

# Package: quickr (via r-universe)

May 8, 2026

**Title** Compiler for R

**Version** 0.3.0.9000

**Description** Compile R functions annotated with type and shape declarations for extremely fast performance and robust runtime type checking. Supports both just-in-time (JIT) and ahead-of-time (AOT) compilation. Compilation is performed by lowering R code to Fortran.

**License** MIT + file LICENSE

**URL** <https://github.com/t-kalinowski/quickr>

**BugReports** <https://github.com/t-kalinowski/quickr/issues>

**Depends** R (>= 4.3.0)

**Imports** dotty, glue, S7

**Suggests** bench, cli, pkgload (>= 1.4.0), rlang, testthat (>= 3.0.0), withr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Config/testthat/start-first** unary-intrinsics, loops

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Repository** <https://fastverse.r-universe.dev>

**Date/Publication** 2026-05-08 17:16:51 UTC

**RemoteUrl** <https://github.com/t-kalinowski/quickr>

**RemoteRef** HEAD

**RemoteSha** 836b3f9fd4340548f70a0c54991e80151d85bfe1

## Contents

compile_package . . . . .	2
quick . . . . .	2

**Index****6**


---

compile_package	<i>Compile all quick() functions in a package.</i>
-----------------	--

---

**Description**

This will compile all quick() functions in an R package, and generate source files in the src/ directory.

**Usage**

```
compile_package(path = ".")
```

**Arguments**

path	Path to an R package
------	----------------------

**Details**

Note, this function is automatically invoked during a pkgload::load\_all() call.

**Value**

Called for its side effect.

---

quick	<i>Compile a Quick Function</i>
-------	---------------------------------

---

**Description**

Compile an R function.

**Usage**

```
quick(fun, name = NULL)
```

**Arguments**

fun	An R function
name	String, name to use for the function. This is optional in regular usage but required in an R package. As a convenience, arguments fun and name can also be supplied as positional arguments to quick with name in the first position.

**Details**

`declare(type())` **syntax::**

The shape and mode of all function arguments must be declared. Local and return variables may optionally also be declared.

`declare(type())` also has support for declaring size constraints, or size relationships between variables. Here are some examples of declare calls:

```
declare(type(x = double(NA))) # x is a 1-d double vector of any length
declare(type(x = double(10))) # x is a 1-d double vector of length 10
declare(type(x = double(1))) # x is a scalar double
```

```
declare(type(x = integer(2, 3))) # x is a 2-d integer matrix with dim (2, 3)
declare(type(x = integer(NA, 3))) # x is a 2-d integer matrix with dim (<any>, 3)
```

```
# x is a 4-d logical matrix with dim (<any>, 24, 24, 3)
declare(type(x = logical(NA, 24, 24, 3)))
```

```
# x and y are 1-d double vectors of any length
declare(type(x = double(NA)),
        type(y = double(NA)))
```

```
# x and y are 1-d double vectors of the same length
declare(
  type(x = double(n)),
  type(y = double(n)),
)
```

```
# x and y are 1-d double vectors, where length(y) == length(x) + 2
declare(type(x = double(n)),
        type(y = double(n+2)))
```

You can provide declarations to `declare()` as:

- Multiple arguments to a single `declare()` call
- Separate `declare()` calls
- Multiple arguments within a code block (`{}`) inside `declare()`

```
declare(
  type(x = double(n)),
  type(y = double(n)),
)
```

```
declare(type(x = double(n)))
declare(type(y = double(n)))
```

```
declare({
  type(x = double(n))
  type(y = double(n))
})
```

**Return values:**

The shape and type of a function return value must be known at compile time. In most situations, this will be automatically inferred by `quick()`. However, if the output is dynamic, then you may need to provide a hint. For example, returning the result of `seq()` will fail because the output shape cannot be inferred.

```
# Will fail to compile:
quick_seq <- quick(function(start, end) {
  declare({
    type(start = integer(1))
    type(end = integer(1))
  })
  out <- seq(start, end)
  out
})
```

However, if the output size can be declared as a dynamic expression using other values known at runtime, compilation will succeed:

```
# Succeeds:
quick_seq <- quick(function(start, end) {
  declare({
    type(start = integer(1))
    type(end = integer(1))
    type(out = integer(end - start + 1))
  })
  out <- seq(start, end)
  out
})
quick_seq(1L, 5L)
```

**Fortran compiler:**

`quickr` compiles via R CMD SHLIB and will normally use the same toolchain that R was built/configured with.

`quickr` only uses LLVM `flang` when it is explicitly requested or, on macOS, when `flang` is available on PATH (and `flang --version` succeeds). If `flang` is requested but unavailable, compilation errors. If `flang` compilation fails, `quickr` retries with the default toolchain; on success it emits a one-time warning and disables automatic `flang` preference for the rest of the session.

In interactive use, you can explicitly control this with:

```
options(quickr.fortran_compiler = "flang")
```

To disable the macOS auto-preference, set `options(quickr.fortran_compiler = "gfortran")`.

**Value**

A quicker R function.

**Examples**

```
add_ab <- quick(function(a, b) {  
  declare(type(a = double(n)),  
          type(b = double(n)))  
  a + b  
})  
add_ab(1, 2)  
add_ab(c(1, 2, 3), c(4, 5, 6))
```

# Index

`compile_package`, [2](#)

`quick`, [2](#)