

Package ‘rzig’

October 22, 2026

Type Package

Title Safe Zig Bindings

Version 1.0.0

Description Scaffolds packages backed by Zig and generates a safe native boundary for plain Zig functions. The generated integration converts objects at the language boundary, registers native routines, propagates errors, and compiles across platforms without requiring C++. Each step of the workflow is a function: locating a supported Zig compiler, scaffolding the package, scanning the Zig source for exported functions, rendering and writing the bindings, checking the package before compilation, and building it with the compiler errors reported by file and line. The scan, the checks and the generated bindings are pure R; a Zig compiler is needed only to build a package created with the scaffold.

License GPL-3

URL <https://marekdejauj.github.io/rzig/>,
<https://github.com/MarekDejaUJ/rzig>

BugReports <https://github.com/MarekDejaUJ/rzig/issues>

Encoding UTF-8

SystemRequirements Zig 0.16.x or 0.17.x
(<https://ziglang.org/download/>), needed to install packages
created with rzig

NeedsCompilation no

Suggests digest, knitr, pkgload, rmarkdown, roxygen2, testthat (>= 3.0.0), yaml

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Author Marek Deja [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0015-2322>>)

Maintainer Marek Deja <marek.deja@uj.edu.pl>

Repository CRAN

Date/Publication 2026-10-22 08:00:02 UTC

Contents

rzig-package	2
build_zig	3
create_rzig_package	4
document	5
eng_zig	6
find_zig	7
install_zig	8
render_bindings	9
rzig_status	10
scan_exports	11
test_zig	12
update_rzig	13
use_rzig	14
use_rzig_ci	16
use_rzig_import	17
use_rzig_lapack	18
use_rzig_mixed	19
zig_function	20

Index	22
--------------	-----------

rzig-package

Safe Zig Bindings

Description

RZig scaffolds packages backed by Zig and generates safe native bindings for plain Zig functions. Each step of the workflow is a function: `find_zig` locates a supported Zig compiler, `use_rzig` adds the build files and the framework sources to a package, `scan_exports` lists the exported Zig functions with their R arguments, `render_bindings` shows the generated manifest, R wrappers and NAMESPACE block, `document` writes them, `rzig_status` reports which steps are complete and which signatures the boundary rejects, and `build_zig` compiles and installs the package, reporting compiler errors by file and line. The scan and the generated bindings are pure R; a Zig compiler of the 0.16 or 0.17 release series is needed only to install a package created with the scaffold.

Author(s)

Marek Deja

build_zig

Compile and install a Zig-backed package, and report the first errors

Description

Runs the checks of `rzig_status()`, then installs a copy of the package with R CMD INSTALL into a library, using the Zig compiler reported by `find_zig()`. The complete installation log is kept, and the errors of the Zig compiler, the C compiler, the linker and the load test are extracted with their files and lines, so a failed build names its cause instead of the bare "non-zero exit status" of `utils::install.packages()`.

Usage

```
build_zig(
  path,
  lib = NULL,
  preflight = TRUE,
  optimize = Sys.getenv("RZIG_OPTIMIZE", "ReleaseSafe")
)
```

Arguments

path	Path to the package root.
lib	The library to install into. By default a new library in the session temporary directory; <code>library()</code> then needs <code>lib.loc = build\$library</code> .
preflight	If TRUE, run the checks of <code>rzig_status()</code> first and stop before compiling when one of them fails.
optimize	The optimize mode of the Zig code: "ReleaseSafe" (optimized, with Zig's safety checks) or "Debug" (unoptimized, with the same safety checks and complete stack traces, for debugging). The default is the environment variable RZIG_OPTIMIZE, which the generated Makevars also read, or "ReleaseSafe". The modes without safety checks (ReleaseFast, ReleaseSmall) are not available, because the boundary turns failed checks into R errors; <code>@setRuntimeSafety(false)</code> removes the checks of a single function.

Details

The package is built from a copy in the session temporary directory, so the package source receives no object files or compiler caches. The installation needs a supported Zig compiler and the C toolchain of R. A package that is already loaded in the R session keeps its old native code until R restarts; the printed result says so.

