Package 'rextendr'

June 19, 2025

Title Call Rust Code from R using the 'extendr' Crate

Version 0.4.1

Description Provides functions to compile and load Rust code from R, similar to how 'Rcpp' or 'cpp11' allow easy interfacing with C++ code. Also provides helper functions to create R packages that use Rust code. Under the hood, the Rust crate 'extendr' is used to do all the heavy lifting.

License MIT + file LICENSE

URL https://extendr.github.io/rextendr/, https://github.com/extendr/rextendr

BugReports https://github.com/extendr/rextendr/issues

Depends R (>= 4.1)

Imports brio, callr, cli, desc, dplyr, glue (>= 1.7.0), jsonlite, pkgbuild (>= 1.4.0), processx, rlang (>= 1.0.5), rprojroot, stringi, vctrs, withr

Suggests devtools, rcmdcheck, knitr, lintr, rmarkdown, testthat (>= 3.1.7), usethis

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements Rust 'cargo'; the crate 'libR-sys' must compile without error

NeedsCompilation no

Author Claus O. Wilke [aut] (ORCID: <https://orcid.org/0000-0002-7470-9261>), Andy Thomason [aut], Mossa M. Reimert [aut], Ilia Kosenkov [aut, cre] (ORCID: <https://orcid.org/0000-0001-5563-7840>), Malcolm Barrett [aut] (ORCID: <https://orcid.org/0000-0003-0299-5825>),

Josiah Parry [ctb] (ORCID: <https://orcid.org/0000-0001-9910-865X>),
Kenneth Vernon [ctb] (ORCID: <https://orcid.org/0000-0003-0098-5092>),
Alberson Miranda [ctb] (ORCID: <https://orcid.org/0000-0001-9252-4175>)

Maintainer Ilia Kosenkov <ilia.kosenkov@outlook.com>

Repository CRAN

Date/Publication 2025-06-19 12:40:02 UTC

Contents

`clean` *Clean Rust binaries and package cache.*

Description

Removes Rust binaries (such as `.dll`/`.so` libraries), C wrapper object files, invokes `cargo clean` to reset cargo target directory (found by default at `pkg_root/src/rust/target/`). Useful when Rust code should be recompiled from scratch.

Usage

```
clean(path = ".", echo = TRUE)
```

Arguments

`path`	character scalar, path to R package root.
`echo`	logical scalar, should cargo command and outputs be printed to console (default is `TRUE`)

Value

character vector with names of all deleted files (invisibly).

Examples

```
## Not run:
clean()

## End(Not run)
```

`cran` *CRAN compliant extendr packages*

Description

R packages developed using extendr are not immediately ready to be published to CRAN. The extendr package template ensures that CRAN publication is (farily) painless.

CRAN requirements

In order to publish a Rust based package on CRAN it must meet certain requirements. These are:

- Rust dependencies are vendored
- The package is compiled offline
- the `DESCRIPTION` file's `SystemRequirements` field contains `Cargo (Rust's package manager), rustc`

The extendr templates handle all of this *except* vendoring dependencies. This must be done prior to publication using `vendor_pkgs()`.

In addition, it is important to make sure that CRAN maintainers are aware that the package they are checking contains Rust code. Depending on which and how many crates are used as a dependencies the `vendor.tar.xz` will be larger than a few megabytes. If a built package is larger than 5mbs CRAN may reject the submission.

To prevent rejection make a note in your `cran-comments.md` file (create one using `usethis::use_cran_comments()`) along the lines of "The package tarball is 6mb because Rust dependencies are vendored within src/rust/vendor.tar.xz which is 5.9mb."

document *Compile Rust code and generate package documentation.*

Description

The function `rextendr::document()` updates the package documentation for an R package that uses `extendr` code, taking into account any changes that were made in the Rust code. It is a wrapper for `devtools::document()`, and it executes `extendr`-specific routines before calling `devtools::document()`. Specifically, it ensures that Rust code is recompiled (when necessary) and that up-to-date R wrappers are generated before regenerating the package documentation.

