

# Package ‘treesitter.c’

February 8, 2026

**Title** 'R' Bindings to the 'C' Grammar for Tree-Sitter

**Version** 0.0.4.2

**Description** Provides bindings to a 'C' grammar for Tree-sitter, to be used alongside the 'treesitter' package. Tree-sitter builds concrete syntax trees for source files and can efficiently update them or generate code like producing R C API wrappers from C functions, structs and global definitions from header files.

**License** GPL-3

**Depends** R (>= 4.3.0)

**Imports** treesitter

**Suggests** tinytest

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**URL** <https://sounkou-bioinfo.github.io/treesitter.c/>,  
<https://github.com/sounkou-bioinfo/treesitter.c>

**BugReports** <https://github.com/sounkou-bioinfo/treesitter.c/issues>

**NeedsCompilation** yes

**Author** Sounkou Mahamane Toure [aut, cre],  
Tree-sitter authors [cph] (Tree-sitter C grammar),  
Eli Bendersky and Co-authors [cph] (pycparser fake\_libc headers)

**Maintainer** Sounkou Mahamane Toure <sounkoutoure@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-02-08 10:50:02 UTC

## Contents

|                                      |   |
|--------------------------------------|---|
| fake_libc_path . . . . .             | 2 |
| get_defines_from_file . . . . .      | 2 |
| get_enum_members_from_root . . . . . | 3 |
| get_enum_nodes . . . . .             | 3 |
| get_function_nodes . . . . .         | 4 |

|                                            |           |
|--------------------------------------------|-----------|
| get_globals_from_root . . . . .            | 4         |
| get_globals_with_types_from_root . . . . . | 5         |
| get_struct_members . . . . .               | 5         |
| get_struct_nodes . . . . .                 | 6         |
| get_union_members_from_root . . . . .      | 6         |
| get_union_nodes . . . . .                  | 7         |
| language . . . . .                         | 7         |
| parse_headers_collect . . . . .            | 8         |
| parse_header_text . . . . .                | 9         |
| parse_r_include_headers . . . . .          | 10        |
| preprocess_header . . . . .                | 11        |
| preprocess_headers . . . . .               | 12        |
| r_cc . . . . .                             | 13        |
| <b>Index</b>                               | <b>14</b> |

---

|                |                                                        |
|----------------|--------------------------------------------------------|
| fake_libc_path | <i>Get the path to the installed fake_libc headers</i> |
|----------------|--------------------------------------------------------|

---

**Description**

Returns the absolute path to the inst/fake\_libc directory in the installed package.

**Usage**

fake\_libc\_path()

**Value**

Character scalar with the path to fake\_libc headers

---

|                       |                                                                                                                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| get_defines_from_file | <i>This function will use the configured C compiler to list macro definitions (-dM -E) if use_cpp = TRUE and a compiler is available; otherwise, a simple scan of #define lines is used as a fallback.</i> |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

**Description**

This function will use the configured C compiler to list macro definitions (-dM -E) if use\_cpp = TRUE and a compiler is available; otherwise, a simple scan of #define lines is used as a fallback.

**Usage**

get\_defines\_from\_file(file, use\_cpp = TRUE, cc = r\_cc(), ccflags = NULL)

**Arguments**

|         |                                                       |
|---------|-------------------------------------------------------|
| file    | Path to a header file                                 |
| use_cpp | Logical; use the C preprocessor if available          |
| cc      | Compiler string; passed to system2 if use_cpp = TRUE. |
| ccflags | Additional flags for the compiler                     |

**Value**

Character vector of macro names defined in file

---

get\_enum\_members\_from\_root

*Extract enum members from a parsed header*

---

**Description**

Extract enum members from a parsed header

**Usage**

```
get_enum_members_from_root(root)
```

**Arguments**

|      |                          |
|------|--------------------------|
| root | A tree-sitter root node. |
|------|--------------------------|

**Value**

Data frame with enum member names and values.

---

get\_enum\_nodes

*Extract enum names from a parsed header*

---

**Description**

Extract enum names from a parsed header

**Usage**

```
get_enum_nodes(root)
```

**Arguments**

|      |                          |
|------|--------------------------|
| root | A tree-sitter root node. |
|------|--------------------------|

**Value**

Data frame with enum names.

