Package 'cpp4r'

November 15, 2026

Type Package

Title Header-Only 'C++' and 'R' Interface

Description Provides a header only, 'C++' interface to 'R' with enhancements over 'cpp11'. Enforces copy-on-write semantics consistent with 'R' behavior. Offers native support for ALTREP objects, 'UTF-8' string handling, modern 'C++' features and idioms, and reduced memory requirements. Allows for vendoring, making it useful for restricted environments. Compared to 'cpp11', it adds support for converting 'C++' maps to 'R' lists, 'Roxygen' documentation directly in 'C++' code, proper handling of matrix attributes, support for nullable external pointers, bidirectional copy of complex number types, flexibility in type conversions, use of nullable pointers, and various performance optimizations.

Version 1.3.1

License MIT + file LICENSE

URL https://cpp4r.org, https://github.com/pachadotdev/cpp4r

BugReports https://github.com/pachadotdev/cpp4r/issues

Depends R (>= 4.0.0)

Suggests tinytest, rbibutils, litedown

Encoding UTF-8

VignetteBuilder litedown

NeedsCompilation no

Author Mauricio Vargas Sepulveda [aut, cre] (ORCID: <https://orcid.org/0000-0003-1017-7574>), Posit Software, PBC [aut] (Original cpp11 package)

Maintainer Mauricio Vargas Sepulveda <m.vargas.sepulveda@gmail.com>

Repository CRAN

Date/Publication 2026-11-15 17:10:02 UTC

Contents

cpp4r-package 2
pkg_template 2
register 3
unvendor 4
vendor 5

Index **6**

`cpp4r-package` *Title: Header-Only 'C++' and 'R' Interface*

Description

Provides a header only, 'C++' interface to 'R' with enhancements over 'cpp11'. Enforces copy-on-write semantics consistent with 'R' behavior. Offers native support for ALTREP objects, 'UTF-8' string handling, modern 'C++' features and idioms, and reduced memory requirements. Allows for vendoring, making it useful for restricted environments. Compared to 'cpp11', it adds support for converting 'C++' maps to 'R' lists, 'Roxygen' documentation directly in 'C++' code, proper handling of matrix attributes, support for nullable external pointers, bidirectional copy of complex number types, flexibility in type conversions, use of nullable pointers, and various performance optimizations.

`pkg_template` *Start a new project with the cpp4r package template*

Description

Copies a package template into a new directory. The template includes a DESCRIPTION file, a minimal R/ directory and placeholders with instructions. You can then edit these files to customize your new package.

Usage

```
pkg_template(path = NULL, pkgname = NULL)
```

Arguments

`path`	Path to the new project
`pkgname`	Name of the new package

Value

The file path to the copied template (invisibly).

Examples

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

# copy the package template into the directory
pkg_template(dir, "mynewpkg")
```

---

register *Generates wrappers for registered C++ functions*

Description

Functions decorated with `[[cpp4r::register]]` in files ending in '.cc', '.cpp', '.h' or '.hpp' will be wrapped in generated code and registered to be called from R. Decorating a function only makes it callable via '.Call()'; it does not export or document it. See the "Package skeleton" vignette for how to export a registered function. If a package contains only header files, those headers are included in the generated translation unit so no additional C++ source file is required.

Usage

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

Arguments

`path`	The path to the package root directory. The default is 'NULL',
`quiet`	If 'TRUE' suppresses output from this function
`extension`	The file extension to use for the generated src/cpp4r file. '.cpp' by default, but '.cc' is also supported.

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("[[cpp4r::register]] int one() { return 1; }", file.path(dir, "src", "one.cpp"))

# register the functions in the package
```

```
register(dir)

# Files generated by registration
file.exists(file.path(dir, "R", "cpp4r.R"))
file.exists(file.path(dir, "src", "cpp4r.cpp"))

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

---

unvendor *Unvendor the cpp4r headers*

Description

This function removes the vendored cpp4r headers from your package by automatically finding the vendored headers.

Usage

```
unvendor(path = NULL)
```

Arguments

path	The directory with the vendored headers. It is recommended to use '"./src/vendor"'. The default is 'NULL'.

Value

The path to the unvendored code (invisibly).

Examples

```
# create a new directory
dir <- paste0(tempdir(), "/", gsub("\\s+|[[:punct:]]", "", Sys.time()))
dir.create(dir, recursive = TRUE)

# vendor the cpp4r headers into the directory
vendor(dir)

# unvendor the cpp4r headers from the directory
unvendor(dir)

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

vendor *Vendor the cpp4r headers*

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.

This function vendors cpp4r into your package by copying the cpp4r headers into the ‘inst/include‘ folder of your package and adding ’cpp4r version: XYZ’ to the top of the files, where XYZ is the version of cpp4r currently installed on your machine.

Vendoring places the responsibility of updating the code on you. Bugfixes and new features in cpp4r will not be available for your code until you run `cpp_vendor()` again.

Usage

```
vendor(path = NULL)
```

Arguments

path	The directory with the vendored headers. It is recommended to use ‘"./src/vendor"‘. The default is ‘NULL‘.

Value

The path to the vendored code (invisibly).

Examples

```
# create a new directory
dir <- paste0(tempdir(), "/", gsub("\\s+|[[:punct:]]", "", Sys.time()))
dir.create(dir, recursive = TRUE, showWarnings = FALSE)

# vendor the cpp4r headers into the directory
vendor(dir)

list.files(dir, recursive = TRUE)

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

Index

cpp4r-package, 2

pkg_template, 2

register, 3

unvendor, 4

vendor, 5