Value

An object of class `rzig_build`: a list with `ok` (logical), `package`, `step` ("preflight", "configure", "compile", "link", "load", "install" or NA on success), `errors` (a data frame with the columns

file, line, column, message and function_name), problems (failed checks before compilation), library, zig (the [rzig_compiler](#) used), optimize, seconds and log (the complete output of R CMD INSTALL). A failed build also signals a warning.

Examples

```
package_path <- tempfile("rzig-build-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)

# A change to main.zig without document() is caught before compiling.
main <- file.path(package_path, "src", "rzig", "src", "main.zig")
cat("/// @export\npub fn twice(x: f64) f64 {\n    return 2 * x;\n}\n",
  file = main, append = TRUE)
result <- suppressWarnings(build_zig(package_path))
result

# With a supported Zig compiler, the regenerated package compiles; the
# compilation takes several seconds.

if (isTRUE(find_zig(required = FALSE)$supported)) {
  document(package_path)
  result <- build_zig(package_path)
  result
}

unlink(package_path, recursive = TRUE)
```

create_rzig_package	<i>Create a package that uses RZig</i>
---------------------	--

Description

Creates a new package at path: a DESCRIPTION, the RZig scaffold of [use_rzig\(\)](#) with its [hello_zig\(\)](#) example and help page, and, by default, a testthat suite with one test of [hello_zig\(\)](#). The package builds with R CMD INSTALL once a supported Zig compiler is available (see [find_zig\(\)](#)) and passes R CMD check as created.

Usage

```
create_rzig_package(path, fields = list(), tests = TRUE, ci = FALSE)
```

Arguments

path	The directory to create. It must not exist or must be empty.
fields	A named list of DESCRIPTION fields that replace or extend the defaults.

tests	If TRUE, add tests/testthat.R and a test of <code>hello_zig()</code> , with testthat in Suggests.
ci	If TRUE, add the GitHub Actions workflow of <code>use_rzig_ci()</code> .

Details

The package name is the last component of path. The default DESCRIPTION fields are placeholders for the title, the description and the author; fields replaces or adds fields, for example `fields = list(Title = "Fast Rank Statistics", "Authors@R" = 'person("Ada", "Lovelace", email = "ada@example.org", role = c("aut", "cre"))')`. The license is GPL-3, the license of the framework sources the package contains (another field value is the package author's choice).

Value

The normalized path of the package, invisibly.

Examples

```
package_path <- file.path(tempdir(), "zigstats")
create_rzig_package(package_path)
list.files(package_path, recursive = TRUE)
unlink(package_path, recursive = TRUE)
```

document

Generate R bindings from Zig exports

Description

Scans the Zig source of the package with `scan_exports()`, renders the bindings with `render_bindings()`, and writes them: the Zig manifest `src/rzig/framework/generated/manifest.zig`, the R wrappers `R/rzig-wrappers.R`, a managed block in NAMESPACE, and a help page `man/<name>.Rd` for every exported function, written from its Zig doc comment (the first paragraph is the title and description; `@param`, `@return`, `@examples`, `@details`, `@seealso`, `@references` and `@note` fill the matching sections). Lines of NAMESPACE outside the block and help pages that rzig did not write are kept. The function is pure R; no Zig compiler is needed.

Usage

```
document(path)
```

Arguments

path Path to a package previously initialized with `use_rzig()`.

Details

When roxygen2 generates NAMESPACE (its first line is # Generated by roxygen2: do not edit by hand), the useDynLib directive is written into R/rzig-wrappers.R as a @rawNamespace tag, and NAMESPACE receives the same lines roxygen2 writes from the tags. The package then works whichever of document(), roxygen2::roxygenise() or devtools::document() ran last; run document() after changing an exported Zig signature so that the wrappers follow it. roxygen2 then writes the help pages from the wrappers, and document() writes none.

Run document() after every change to an exported Zig signature or its documentation comment, and before installing the package.

Value

The `rzig_bindings` object that was written, with the element files naming the files written, invisibly.

Examples

```
package_path <- tempfile("rzig-document-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
bindings <- document(package_path)
bindings$files
cat(readLines(file.path(package_path, "NAMESPACE")), sep = "\n")
unlink(package_path, recursive = TRUE)
```

eng_zig

A knitr engine for Zig chunks

Description

Compiles the code of a chunk with `zig_function()` and defines its exported functions in the environment of the document, so later R chunks can call them. rzig registers the engine as zig when knitr is loaded; to register it explicitly, call `knitr::knit_engines$set(zig = rzig::eng_zig)`.

