Package ‘pkgbuild’

October 9, 2019

Title Find Tools Needed to Build R Packages

Version 1.0.6

Description Provides functions used to build R packages. Locates compilers needed to build R packages on various platforms and ensures the PATH is configured appropriately so R can use them.

Imports callr (>= 3.2.0), cli, crayon, desc, prettyunits, R6, rprojroot, withr (>= 2.1.2)

Suggests Rcpp, testthat, covr

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

URL https://github.com/r-lib/pkgbuild

BugReports https://github.com/r-lib/pkgbuild/issues

Depends R (>= 3.1)

NeedsCompilation no

Author Hadley Wickham [aut],
Jim Hester [aut, cre],
RStudio [cph]

Maintainer Jim Hester <jim.hestер@rstudio.com>

Repository CRAN

Date/Publication 2019-10-09 14:00:02 UTC

R topics documented:

build ................................................................. 2
clean_dll .......................................................... 3
compiler_flags ..................................................... 3
compile_dll ...................................................... 4
has_build_tools .................................................. 5
**build**

Description

Building converts a package source directory into a single bundled file. If `binary = FALSE` this creates a tar.gz package that can be installed on any platform, provided they have a full development environment (although packages without source code can typically be installed out of the box). If `binary = TRUE`, the package will have a platform specific extension (e.g. .zip for windows), and will only be installable on the current platform, but no development environment is needed.

Usage

```r
build(path = ".", dest_path = NULL, binary = FALSE,
      vignettes = TRUE, manual = FALSE, clean_doc = NULL, args = NULL,
      quiet = FALSE, needs_compilation = pkg_has_src(path),
      compile_attributes = FALSE, register_routines = FALSE)
```

Arguments

- **path**  Path to a package, or within a package.
- **dest_path**  path in which to produce package. If NULL, defaults to the parent directory of the package.
- **binary**  Produce a binary (--binary) or source ( --no-manual --no-resave-data) version of the package.
- **vignettes, manual**  For source packages: if FALSE, don’t build PDF vignettes (--no-build-vignettes) or manual (--no-manual).
- **clean_doc**  If TRUE, clean the files in inst/doc before building the package. If NULL and interactive, ask to remove the files prior to cleaning. In most cases cleaning the files is the correct behavior to avoid stale vignette outputs in the built package.
- **args**  An optional character vector of additional command line arguments to be passed to R CMD build if binary = FALSE, or R CMD install if binary = TRUE.
- **quiet**  if TRUE suppresses output from this function.
- **needs_compilation**  Usually only needed if the packages has C/C++/Fortran code. By default this is autodetected.
clean_dll

compile_attributes
if TRUE and the package uses Rcpp, call Rcpp::compileAttributes() before building the package. It is ignored if package does not need compilation.

register_routines
if TRUE and the package does not use Rcpp, call register routines with tools::package_native_routine_registration_skeleton() before building the package. It is ignored if package does not need compilation.

Value
a string giving the location (including file name) of the built package

---

clean_dll  Remove compiled objects from /src/ directory

Description
Invisibly returns the names of the deleted files.

Usage
clean_dll(path = ".")

Arguments
path Path to a package, or within a package.

See Also
compile_dll()

compiler_flags  Default compiler flags used by devtools.

Description
These default flags enforce good coding practice by ensuring that CFLAGS and CXXFLAGS are set to -Wall -pedantic. These tests are run by cran and are generally considered to be good practice.

Usage
compiler_flags(debug = FALSE)

Arguments
debug If TRUE adds -g -00 to all flags (Adding FFLAGS and FCFLAGS
Details
By default `compile_dll()` is run with `compiler_flags(TRUE)`, and check with `compiler_flags(FALSE)`. If you want to avoid the possible performance penalty from the debug flags, install the package.