---

|                    |                                                                          |
|--------------------|--------------------------------------------------------------------------|
| get_function_nodes | <i>Extract function names (declarations and definitions) from a root</i> |
|--------------------|--------------------------------------------------------------------------|

---

**Description**

Returns a data frame with capture\_name, text, start\_line, and start\_col.

**Usage**

```
get_function_nodes(root, extract_params = FALSE, extract_return = FALSE)
```

**Arguments**

root                    A tree-sitter root node.

extract\_params   Logical; whether to extract parameter types for found functions. Default FALSE.

extract\_return   Logical; whether to extract return types for found functions. Default FALSE.

**Value**

Data frame with function captures; when extract\_params=TRUE a params list-column is present.

---

|                       |                                                              |
|-----------------------|--------------------------------------------------------------|
| get_globals_from_root | <i>Extract global variable names from a parsed tree root</i> |
|-----------------------|--------------------------------------------------------------|

---

**Description**

Extract global variable names from a parsed tree root

**Usage**

```
get_globals_from_root(root)
```

**Arguments**

root                    A tree-sitter root node.

**Value**

Data frame with top-level global names.

---

`get_globals_with_types_from_root`*Extract global variables with types from a parsed tree root*

---

**Description**

Extract global variables with types from a parsed tree root

**Usage**

```
get_globals_with_types_from_root(root)
```

**Arguments**

|                   |                          |
|-------------------|--------------------------|
| <code>root</code> | A tree-sitter root node. |
|-------------------|--------------------------|

**Value**

Data frame with global names and C types.

---

`get_struct_members`*Extract members of structs (including nested anonymous struct members)*

---

**Description**

Extract members of structs (including nested anonymous struct members)

**Usage**

```
get_struct_members(root)
```

**Arguments**

|                   |                          |
|-------------------|--------------------------|
| <code>root</code> | A tree-sitter root node. |
|-------------------|--------------------------|

**Value**

Data frame describing struct members, including bitfields.

---

|                  |                                                     |
|------------------|-----------------------------------------------------|
| get_struct_nodes | <i>Extract struct names from a parsed tree root</i> |
|------------------|-----------------------------------------------------|

---

**Description**

Extract struct names from a parsed tree root

**Usage**

get\_struct\_nodes(root)

**Arguments**

root                    A tree-sitter root node from parse\_header\_text().

**Value**

Data frame with struct name captures.

---

|                             |                                  |
|-----------------------------|----------------------------------|
| get_union_members_from_root | <i>Extract members of unions</i> |
|-----------------------------|----------------------------------|

---

**Description**

Extract members of unions

**Usage**

get\_union\_members\_from\_root(root)

**Arguments**

root                    A tree-sitter root node.

**Value**

Data frame describing union members, including bitfields.

---

|                 |                                                 |
|-----------------|-------------------------------------------------|
| get_union_nodes | <i>Extract union names from a parsed header</i> |
|-----------------|-------------------------------------------------|

---

**Description**

Extract union names from a parsed header

**Usage**

```
get_union_nodes(root)
```

**Arguments**

|      |                          |
|------|--------------------------|
| root | A tree-sitter root node. |
|------|--------------------------|

**Value**

Data frame with union names.

---

|          |                                   |
|----------|-----------------------------------|
| language | <i>tree-sitter language for C</i> |
|----------|-----------------------------------|

---

**Description**

language() returns a tree\_sitter\_language object for C for use with the treesitter package.

**Usage**

```
language()
```

**Value**

A tree\_sitter\_language object.

**Examples**

```
language()
```

---

`parse_headers_collect` *Parse a directory of headers and return named list of data.frames with*

---

## Description

functions, structs, struct members, enums, unions, globals, and macros.