Usage

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

Arguments

`pkg`	The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
`quiet`	if `TRUE` suppresses output from this function.
`roclets`	Character vector of roclet names to use with package. The default, `NULL`, uses the roxygen `roclets` option, which defaults to `c("collate", "namespace", "rd")`.

Value

No return value, called for side effects.

eng_extendr *Knitr engines*

Description

Two knitr engines that enable code chunks of type `extendr` (individual Rust statements to be evaluated via `rust_eval()`) and `extendrsrc` (Rust functions or classes that will be exported to R via `rust_source()`).

Usage

```
eng_extendr(options)

eng_extendrsrc(options)
```

Arguments

`options`	A list of chunk options.

Value

A character string representing the engine output.

`read_cargo_metadata` *Retrieve metadata for packages and workspaces*

Description

Retrieve metadata for packages and workspaces

Usage

```
read_cargo_metadata(path = ".", dependencies = FALSE, echo = FALSE)
```

Arguments

`path`	character scalar, the R package directory
`dependencies`	Default `FALSE`. A logical scalar, whether to include all recursive dependencies in stdout.
`echo`	Default `FALSE`. A logical scalar, should cargo command and outputs be printed to the console.

Details

For more details, see Cargo docs for `cargo-metadata`. See especially "JSON Format" to get a sense of what you can expect to find in the returned list.

Value

A `list` including the following elements:

- `packages`
- `workspace_members`
- `workspace_default_members`
- `resolve`
- `target_directory`
- `version`
- `workspace_root`
- `metadata`

Examples

```
## Not run:
read_cargo_metadata()

## End(Not run)
```

`register_extendr` *Register the extendr module of a package with R*

Description

This function generates wrapper code corresponding to the extendr module for an R package. This is useful in package development, where we generally want appropriate R code wrapping the Rust functions implemented via extendr. In most development settings, you will not want to call this function directly, but instead call `rextendr::document()`.

Usage

```
register_extendr(path = ".", quiet = FALSE, force = FALSE, compile = NA)
```

Arguments

`path`	Path from which package root is looked up.
`quiet`	Logical indicating whether any progress messages should be generated or not.
`force`	Logical indicating whether to force regenerating `R/extendr-wrappers.R` even when it doesn't seem to need updated. (By default, generation is skipped when it's newer than the DLL).
`compile`	Logical indicating whether to recompile DLLs: `TRUE` always recompiles `NA` recompiles if needed (i.e., any source files or manifest file are newer than the DLL) `FALSE` never recompiles

Details

The function `register_extendr()` compiles the package Rust code if required, and then the wrapper code is retrieved from the compiled Rust code and saved into `R/extendr-wrappers.R`. Afterwards, you will have to re-document and then re-install the package for the wrapper functions to take effect.

Value

(Invisibly) Path to the file containing generated wrappers.

See Also

`document()`

`rust_eval` *Evaluate Rust code*

Description

Compile and evaluate one or more Rust expressions. If the last expression in the Rust code returns a value (i.e., does not end with ;), then this value is returned to R. The value returned does not need to be of type `Robj`, as long as it can be cast into this type with `.into()`. This conversion is done automatically, so you don't have to worry about it in your code.

Usage

```
rust_eval(code, env = parent.frame(), ...)
```

Arguments

`code`	Input rust code.
`env`	The R environment in which the Rust code will be evaluated.
`...`	Other parameters handed off to `rust_function()`.

Value

The return value generated by the Rust code.

Examples

```
## Not run:
# Rust code without return value, called only for its side effects
rust_eval(
  code = 'rprintln!("hello from Rust!");'
)

# Rust code with return value
rust_eval(
  code = "
    let x = 5;
    let y = 7;
    let z = x * y;
    z // return to R; rust_eval() takes care of type conversion code
 "
)

## End(Not run)
```

rust_sitrep *Report on Rust infrastructure*

Description

Prints out a detailed report on the state of Rust infrastructure on the host machine.

