

# Package: StanHeaders (via r-universe)

August 13, 2024

**Title** C++ Header Files for Stan

**Version** 2.35.0.9000

**URL** <https://mc-stan.org/>

**BugReports** <https://github.com/stan-dev/rstan/issues>

**Description** The C++ header files of the Stan project are provided by this package, but it contains little R code or documentation. The main reference is the vignette. There is a shared object containing part of the 'CVODES' library, but its functionality is not accessible from R. 'StanHeaders' is primarily useful for developers who want to utilize the 'LinkingTo' directive of their package's DESCRIPTION file to build on the Stan library without incurring unnecessary dependencies. The Stan project develops a probabilistic programming language that implements full or approximate Bayesian statistical inference via Markov Chain Monte Carlo or 'variational' methods and implements (optionally penalized) maximum likelihood estimation via optimization. The Stan library includes an advanced automatic differentiation scheme, 'templated' statistical and linear algebra functions that can handle the automatically 'differentiable' scalar types (and doubles, 'ints', etc.), and a parser for the Stan language. The 'rstan' package provides user-facing R functions to parse, compile, test, estimate, and analyze Stan models.

**Imports** RcppParallel (>= 5.1.4)

**Suggests** Rcpp, BH (>= 1.75.0-0), knitr (>= 1.36), rmarkdown, Matrix, methods, rstan, withr

**LinkingTo** RcppEigen (>= 0.3.3.9.3), RcppParallel (>= 5.1.4)

**VignetteBuilder** knitr

**SystemRequirements** GNU make, pandoc

**Depends** R (>= 3.4.0)

**License** BSD\_3\_clause + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.1.2

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

**RemoteUrl** <https://github.com/stan-dev/rstan>

**RemoteRef** experimental

**RemoteSha** 2673051b274a530bbb68ac053d7d5a6d5dc2edec

## Contents

CxxFlags . . . . .	2
stanFunction . . . . .	3
<b>Index</b>	<b>5</b>

---

CxxFlags	<i>Compilation flags for StanHeaders</i>
----------	--

---

### Description

Output the compiler or linker flags required to build with the **StanHeaders** package

### Usage

```
CxxFlags(as_character = FALSE)
LdFlags(as_character = FALSE)
```

### Arguments

**as\_character** A logical scalar that defaults to **FALSE** that indicates whether to return the compiler or linker flags as a **character** vector of length one. Otherwise, the compiler or linker flags are merely output to the screen, which is appropriate when called from a Makevars or Makevars.win file

### Details

These functions are currently not exported and are typically called from a Makevars or a Makevars.win file of another package as follows:

```
PKG_CXXFLAGS += $(shell "${R_HOME}/bin/Rscript" -e "StanHeaders::CxxFlags()") PKG_LDLIBS
+= $(shell "${R_HOME}/bin/Rscript" -e "StanHeaders::LdFlags()")
```

### Value

If **as\_character** is **TRUE**, then these functions return a character vector of length one. Otherwise, (which is the default) these functions return **NULL** invisibly after outputting the compiler or linker flags to the screen.

**Description**

Call a function defined in the Stan Math Library from R using this wrapper around [cppFunction](#).

**Usage**

```
stanFunction(function_name, ..., env = parent.frame(), rebuild = FALSE,
             cacheDir = getOption("rcpp.cache.dir", tempdir()),
             showOutput = verbose, verbose = getOption("verbose"))
```

**Arguments**

`function_name` A [character](#) vector of length one that is the unscoped basename of a C++ function under the `prim/` directory of the Stan Math Library that you would like to evaluate. Functions (such as `integrate_1d`) of other functions are not permitted and neither are functions (such as `reject`) of characters.

`...` Further arguments that are passed to `function_name` in `tag = value` form, which are passed to `function_name` by *position*. See the [Details](#) and [Examples](#) sections.

`env`, `rebuild`, `cacheDir`, `showOutput`, `verbose`  
The same as in [cppFunction](#)

**Details**

The `stanFunction` function essentially compiles and evaluates a C++ function of the form

```
auto function_name(...) { return stan::math::function_name(...); }
```

It is essential to pass all arguments to `function_name` through the `...` in order for the C++ wrapper to know what the argument types are. The mapping between R types and Stan types is

<b>R type</b>	<b>Stan type</b>
double	real
integer	int
complex	complex
vector	vector or complex_vector
matrix(*, nrow = 1)	row_vector or complex_row_vector
matrix	matrix or complex_matrix

and, in addition, lists of the aforementioned R types map to arrays of Stan types and thus must not be ragged if they are nested. The Stan version of the function is called with arguments specified by position, i.e. in the order that they appear in the `...`. However, the R wrapper function has arguments whose names are the same as the names passed through the `...`.

**Value**

The result of `function_name` evaluated at the arguments that are passed through the `...`, which could be of various R types. It also has the side effect of defining a function named `function_name` in the environment given by the `env` argument that can subsequently be called with inputs of the same type (but not necessarily the same value) that were passed through the `...`.

**Examples**

```
files <- dir(system.file("include", "stan", "math", "prim",
                        package = "StanHeaders"),
            pattern = "hpp$", recursive = TRUE)
functions <- sub("\\.hpp$", "",
                sort(unique(basename(files[dirname(files) != "."]))))
length(functions) # you could call most of these Stan functions

## Not run:
log(sum(exp(exp(1)), exp(pi))) # true value

stanFunction("log_sum_exp", x = exp(1), y = pi)
args(log_sum_exp) # now exists in .GlobalEnv
log_sum_exp(x = pi, y = exp(1))

# but log_sum_exp() was not defined for a vector or matrix
x <- c(exp(1), pi)
try(log_sum_exp(x))
stanFunction("log_sum_exp", x = x) # now it is

# log_sum_exp() is now also defined for a matrix
log_sum_exp(as.matrix(x))
log_sum_exp(t(as.matrix(x)))
log_sum_exp(rbind(x, x))

# but log_sum_exp() was not defined for a list
try(log_sum_exp(as.list(x)))
stanFunction("log_sum_exp", x = as.list(x)) # now it is

# in rare cases, passing a nested list is needed
stanFunction("dims", x = list(list(1:3)))

# functions of complex arguments work
stanFunction("eigenvalues", # different ordering than base:eigen()
            x = matrix(complex(real = 1:9, imaginary = pi),
                       nrow = 3, ncol = 3))

# nullary functions work but are not that interesting
stanFunction("negative_infinity")

# PRNG functions work by adding a seed argument
stanFunction("lkj_corr_rng", K = 3L, eta = 1)
args(lkj_corr_rng) # has a seed argument

## End(Not run)
```

# Index

[character](#), [2](#), [3](#)  
[cppFunction](#), [3](#)  
[CxxFlags](#), [2](#)

[FALSE](#), [2](#)

[LdFlags \(CxxFlags\)](#), [2](#)

[NULL](#), [2](#)

[stanFunction](#), [3](#)

[TRUE](#), [2](#)