## Usage

```
parse_headers_collect(
  dir = R.home("include"),
  recursive = TRUE,
  pattern = c("\\.h$", "\\..H$"),
  preprocess = FALSE,
  cc = r_cc(),
  ccflags = NULL,
  include_dirs = NULL,
  extract_params = FALSE,
  extract_return = FALSE,
  ...
)
```

## Arguments

|                             |                                                                                                                                                                        |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dir</code>            | Directory to search for header files. Defaults to <code>R.home("include")</code> .                                                                                     |
| <code>recursive</code>      | Whether to search recursively for headers. Default <code>TRUE</code> .                                                                                                 |
| <code>pattern</code>        | File name pattern to match header files. Default is <code>\\.h\$</code> and <code>\\.H\$</code> .                                                                      |
| <code>preprocess</code>     | Run the C preprocessor (using R's configured CC) on header files before parsing. Defaults to <code>FALSE</code> .                                                      |
| <code>cc</code>             | The C compiler to use for preprocessing. If <code>NULL</code> the function queries R CMD config CC and falls back to <code>Sys.getenv("CC")</code> and the cc on PATH. |
| <code>ccflags</code>        | Extra flags to pass to the compiler when preprocessing. If <code>NULL</code> flags are taken from R CMD config CFLAGS and R CMD config CPPFLAGS.                       |
| <code>include_dirs</code>   | Additional directories to add to the include path for preprocessing. A character vector of directories.                                                                |
| <code>extract_params</code> | Logical; whether to extract parameter types for functions. Default <code>FALSE</code> .                                                                                |
| <code>extract_return</code> | Logical; whether to extract return types for functions. Default <code>FALSE</code> .                                                                                   |
| <code>...</code>            | Additional arguments passed to <code>preprocess_header</code> (e.g., extra compiler flags)                                                                             |

## Details

This helper loops over headers found in a directory and returns a list with tidy data.frames. Useful for programmatic analysis of header collections.



**Value**

A named list of data frames with components: functions, structs, struct\_members, enums, unions, globals, defines.

**Examples**

```
## Not run:
if (requireNamespace("treesitter", quietly = TRUE)) {
  res <- parse_headers_collect(dir = R.home("include"), preprocess = FALSE)
  head(res$functions)
}

## End(Not run)
```

---

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| parse_header_text | <i>Convert character content of a header file into a tree-sitter root</i> |
|-------------------|---------------------------------------------------------------------------|

---

**Description**

Convert character content of a header file into a tree-sitter root

**Usage**

```
parse_header_text(text, lang = language())
```

**Arguments**

|      |                                                                |
|------|----------------------------------------------------------------|
| text | Character scalar with the header content to parse.             |
| lang | tree-sitter language object (default: C language from package) |

**Value**

The tree root node object

**Examples**

```
if (requireNamespace("treesitter", quietly = TRUE)) {
  root <- parse_header_text("int foo(int);\n")
  root
}
```

---

 parse\_r\_include\_headers

*Parse C header files for function declarations using tree-sitter*


---

## Description

This utility uses the C language provided by this package and the treesitter parser to find function declarations and definitions in C header files. The default dir is `R.home("include")`, which is typically where R's headers live.

## Usage

```
parse_r_include_headers(
  dir = R.home("include"),
  recursive = TRUE,
  pattern = c("\\.h$", "\\..H$"),
  preprocess = FALSE,
  cc = r_cc(),
  ccflags = NULL,
  include_dirs = NULL,
  ...
)
```

## Arguments

|                           |                                                                                                                                                           |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dir</code>          | Directory to search for header files. Defaults to <code>R.home("include")</code> .                                                                        |
| <code>recursive</code>    | Whether to search recursively for headers. Default TRUE.                                                                                                  |
| <code>pattern</code>      | File name pattern to match header files. Default is <code>\\.h\$</code> and <code>\\.H\$</code> .                                                         |
| <code>preprocess</code>   | Run the C preprocessor (using R's configured CC) on header files before parsing. Defaults to FALSE.                                                       |
| <code>cc</code>           | The C compiler to use for preprocessing. If NULL the function queries R CMD config CC and falls back to <code>Sys.getenv("CC")</code> and the cc on PATH. |
| <code>ccflags</code>      | Extra flags to pass to the compiler when preprocessing. If NULL flags are taken from R CMD config CFLAGS and R CMD config CPPFLAGS.                       |
| <code>include_dirs</code> | Additional directories to add to the include path for preprocessing. A character vector of directories.                                                   |
| <code>...</code>          | Arguments passed on to <a href="#">parse_headers_collect</a>                                                                                              |
|                           | <code>extract_params</code> Logical; whether to extract parameter types for functions. Default FALSE.                                                     |
|                           | <code>extract_return</code> Logical; whether to extract return types for functions. Default FALSE.                                                        |

## Value

A data frame with columns name, file, line, and kind (either 'declaration' or 'definition').