Usage

```
eng_zig(options)
```

Arguments

options The chunk options knitr passes to an engine.

Value

The chunk output for knitr.

Examples

```
# A document can use a chunk such as
# ```{zig}
#   pub fn twice(x: f64) f64 { return 2 * x; }
#   ```
# after which R chunks call twice(). The engine is registered as "zig":
if (requireNamespace("knitr", quietly = TRUE)) {
  knitr::knit_engines$set(zig = eng_zig)
  is.function(knitr::knit_engines$get("zig"))
}
```

find_zig

Locate a supported Zig compiler

Description

Uses the compiler named by the environment variable ZIG when it is set. Otherwise searches the PATH, the installations of `install_zig()` in `zig_home()`, `~/.local/share/zig/*/zig` and `~/zig/zig`, in that order, and reports the first compiler of a supported release series; when none is supported, it reports the first compiler found and why it is rejected. `rzig` itself runs without Zig: `use_rzig()`, `scan_exports()` and `document()` are pure R. A Zig compiler of the supported release series is needed to install a package created with `use_rzig()`.

Usage

```
find_zig(required = TRUE)
```

Arguments

<code>required</code>	If TRUE, a missing or unsupported compiler is an error. If FALSE, the returned object records the problem instead.
-----------------------	--

Details

`rzig` supports the Zig 0.16 and 0.17 release series (versions `0.16.x` and `0.17.x`). Zig changes its language and standard library between release series, so a newer or older compiler is reported as unsupported. Development builds such as `0.17.0-dev.1234+abcdef` are unsupported too: their language and standard library can differ from the release.

Value

An object of class `rzig_compiler`: a list with `path` (character or NULL), `version` (character or NULL), `supported` (logical) and `series` (the supported release series, `"0.16.x"` or `"0.17.x"`).

Examples

```
find_zig(required = FALSE)
```

install_zig

*Download and install a supported Zig compiler***Description**

Downloads the Zig release version for this platform, verifies the archive against the size and SHA-256 that <https://ziglang.org/download/index.json> publishes (pinned in rzig), and unpacks it into dir. `find_zig()` and the configure scripts of packages created with `use_rzig()` find compilers installed in `zig_home()`, so R CMD INSTALL of such a package needs no further setup.

Usage

```
install_zig(
  version = "0.16.0",
  dir = zig_home(),
  mirrors = TRUE,
  quiet = FALSE
)

uninstall_zig(version = NULL, dir = zig_home())

zig_home()
```

Arguments

version	The Zig release: "0.16.0" or "0.17.0".
dir	The folder that holds the installations.
mirrors	If FALSE, download from ziglang.org only.
quiet	If TRUE, show no download progress.

Details

The archive (about 55 MB, about 300 MB unpacked) comes from one of the community mirrors of Zig, tried in an order that varies between sessions, or else from ziglang.org, as the Zig Software Foundation asks of automated downloads. A download whose size or SHA-256 differs from the pin is deleted before anything is extracted or run. An existing installation of the same version is reused. `uninstall_zig()` removes installations made by `install_zig()`.

`zig_home()` is the folder named by the environment variable RZIG_ZIG_HOME, or zig in the user cache folder of rzig (`tools::R_user_dir()`).

Value

`install_zig()`: the path of the zig executable, invisibly. `uninstall_zig()`: the folders removed, invisibly. `zig_home()`: a path.

Examples

```
zig_home()
## Not run:
install_zig("0.16.0")
find_zig()

## End(Not run)
```

render_bindings

Render the generated bindings for a set of exported functions

Description

Produces the three texts that `document()` writes into a package, without writing anything: the Zig manifest that registers the exported functions, the R wrappers that call them, and the `NAMESPACE` block that resolves the native symbols. Use the function to inspect what `document()` will generate.

Usage

```
render_bindings(
  exports,
  package = attr(exports, "package", exact = TRUE),
  roxygen = FALSE
)
```

Arguments

exports	An object returned by <code>scan_exports()</code> .
package	The package name; by default taken from exports.
roxygen	Render for a package whose <code>NAMESPACE</code> roxygen2 generates: the wrappers then carry the <code>useDynLib</code> directive as a <code>@rawNamespace</code> tag, and namespace holds the lines roxygen2 writes for them.

