisations.
...	Further arguments passed on to pkgload::load_all()

Value

Invisibly registers cppally tagged functions and compiles C++ code.

use_check_data_frames *Adds the* CPPALLY_CHECK_DATA_FRAMES *flag to Makevars*

Description

Adds a flag to Makevars which enables stricter validation on data frames at the point of r_df construction. This ensures that column lengths are always valid, avoiding potential R crashes downstream.

The default behaviour is NOT to validate column lengths, enabling faster r_df creating from SEXP.

Usage

```
use_check_data_frames()
```

Value

Invisibly adds the CPPALLY_CHECK_DATA_FRAMES flag to Makevars.

use_check_factors *Adds the* CPPALLY_CHECK_FACTORS *flag to Makevars*

Description

Adds a flag to Makevars which enables stricter validation on factor levels at the point of `r_factors` construction. This avoids creating `r_factors` objects with invalid levels and avoiding potential R crashes.

The default behaviour is NOT to validate factor levels, which is naturally faster when calling C++ functions that take `r_factors` inputs.

Usage

```
use_check_factors()
```

Value

Invisibly adds the `CPPALLY_CHECK_FACTORS` flag to Makevars.

use_copy_on_modify *Adds the* CPPALLY_COPY_ON_MODIFY *flag to Makevars*

Description

Adds a flag to Makevars which enables copy-on-modify behaviour like in R. This ensures you can never modify object B's data by modifying object A (unless you do it manually via `r_vec::data()`).

The default behaviour is that `r_vec::set()` always modifies-in-place with no checks to shared or referenced objects. This default behaviour is generally much faster.

Parallelisation:

Adding this global flag effectively kills most parallelisation. To achieve copy-on-modify we have to check whether the `r_sexp` object being modified is referenced anywhere else at each and every call to `set()`. This check (via `r_sexp::is_exclusive()`) is not thread-safe and therefore most cppally parallelisation is disabled once CPPALLY_COPY_ON_MODIFY is set.

Usage

```
use_copy_on_modify()
```

Value

Invisibly adds the `CPPALLY_COPY_ON_MODIFY` flag to Makevars.

use_cppally *Helper for developing packages with cppally*

Description

usethis style helper to add the necessary setup to a new package to help users get started with writing C++ code.

Usage

```
use_cppally()
```

Value

Invisibly sets up the necessary conditions for developing a package with cppally.

use_preserve_altrep_flag *Adds the CPPALLY_PRESERVE_ALTREP flag to Makevars*

Description

Adds a flag to Makevars which enables lazy materialisation of ALTREP vectors.

Usage

```
use_preserve_altrep_flag()
```

Value

Invisibly adds the CPPALLY_PRESERVE_ALTREP flag to Makevars.

Index