Package 'rextendr'

October 14, 2022

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

Version 0.2.0

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/

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

Depends R (>= 4.0)

Imports brio, callr, cli, desc, dplyr, glue, pkgbuild, pkgload, purrr, rlang (>= 0.4.10), rprojroot, stringi, tibble, withr

Suggests devtools, knitr, mockr, rmarkdown, rstudioapi, testthat (>= 3.0.2), usethis

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

RoxygenNote 7.1.1

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

NeedsCompilation no

Author Claus O. Wilke [aut, cre] (<https://orcid.org/0000-0002-7470-9261>), Andy Thomason [aut], Mossa M. Reimert [aut], Ilia Kosenkov [aut] (<https://orcid.org/0000-0001-5563-7840>), Hiroaki Yutani [aut] (<https://orcid.org/0000-0002-3385-7233>), Malcolm Barrett [aut] (<https://orcid.org/0000-0003-0299-5825>)

Maintainer Claus O. Wilke <wilke@austin.utexas.edu>

Repository CRAN

Date/Publication 2021-06-15 14:50:13 UTC

R topics documented:

`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 re-generating the package documentation.

Usage

```
document(pkg = ".", quiet = getOption("usethis.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.

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 re-generating `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()`

---

rextendr *Call Rust code from R using the 'extendr' Crate*

Description

The rextendr package implements functions to interface with Rust code from R. See `rust_source()` for details.

---

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_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"),
  toolchain = getOption("rextendr.toolchain"),
  extendr_deps = getOption("rextendr.extendr_deps"),
  env = parent.frame(),
  use_extendr_api = TRUE,
  generate_module_macro = TRUE,
  cache_build = TRUE,
  quiet = FALSE,
  use_rtools = TRUE
```

```
)

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

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".
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" or "release". 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 list(`extendr-api` = "*").
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 RTOOLS40_HOME environment variable (if it is set). The appended path depends on the process architecture. Does nothing on other platforms.
...	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 = )

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_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 = ".", quiet = getOption("usethis.quiet", FALSE))
```

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. Also checks the `usethis.quiet` option.

Details

To avoid possibly messing up your R package, `use_extendr()` will not do anything if either a directory `src` or a file `R/extendr-wrappers.R` is already present in your package source.

Value

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

Index