Value

An object of class `rzig_bindings`: a list with the character strings `manifest` (`src/rzig/framework/generated/manifest`), `wrappers` (`R/rzig-wrappers.R`) and `namespace` (the generated `NAMESPACE` block, or with `roxygen = TRUE` the `NAMESPACE` lines that roxygen2 writes for the wrappers), and the `exports` the texts were rendered from.

Examples

```
package_path <- tempfile("rzig-render-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
```

```
bindings <- render_bindings(scan_exports(package_path))
cat(bindings$wrappers)
cat(bindings$namespace)
unlink(package_path, recursive = TRUE)
```

rzig_status

Report the state of a Zig-backed package

Description

Checks each step of the RZig workflow in a package and reports what is present, what is missing and what would make the installation fail:

Usage

```
rzig_status(path, build = FALSE, lib = NULL)
```

Arguments

path	Path to the package root.
build	If TRUE, also compile and install the package with <code>build_zig()</code> when every other check passes.
lib	With build = TRUE, the library to install into; see <code>build_zig()</code> .

Details

- **DESCRIPTION:** the package name and a valid version.
- **Scaffold:** the files written by `use_rzig()`, framework sources that match the installed rzig, and executable configure and cleanup scripts. Build files that differ from the rzig templates (configure, the Makevars, entry.c, build.zig, build.zig.zon, tools/) are noted, since a package may extend them.
- **Zig source:** `src/rzig/src/main.zig` with the registration block and the panic handler of rzig.
- **Exports:** the exported functions found by `scan_exports()`, with every argument and return type that the boundary rejects, more than 32 arguments, misuse of `rzig.Mut([]f64)`, error sets other than `rzig.Error`, and R functions of the same name elsewhere in R/. An export that masks a function of base R (for example `log`) is noted.
- **Bindings:** whether the files written by `document()` match the exports.
- **Zig compiler:** a compiler of a supported release series (`find_zig()`).
- **C toolchain:** the C compiler of R, make and the R headers.
- **Build** (only with build = TRUE): the result of `build_zig()`.

Every check except the build runs in R without compiling anything.

Value

An object of class `rzig_status`: a data frame with the columns `step`, `ok` and `detail`, one row per check, with the attributes `path`, `package` and, with `build = TRUE`, `build` (the [rzig_build](#) object).

Examples

```
package_path <- tempfile("rzig-status-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
rzig_status(package_path)
main <- file.path(package_path, "src", "rzig", "src", "main.zig")
cat("/// @export\npub fn half(x: f32) f32 {\n    return x / 2;\n}\n",
  file = main, append = TRUE)
rzig_status(package_path)
unlink(package_path, recursive = TRUE)
```

scan_exports

*Scan the Zig source of a package for exported functions***Description**

Reads `src/rzig/src/main.zig`, and every Zig file that it imports with `pub const name = @import("file.zig");` (recursively), and lists every public top-level Zig function whose documentation comment contains a line `/// @export`. A function of an imported file is registered as `name.function`; its R name is the function name. `/// @export` above `pub const twice = other.twice`; re-exports a function of a file imported as `other` under the name `twice`. A file imported with a plain `const` cannot be reached by the generated registration, so a marker in it is an error; a marked file that is not imported at all is ignored, and [rzig_status\(\)](#) notes it. R names must be unique across all files.

Usage

```
scan_exports(path)
```

Arguments

`path` Path to a package previously initialized with [use_rzig\(\)](#).

Details

The marker line takes options: `/// @export internal` registers the function without exporting its R wrapper, `/// @export invisible` makes the wrapper return its value invisibly, and `/// @export name=<R name>` gives the wrapper an R name other than the Zig name. Lines `/// @default <argument> <R expression>` give default arguments of the wrapper, for example `/// @default tolerance 1e-8`. A line `/// @export internal` registers the function and generates its R wrapper without exporting the wrapper from the package, for native routines that

the package's own R code calls. The scan is pure R and needs no Zig compiler; the Zig compiler checks the parameter and return types when the package is installed.