Usage

```
rust_sitrep()
```

Value

Nothing

rust_source *Compile Rust code and call from R*

Description

`rust_source()` compiles and loads a single Rust file for use in R. `rust_function()` compiles and loads a single Rust function for use in R.

Usage

```
rust_source(
  file,
  code = NULL,
  module_name = "rextendr",
  dependencies = NULL,
  patch.crates_io = getOption("rextendr.patch.crates_io"),
  profile = c("dev", "release", "perf"),
  toolchain = getOption("rextendr.toolchain"),
  extendr_deps = NULL,
  features = NULL,
  env = parent.frame(),
  use_extendr_api = TRUE,
  generate_module_macro = TRUE,
  cache_build = TRUE,
  quiet = FALSE,
  use_rtools = TRUE,
  use_dev_extendr = FALSE
)

rust_function(
```

```
  code,
  extendr_fn_options = NULL,
  env = parent.frame(),
  quiet = FALSE,
  use_dev_extendr = FALSE,
  ...
)
```

Arguments

file	Input rust file to source.
code	Input rust code, to be used instead of file.
module_name	Name of the module defined in the Rust source via extendr_module!. Default is "rextendr". If generate_module_macro is FALSE or if file is specified, should *match exactly* the name of the module defined in the source.
dependencies	Character vector of dependencies lines to be added to the Cargo.toml file.
patch.crates_io	Character vector of patch statements for crates.io to be added to the Cargo.toml file.
profile	Rust profile. Can be either "dev", "release" or "perf". The default, "dev", compiles faster but produces slower code.
toolchain	Rust toolchain. The default, NULL, compiles with the system default toolchain. Accepts valid Rust toolchain qualifiers, such as "nightly", or (on Windows) "stable-msvc".
extendr_deps	Versions of extendr-* crates. Defaults to rextendr.extendr_deps option (list(`extendr-api` = "*")) if use_dev_extendr is not TRUE, otherwise, uses rextendr.extendr_dev_deps option (list(`extendr-api` = list(git = "https://github.com/extendr/extendr")).
features	A vector of extendr-api features that should be enabled. Supported values are "ndarray", "faer", "either", "num-complex", "serde", and "graphics". Unknown features will produce a warning if quiet is not TRUE.
env	The R environment in which the wrapping functions will be defined.
use_extendr_api	Logical indicating whether use extendr_api::prelude::*; should be added at the top of the Rust source provided via code. Default is TRUE. Ignored for Rust source provided via file.
generate_module_macro	Logical indicating whether the Rust module macro should be automatically generated from the code. Default is TRUE. Ignored for Rust source provided via file. The macro generation is done with make_module_macro() and it may fail in complex cases. If something doesn't work, try calling make_module_macro() on your code to see whether the generated macro code has issues.
cache_build	Logical indicating whether builds should be cached between calls to rust_source().
quiet	Logical indicating whether compile output should be generated or not.

use_rtools
: Logical indicating whether to append the path to Rtools to the PATH variable on Windows using the RTOOLS4X_HOME environment variable (if it is set). The appended path depends on the process architecture. Does nothing on other platforms.

use_dev_extendr
: Logical indicating whether to use development version of extendr. Has no effect if extendr_deps are set.

extendr_fn_options
: A list of extendr function options that are inserted into #[extendr(...)] attribute

...
: Other parameters handed off to rust_source().

Value

The result from dyn.load(), which is an object of class DLLInfo. See getLoadedDLLs() for more details.