See Also
Other debugging flags: `with_debug`

Examples
```r
compiler_flags()
compiler_flags(TRUE)
```

---

**compile_dll**

*Compile a .dll/.so from source.*

Description
`compile_dll` performs a fake R CMD install so code that works here should work with a regular install (and vice versa). During compilation, debug flags are set with `compiler_flags(TRUE)`.

Usage
```r
compile_dll(path = ".", force = FALSE,
            compile_attributes = pkg_links_to_rcpp(path),
            register_routines = FALSE, quiet = FALSE)
```

Arguments
- `path` Path to a package, or within a package.
- `force` If TRUE, for compilation even if `needs_compile()` is FALSE.
- `compile_attributes` if TRUE and the package uses Rcpp, call `Rcpp::compileAttributes()` before building the package. It is ignored if package does not need compilation.
- `register_routines` if TRUE and the package does not use Rcpp, call register routines with `tools::package_native_routine_registration_skeleton()` before building the package. It is ignored if package does not need compilation.
- `quiet` if TRUE suppresses output from this function.

Details
Invisibly returns the names of the DLL.

Note
If this is used to compile code that uses Rcpp, you will need to add the following line to your `Makevars` file so that it knows where to find the Rcpp headers: `PKG_CPPFLAGS=$(R_HOME)/bin/Rscript -e 'Rcpp::CxxFlags()'"
has_build_tools

See Also
clean_dll() to delete the compiled files.

has_build_tools

Description

has_build_tools returns a logical, check_build_tools throws an error. with_build_tools checks that build tools are available, then runs code in an correctly staged environment. If run interactively from RStudio, and the build tools are not available these functions will trigger an automated install.

Usage

has_build_tools(debug = FALSE)
check_build_tools(debug = FALSE, quiet = FALSE)
with_build_tools(code, debug = FALSE, required = TRUE)
local_build_tools(debug = FALSE, required = TRUE,
                   .local_envir = parent.frame())

Arguments

d debug If TRUE, will print out extra information useful for debugging. If FALSE, it will use result cached from a previous run.

quiet if TRUE suppresses output from this function.

code Code to rerun in environment where build tools are guaranteed to exist.

required If TRUE, and build tools are not available, will throw an error. Otherwise will attempt to run code without them.

.local_envir [environment]
The environment to use for scoping.

Details

Errors like running command "C:\PROGRA~1/R/R-34~1.2/bin/x64/R" CMD config CC' had status 127 indicate the code expected Rtools to be on the system PATH. You can then verify you have rtools installed with has_build_tools() and temporarily add Rtools to the PATH with_build_tools({
code }).

It is possible to add Rtools to your system PATH manually; you can use rtools_path() to show the installed location. However because this requires manual updating when a new version of Rtools is installed and the binaries in Rtools may conflict with existing binaries elsewhere on the PATH it is better practice to use with_build_tools() as needed.
has_compiler

See Also

has_rtools

Examples

has_build_tools(debug = TRUE)
check_build_tools()

has_compiler(debug = FALSE)
check_compiler(debug = FALSE)

Arguments

debug If TRUE, will print out extra information useful for debugging. If FALSE, it will use result cached from a previous run.

Description

has_devel returns TRUE or FALSE. check_devel throws an error if you don’t have developer tools installed. Implementation based on a suggestion by Simon Urbanek. End-users (particularly those on Windows) should generally run check_build_tools() rather than check_compiler().

Usage

has_compiler(debug = FALSE)
check_compiler(debug = FALSE)

See Also

check_build_tools()

Examples

has_compiler()
check_compiler()

with_build_tools(has_compiler())
has_latex

Is latex installed?

Description
Checks for presence of pdflatex on path.

Usage

has_latex()
check_latex()

pkgbuild_process

Build package in the background

Description
This R6 class is a counterpart of the build() function, and represents a background process that builds an R package.

Usage

bp <- pkgbuild_process$new(path = ",", dest_path = NULL,
  binary = FALSE, vignettes = TRUE, manual = FALSE, args = NULL)
bp$get_dest_path()

Other methods are inherited from callr::rcmd_process and processx::process.