A struct whose documentation comment contains `/// @class` becomes an R class: its `pub fn init` is the constructor, exported under the name of the struct, and every public function whose first parameter is the struct (`self: *T`, `self: *const T` or `self: T`) is a method, registered as the internal routine `T_method` and called as `x$method(...)` on an object `x`. Objects are [rzig.External](#) values (see the README); `deinit`, when declared, runs when R collects the object.

A function is found when its declaration starts at the beginning of a line with `pub fn` and the lines directly above it are `///` documentation comments (blank lines and ordinary `//` comments between the comment block and the declaration are ignored). The first parameter is omitted from the R signature when its type is `*rzig.Ctx`.

Value

An object of class `rzig_exports`: a list with one element per exported function, each a list with name (the R function name), identifier (the Zig identifier as written), parameters (the R argument names), parameter_types (the Zig types of the R arguments), return_type (the Zig return type as written), doc (the documentation comment without the `@export` line), line (the line of the declaration), internal (TRUE for `/// @export internal`) and file (the file below `src/rzig/src`), invisible and defaults (a named character vector of R expressions). The attributes `package`, `path`, `source` (the entry file), `files` (every file scanned) and `unreachable` (marked files that `main.zig` does not import) record where the exports come from. [as.data.frame\(\)](#) turns the object into a table with one row per function.

Examples

```
package_path <- tempfile("rzig-scan-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
exports <- scan_exports(package_path)
exports
as.data.frame(exports)
unlink(package_path, recursive = TRUE)
```

test_zig

Run the Zig tests of a package

Description

Runs `zig build test` for the client package: the test blocks reached from `src/rzig/src/tests.zig` are compiled for the host and run without R. `tests.zig` imports the files whose tests should run, for example `test { _ = @import("kernels.zig"); }`; files that use the `rzig` module need R and cannot run there, so numerical kernels are best kept in files of plain Zig. The package folder receives no build output.

Usage

```
test_zig(path)
```

Arguments

path Path to a package previously initialized with `use_rzig()`.

Value

An object of class `rzig_zig_tests`: a list with `ok`, `failed` (the names of failed tests), `seconds` and `log`, invisibly. A failure also signals a warning.

Examples

```
if (isTRUE(find_zig(required = FALSE)$supported)) {
  package_path <- tempfile("rzig-tests-", tmpdir = tempdir())
  dir.create(package_path)
  writeLines(c("Package: examplepkg", "Version: 0.0.1"),
    file.path(package_path, "DESCRIPTION"))
  use_rzig(package_path)
  writeLines(c("const std = @import(\"std\");", "test \"addition\" {",
    "  try std.testing.expectEqual(4, 2 + 2);", "}"),
    file.path(package_path, "src", "rzig", "src", "tests.zig"))
  test_zig(package_path)
  unlink(package_path, recursive = TRUE)
}
```

update_rzig

Update the rzig framework and build files of a package

Description

Brings a package created with `use_rzig()` to the installed version of `rzig` without touching the package author's work:

Usage

```
update_rzig(path, force = FALSE)
```

Arguments

path Path to a package previously initialized with `use_rzig()`.

force If TRUE, also replace customized build files with the templates of the installed `rzig`.

Details

- the framework sources under `src/rzig/framework` are replaced by those of the installed `rzig` (the generated manifest is rewritten by `document()`);
- each build file (`configure`, `configure.win`, `cleanup`, `cleanup.win`, `src/Makevars.in`, `src/Makevars.win.in`, `src/entry.c`, `src/rzig/build.zig`, `src/rzig/build.zig.zon` and `src/rzig/tools/repack_macos_archive.sh`) is replaced only when it is identical to a template shipped by some version of `rzig`, that is, when the package author has not changed it. A changed (customized) build file is kept and listed, so that its differences from the new template can be reviewed; `force = TRUE` replaces it as well;
- missing build files are added, and the Zig scanner that `rzig 0.2` copied into `src/rzig/tools/scan.zig` is removed;
- the package author's Zig sources in `src/rzig/src`, the list of C libraries `src/rzig/c_libraries.zon` (added when missing) and any file `rzig` does not know are never changed or removed;
- finally `document()` regenerates the bindings.

Value

An object of class `rzig_update`: a list with `path`, `package`, `actions` (a data frame with the columns `file` and `action`, one of "refreshed", "updated", "added", "removed", "kept" (customized), "replaced" (customized, with `force = TRUE`) or "current"), `unknown` (files under `src/rzig/framework` that `rzig` does not ship) and `exports`, invisibly.