Examples

```
## Not run:
# creating a single rust function
rust_function("fn add(a:f64, b:f64) -> f64 { a + b }")
add(2.5, 4.7)

# creating multiple rust functions at once
code <- r"(
#[extendr]
fn hello() -> &'static str {
    "Hello, world!"
}

#[extendr]
fn test( a: &str, b: i64) {
    rprintln!("Data sent to Rust: {}, {}", a, b);
}
)"

rust_source(code = code)
hello()
test("a string", 42)


# use case with an external dependency: a function that converts
# markdown text to html, using the `pulldown_cmark` crate.
code <- r"(
  use pulldown_cmark::{Parser, Options, html};

  #[extendr]
  fn md_to_html(input: &str) -> String {
    let mut options = Options::empty();
    options.insert(Options::ENABLE_TABLES);
```

```
    let parser = Parser::new_ext(input, options);
    let mut output = String::new();
    html::push_html(&mut output, parser);
    output
  }
)"
rust_source(
  code = code,
  dependencies = list(`pulldown-cmark` = "0.8")
)

md_text <- "# The story of the fox
The quick brown fox **jumps over** the lazy dog.
The quick *brown fox* jumps over the lazy dog."

md_to_html(md_text)

## End(Not run)
```

`to_toml` *Convert R* `list()` *into toml-compatible format.*

Description

`to_toml()` can be used to build `Cargo.toml`. The cargo manifest can be represented in terms of R objects, allowing limited validation and syntax verification. This function converts manifests written using R objects into toml representation, applying basic formatting, which is ideal for generating cargo manifests at runtime.

Usage

```
to_toml(..., .str_as_literal = TRUE, .format_int = "%d", .format_dbl = "%g")
```

Arguments

`...`	A list from which toml is constructed. Supports nesting and tidy evaluation.
`.str_as_literal`	Logical indicating whether to treat strings as literal (single quotes no escapes) or basic (escaping some sequences) ones. Default is `TRUE`.
`.format_int, .format_dbl`	Character scalar describing number formatting. Compatible with `sprintf`.

Value

A character vector, each element corresponds to one line of the resulting output.

Examples

```
# Produces [workspace] with no children
to_toml(workspace = NULL)

to_toml(patch.crates_io = list(`extendr-api` = list(git = "git-ref")))

# Single-element arrays are distinguished from scalars
# using explicitly set `dim`
to_toml(lib = list(`crate-type` = array("cdylib", 1)))
```

`use_crate` *Add dependencies to a Cargo.toml manifest file*

Description

Analogous to `usethis::use_package()` but for crate dependencies.

Usage

```
use_crate(
  crate,
  features = NULL,
  git = NULL,
  version = NULL,
  optional = FALSE,
  path = ".",
  echo = TRUE
)
```

Arguments

`crate`	character scalar, the name of the crate to add
`features`	character vector, a list of features to include from the crate
`git`	character scalar, the full URL of the remote Git repository
`version`	character scalar, the version of the crate to add
`optional`	boolean scalar, whether to mark the dependency as optional (FALSE by default)
`path`	character scalar, the package directory
`echo`	logical scalar, should cargo command and outputs be printed to console (default is TRUE)

Details

For more details regarding these and other options, see the Cargo docs for `cargo-add`.

Value

`NULL` (invisibly)

Examples

```
## Not run:
# add to [dependencies]
use_crate("serde")

# add to [dependencies] and [features]
use_crate("serde", features = "derive")

# add to [dependencies] using github repository as source
use_crate("serde", git = "https://github.com/serde-rs/serde")

# add to [dependencies] with specific version
use_crate("serde", version = "1.0.1")

# add to [dependencies] with optional compilation
use_crate("serde", optional = TRUE)

## End(Not run)
```

`use_extendr` *Set up a package for use with Rust extendr code*

Description

Create the scaffolding needed to add Rust extendr code to an R package. `use_extendr()` adds a small Rust library with a single Rust function that returns the string `"Hello world!"`. It also adds wrapper code so this Rust function can be called from R with `hello_world()`.