Arguments
See the corresponding arguments of build().

Details
Most methods are inherited from callr::rcmd_process and processx::process.
bp$get_dest_path() returns the path to the built package.

Examples

## Here we are just waiting, but in a more realistic example, you
## would probably run some other code instead...
bp <- pkgbuild_process$new("mypackage", dest_path = tempdir())
bp$is_alive()
bp$get_pid()
bp$wait()
pkg_has_src  

**Does a source package have src/ directory?**

**Description**

If it does, you definitely need build tools.

**Usage**

```r
pkg_has_src(path = ".")
```

**Arguments**

- `path`  
  Path to package (or directory within package).

rcmd_build_tools  

**Call R CMD <command> with build tools active**

**Description**

This is a wrapper around `callr::rcmd_safe()` that checks that you have build tools available, and on Windows, automatically sets the path to include Rtools.

**Usage**

```r
rcmd_build_tools(..., env = character(), required = TRUE, quiet = FALSE)
```

**Arguments**

- `...`  
  Parameters passed on to `rcmd_safe`.
- `env`  
  Additional environment variables to set. The defaults from `callr::rcmd_safe_env()` are always set.
- `required`  
  If TRUE, and build tools are not available, will throw an error. Otherwise will attempt to run code without them.
- `quiet`  
  if TRUE suppresses output from this function.
Examples

```r
# These env vars are always set
callr::rcmd_safe_env()

if (has_build_tools()) {
  rcmd_build_tools("CONFIG", "CC")$stdout
  rcmd_build_tools("CC", "--version")$stdout
}
```

---

**without_compiler**  
*Tools for testing pkgbuild*

**without_cache**

Description

`without_compiler` temporarily disables code compilation by setting `CC`, `CXX`, `makevars` to `test`. `without_cache` resets the cache before and after running code.

Usage

```r
without_compiler(code)
without_cache(code)
```

Arguments

code  
Code to execute with broken compilers

---

**with_debug**  
*Temporarily set debugging compilation flags.*

Description

Temporarily set debugging compilation flags.

Usage

```r
with_debug(code, CFLAGS = NULL, CXXFLAGS = NULL, FFLAGS = NULL, 
            FCFLAGS = NULL, debug = TRUE)
```

Arguments

code  
to execute.
CFLAGS  
flags for compiling C code
CXXFLAGS  
flags for compiling C++ code
FFLAGS  
flags for compiling Fortran code.
FCFLAGS  
flags for Fortran 9x code.
debug  
If TRUE adds `-g -O0` to all flags (Adding `FFLAGS` and `FCFLAGS`)
See Also

Other debugging flags: compiler_flags

Examples

flags <- names(compiler_flags(TRUE))
with_debug(Sys.getenv(flags))

## Not run:
install("mypkg")
with_debug(install("mypkg"))

## End(Not run)
Index

build, 2
build(), 7

callr::rcmd_process, 7
callr::rcmd_safe(), 8
callr::rcmd_safe_env(), 8
check_build_tools (has_build_tools), 5
check_build_tools(), 6
check_compiler (has_compiler), 6
check_compiler(), 6
check_latex (has_latex), 7
clean_dll, 3
clean_dll(), 5
compile_dll, 4
compile_dll(), 3, 4
compiler_flags, 3, 4, 10

has_build_tools, 5
has_compiler, 6
has_devel (has_compiler), 6
has_latex, 7

local_build_tools (has_build_tools), 5

needs_compile(), 4

pkg_has_src, 8
pkgbuild_process, 7
processx::process, 7

rcmd_build_tools, 8
Rcpp::compileAttributes(), 3, 4
rtools_path(), 5

with_build_tools (has_build_tools), 5
with debug, 4, 9
without_cache (without_compiler), 9
without Compiler, 9