Examples

```
package_path <- tempfile("rzig-update-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
# A customized build file is kept.
cat("# package-specific setting\n",
  file = file.path(package_path, "src", "Makevars.in"), append = TRUE)
update_rzig(package_path)
unlink(package_path, recursive = TRUE)
```

use_rzig

Add RZig to a package

Description

Creates the Zig source tree, the portable build files, the native entry stub and a working `hello_zig()` example in an existing package, then generates the bindings with `document()`. The function is pure R; a Zig compiler of the supported release series (see `find_zig()`) is needed only when the package is installed.

Usage

```
use_rzig(path, overwrite = FALSE, force = FALSE, mixed = FALSE)
```

Arguments

path	Path to the package root; the directory must contain a DESCRIPTION file.
overwrite	Allow use_rzig() on a package that already has RZig files. The framework sources and every build file that still matches an rzig template are then replaced, as with update_rzig() ; the package author's Zig sources in src/rzig/src and files rzig does not know are kept. A build file the package author changed stops the call unless force = TRUE. update_rzig() is the usual way to update a package: it keeps customized build files and lists them.
force	With overwrite = TRUE, also replace customized build files with the templates.
mixed	Build the C, C++ and Fortran sources of src/ with R's compilers next to the Zig code; see use_rzig_mixed() . Without it, use_rzig() stops when src/ contains such sources.

Details

The files written below the package root are: configure, configure.win, cleanup, cleanup.win, src/entry.c, src/Makevars.in, src/Makevars.win.in, the framework sources under src/rzig/framework, the list of vendored C libraries src/rzig/c_libraries.zon (empty), the package author's Zig source src/rzig/src/main.zig, the generated R/rzig-wrappers.R, and a managed block in NAMESPACE. Entries for the Zig build caches are added to .Rbuildignore, and the Zig sources are listed in the Config/build/extra-sources field of DESCRIPTION, so that devtools::load_all() recompiles the package after a change to a Zig file.

Value

An object of class rzig_scaffold: a list with path (the normalized package root), package, files (the paths written, relative to the root) and exports (the [rzig_exports](#) found in main.zig), invisibly.

Examples

```
package_path <- tempfile("rzig-example-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
scaffold <- use_rzig(package_path)
scaffold
file.exists(file.path(package_path, "src", "rzig", "src", "main.zig"))
unlink(package_path, recursive = TRUE)
```

use_rzig_ci

Add a GitHub Actions workflow that checks the package with Zig

Description

Writes `.github/workflows/rzig-check.yaml`, a workflow that installs R and Zig and runs R CMD check `--as-cran` on each operating system and Zig version of its matrix. When `src/rzig/src/tests.zig` exists, the Zig tests (`test_zig()`) run before the check. `.github` is added to `.Rbuildignore`.

Usage

```
use_rzig_ci(
  path,
  zig = c("0.16.0", "0.17.0"),
  os = c("ubuntu-latest", "macos-latest", "windows-latest"),
  overwrite = FALSE
)
```

Arguments

<code>path</code>	Path to a package previously initialized with <code>use_rzig()</code> .
<code>zig</code>	The Zig versions to test: releases of the series rzig supports (see <code>find_zig()</code>).
<code>os</code>	The GitHub runners to test on.
<code>overwrite</code>	If TRUE, replace an existing workflow file.

Value

The path of the workflow file, invisibly.

Examples

```
package_path <- tempfile("rzig-ci-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
use_rzig_ci(package_path, os = "ubuntu-latest")
unlink(package_path, recursive = TRUE)
```

use_rzig_import	<i>Share Zig modules between R packages</i>
-----------------	---

Description

A package can share Zig code with other packages as a Zig module: the file `inst/zig/<module>/root.zig`, with the files it imports from that folder, installed with the package. `use_rzig_import()` lets the Zig code of the package at path use such a module with `@import("<module>")`: it adds `<package>/<module>` to the field `Config/rzig/imports` of `DESCRIPTION` and `<package>` to `LinkingTo`, so that R installs the providing package first. `configure` finds the module in the installed package and passes it to `zig build`; the module can import `rzig` and the other imported modules. The module is compiled into the importing package, so nothing is linked between the packages when they run.