Usage

```
use_extendr(
  path = ".",
  crate_name = NULL,
  lib_name = NULL,
  quiet = FALSE,
  overwrite = NULL,
  edition = c("2021", "2018")
)
```

Arguments

`path`	File path to the package for which to generate wrapper code.
`crate_name`	String that is used as the name of the Rust crate. If `NULL`, sanitized R package name is used instead.
`lib_name`	String that is used as the name of the Rust library. If `NULL`, sanitized R package name is used instead.
`quiet`	Logical indicating whether any progress messages should be generated or not.

overwrite	Logical scalar or NULL indicating whether the files in the path should be overwritten. If NULL (default), the function will ask the user whether each file should be overwritten in an interactive session or do nothing in a non-interactive session. If FALSE and each file already exists, the function will do nothing. If TRUE, all files will be overwritten.
edition	String indicating which Rust edition is used; Default "2021".

Value

A logical value (invisible) indicating whether any package files were generated or not.

---

use_msrv *Set the minimum supported rust version (MSRV)*

Description

use_msrv() sets the minimum supported rust version for your R package.

Usage

```
use_msrv(version, path = ".", overwrite = FALSE)
```

Arguments

version	character scalar, the minimum supported Rust version.
path	character scalar, path to folder containing DESCRIPTION file.
overwrite	default FALSE. Overwrites the SystemRequirements field if already set when TRUE.

Details

The minimum supported rust version (MSRV) is determined by the SystemRequirements field in a package's DESCRIPTION file. For example, to set the MSRV to 1.67.0, the SystemRequirements must have rustc >= 1.67.0.

By default, there is no MSRV set. However, some crates have features that depend on a minimum version of Rust. As of this writing the version of Rust on CRAN's Fedora machine's is 1.69. If you require a version of Rust that is greater than that, you must set it in your DESCRIPTION file.

It is also important to note that if CRAN's machines do not meet the specified MSRV, they will not be able to build a binary of your package. As a consequence, if users try to install the package they will be required to have Rust installed as well.

To determine the MSRV of your R package, we recommend installing the cargo-msrv cli. You can do so by running cargo install cargo-msrv. To determine your MSRV, set your working directory to src/rust then run cargo msrv. Note that this may take a while.

For more details, please see cargo-msrv.

Value

version

Examples

```
## Not run:
use_msrv("1.67.1")

## End(Not run)
```

---

vendor_pkgs *Vendor Rust dependencies*

Description

vendor_pkgs() is used to package the dependencies as required by CRAN. It executes cargo vendor on your behalf creating a vendor/ directory and a compressed vendor.tar.xz which will be shipped with package itself. If you have modified your dependencies, you will need need to repackage

Usage

```
vendor_pkgs(path = ".", quiet = FALSE, overwrite = NULL)
```

Arguments

path	File path to the package for which to generate wrapper code.
quiet	Logical indicating whether any progress messages should be generated or not.
overwrite	Logical scalar or NULL indicating whether the files in the path should be overwritten. If NULL (default), the function will ask the user whether each file should be overwritten in an interactive session or do nothing in a non-interactive session. If FALSE and each file already exists, the function will do nothing. If TRUE, all files will be overwritten.

Value

- vendor_pkgs() returns a data.frame with two columns crate and version

Examples

```
## Not run:
vendor_pkgs()

## End(Not run)
```

`write_license_note` *Generate LICENSE.note file.*

Description

LICENSE.note generated by this function contains information about all recursive dependencies in Rust crate.

Usage

```
write_license_note(path = ".", quiet = FALSE, force = TRUE)
```

Arguments

`path`	character scalar, the R package directory
`quiet`	logical scalar, whether to signal successful writing of LICENSE.note (default is `FALSE`)
`force`	logical scalar, whether to regenerate LICENSE.note if LICENSE.note already exists (default is `TRUE`)

Value

text printed to LICENSE.note (invisibly).

Examples

```
## Not run:
write_license_note()

## End(Not run)
```

Index