Usage

```
use_rzig_import(path, package, module)

use_rzig_module(path, module, import = TRUE)
```

Arguments

<code>path</code>	Path to the package root.
<code>package</code>	The name of the package that provides the module.
<code>module</code>	The name of the Zig module, a Zig identifier other than <code>rzig</code> , <code>std</code> , <code>builtin</code> and <code>root</code> .
<code>import</code>	For <code>use_rzig_module()</code> : also import the module in the package's own Zig code.

Details

`use_rzig_module()` creates `inst/zig/<module>/root.zig` with a starter in the providing package and, with `import = TRUE`, imports the module in the package's own Zig code as well (an entry that names the package itself refers to its source). It also writes `inst/include/<package>_rzig.h`, a header without declarations: R CMD check expects a package in `LinkingTo` to provide an include directory.

Value

The path of `DESCRIPTION` (`use_rzig_import()`) or of the module's `root.zig` (`use_rzig_module()`), invisibly.

Examples

```
provider <- tempfile("rzig-provider-", tmpdir = tmpdir())
dir.create(provider)
writeLines(c("Package: kernelpkg", "Version: 0.0.1"), file.path(provider, "DESCRIPTION"))
use_rzig_module(provider, "kernels")
consumer <- tempfile("rzig-consumer-", tmpdir = tmpdir())
dir.create(consumer)
writeLines(c("Package: userpkg", "Version: 0.0.1"), file.path(consumer, "DESCRIPTION"))
use_rzig_import(consumer, "kernelpkg", "kernels")
read.dcf(file.path(consumer, "DESCRIPTION"), fields = c("LinkingTo", "Config/rzig/imports"))
unlink(c(provider, consumer), recursive = TRUE)
```

use_rzig_lapack

Link the LAPACK and BLAS libraries of R

Description

Adds Config/rzig/lapack: true to DESCRIPTION. The generated configure then links \$(LAPACK_LIBS) \$(BLAS_LIBS) the libraries that rzig.Blas and rzig.Lapack call, in the order *Writing R Extensions* asks for. \$(FLIBS) is the Fortran runtime of R: on macOS, installing such a package from source needs the gfortran that CRAN's R builds use (<https://mac.r-project.org/tools/>), on Windows it comes with Rtools.

Usage

```
use_rzig_lapack(path)
```

Arguments

path Path to a package previously initialized with `use_rzig()`.

Value

The path of DESCRIPTION, invisibly.

Examples

```
package_path <- tempfile("rzig-lapack-", tmpdir = tmpdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig(package_path)
use_rzig_lapack(package_path)
read.dcf(file.path(package_path, "DESCRIPTION"), fields = "Config/rzig/lapack")
unlink(package_path, recursive = TRUE)
```

use_rzig_mixed	<i>Build C, C++ and Fortran sources next to the Zig code</i>
----------------	--

Description

Adds `Config/rzig/mixed: true` to `DESCRIPTION`. The generated configure then leaves `OBJECTS` unset, so that R compiles every C, C++ and Fortran source file of `src/` with its own compilers and links them with the Zig library, and `src/entry.c` leaves `R_init_<pkg>` to the package. The package's `R_init_<pkg>` registers its own routines with `rzig_register()`, which adds the Zig routines to the `.Call` routines:

Usage

```
use_rzig_mixed(path)
```

Arguments

path	Path to the package root.
------	---------------------------

Details

```
#include <R_ext/Rdynload.h>

void rzig_register(DllInfo *dll, const R_CMethodDef *c_methods,
                  const R_CallMethodDef *call_methods,
                  const R_FortranMethodDef *fortran_methods,
                  const R_ExternalMethodDef *external_methods);

static const R_CallMethodDef call_methods[] = {
    {"legacy_add", (DL_FUNC) &legacy_add, 2},
    {NULL, NULL, 0}
};

void R_init_mypkg(DllInfo *dll) {
    rzig_register(dll, NULL, call_methods, NULL, NULL);
}
```

Dynamic symbol lookup stays disabled; the R code may name the routines by symbol or by string. A C routine with the name of a Zig export is not registered (a message says so when the package loads). [use_rzig\(\)](#) with `mixed = TRUE` calls this function before it scaffolds the package.

Value

The path of `DESCRIPTION`, invisibly.

Examples

```
package_path <- tempfile("rzig-mixed-", tmpdir = tempdir())
dir.create(package_path)
writeLines(c("Package: examplepkg", "Version: 0.0.1"),
  file.path(package_path, "DESCRIPTION"))
use_rzig_mixed(package_path)
read.dcf(file.path(package_path, "DESCRIPTION"), fields = "Config/rzig/mixed")
unlink(package_path, recursive = TRUE)
```

zig_function

Compile Zig code from the R prompt

Description

zig_function() compiles Zig source code into a small package in the session's temporary directory and defines its exported functions in env, the counterpart of Rcpp::cppFunction(). zig_source() does the same for a file, and zig_eval() evaluates one Zig expression.

Usage

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

```
zig_source(file, env = parent.frame(), quiet = TRUE)
```

```
zig_eval(expr, type = "f64")
```

Arguments

code	Zig source code, a character vector whose elements are lines.
env	The environment in which the functions are defined.
quiet	If FALSE, print the build result.
file	A Zig source file.
expr	A Zig expression, as a string.
type	The Zig type of the value of expr, a return type the boundary supports (for example "f64", "i32", "bool", "[]const u8").

Details

The code is placed in a generated main.zig after the lines `const std = @import("std");`, `const builtin = @import("builtin");`, `const rzig = @import("rzig");` and the panic handler declaration (each left out when the code declares it), and before the registration block. Public functions marked `/// @export` are exported; when the code marks none, every top-level `pub fn` is exported. The types, errors and checks of the boundary are those of a package ([scan_exports\(\)](#), [rzig_status\(\)](#)).

Builds are cached for the session by the code, the rzig version and the Zig compiler: calling zig_function() again with the same code defines the functions without compiling. A failed build is reported with the lines of code and stops with an error.

Value

`zig_function()` and `zig_source()`: the function, if the code exports exactly one, otherwise a named list of the functions, invisibly. `zig_eval()`: the value of the expression.

Examples

```
if (isTRUE(find_zig(required = FALSE)$supported)) {  
  twice <- zig_function("pub fn twice(x: f64) f64 { return 2 * x; }")  
  twice(21)  
  zig_eval("@sqrt(2.0)", type = "f64")  
}
```

Index

`as.data.frame()`, [12](#)

`build_zig`, [2, 3](#)
`build_zig()`, [10](#)

`create_rzig_package`, [4](#)

`document`, [2, 5](#)
`document()`, [7, 9, 10, 14](#)

`eng_zig`, [6](#)

`find_zig`, [2, 7](#)
`find_zig()`, [3, 4, 8, 10, 14, 16](#)

`install_zig`, [8](#)
`install_zig()`, [7](#)

`library()`, [3](#)

`render_bindings`, [2, 9](#)
`render_bindings()`, [5](#)
`rzig (rzig-package)`, [2](#)
`rzig-package`, [2](#)
`rzig.External`, [12](#)
`rzig_bindings`, [6](#)
`rzig_build`, [11](#)
`rzig_compiler`, [4](#)
`rzig_exports`, [15](#)
`rzig_status`, [2, 10](#)
`rzig_status()`, [3, 11, 20](#)

`scan_exports`, [2, 11](#)
`scan_exports()`, [5, 7, 9, 10, 20](#)

`test_zig`, [12](#)
`test_zig()`, [16](#)
`tools::R_user_dir()`, [8](#)

`uninstall_zig (install_zig)`, [8](#)
`update_rzig`, [13](#)
`update_rzig()`, [15](#)

`use_rzig`, [2, 14](#)
`use_rzig()`, [4, 5, 7, 8, 10, 11, 13, 16, 18, 19](#)
`use_rzig_ci`, [16](#)
`use_rzig_ci()`, [5](#)
`use_rzig_import`, [17](#)
`use_rzig_lapack`, [18](#)
`use_rzig_mixed`, [19](#)
`use_rzig_mixed()`, [15](#)
`use_rzig_module (use_rzig_import)`, [17](#)
`utils::install.packages()`, [3](#)

`zig_eval (zig_function)`, [20](#)
`zig_function`, [20](#)
`zig_function()`, [6](#)
`zig_home (install_zig)`, [8](#)
`zig_home()`, [7](#)
`zig_source (zig_function)`, [20](#)