**Examples**

```

if (requireNamespace("treesitter", quietly = TRUE)) {
  # Parse a small header file from a temp dir
  tmp <- tempdir()
  path <- file.path(tmp, "example.h")
  writeLines(c(
    "int foo(int a);",
    "static inline int bar(void) { return 1; }"
  ), path)
  parse_r_include_headers(dir = tmp)
}

```

---

```
preprocess_header      Run the C preprocessor on file using the provided compiler
```

---

**Description**

This function runs the configured C compiler with the `-E` preprocessor flag and returns the combined preprocessed output as a single string.

**Usage**

```
preprocess_header(file, cc = r_cc(), ccflags = NULL, ...)
```

**Arguments**

|                        |                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------|
| <code>file</code>      | Path to a header file to preprocess.                                                           |
| <code>cc</code>        | (Character) Compiler command to use. If <code>NULL</code> , resolved via <code>r_cc()</code> . |
| <code>ccflags</code>   | (Character) Additional flags to pass to the compiler.                                          |
| <code>...</code>       | Arguments passed on to <a href="#">preprocess_headers</a>                                      |
| <code>dir</code>       | Directory where header files will be searched.                                                 |
| <code>recursive</code> | Logical; whether to search recursively.                                                        |
| <code>pattern</code>   | File name pattern(s) used to identify header files.                                            |

**Value**

Character scalar with the preprocessed output of `file`.

**Examples**

```

## Not run:
# Check for a compiler before running an example that invokes the preprocessor
rcc <- treesitter.c::r_cc()
if (nzchar(rcc)) {
  rcc_prog <- strsplit(rcc, "\\s+")[1][1]
  if (nzchar(Sys.which(rcc_prog))) {
    tmp <- tempfile("hdr3")

```

```

    dir.create(tmp)
    path <- file.path(tmp, "p.h")
    writeLines(c("#define TYPE int", "TYPE foo(TYPE x);"), path)
    out <- preprocess_header(path)
    grepl("int foo\\(", out)
  } else {
    message("Skipping preprocess example: compiler not found on PATH")
  }
}

## End(Not run)

```

---

```
preprocess_headers      Preprocess a set of header files found under dir
```

---

## Description

This helper calls `preprocess_header()` for each matching file in `dir` and returns a named list with the path as the keys and the preprocessed text as the values.

## Usage

```

preprocess_headers(
  dir = R.home("include"),
  recursive = TRUE,
  pattern = c("\\.h$", "\\..H$"),
  cc = r_cc(),
  ccflags = NULL,
  ...
)

```

## Arguments

|                           |                                                                                                         |
|---------------------------|---------------------------------------------------------------------------------------------------------|
| <code>dir</code>          | Directory where header files will be searched.                                                          |
| <code>recursive</code>    | Logical; whether to search recursively.                                                                 |
| <code>pattern</code>      | File name pattern(s) used to identify header files.                                                     |
| <code>cc</code>           | Compiler string; passed to <code>preprocess_header</code> .                                             |
| <code>ccflags</code>      | Compiler flags; passed to <code>preprocess_header</code> .                                              |
| <code>...</code>          | Arguments passed on to <a href="#">parse_r_include_headers</a>                                          |
| <code>preprocess</code>   | Run the C preprocessor (using R's configured CC) on header files before parsing. Defaults to FALSE.     |
| <code>include_dirs</code> | Additional directories to add to the include path for preprocessing. A character vector of directories. |

## Value

Named list of file => preprocessed text.

---

`r_cc`

---

*Return the default R-configured C compiler (possibly with flags)*

---

**Description**

This function queries common places to find the C compiler used by R. It checks `Sys.getenv("CC")`, then `R CMD config CC`, and finally `cc` on `PATH`. The returned value may include flags (e.g., `'gcc -std=...'`).

**Usage**

```
r_cc()
```

**Value**

Character scalar with compiler program (or empty string).

# Index

fake\_libc\_path, [2](#)

get\_defines\_from\_file, [2](#)  
get\_enum\_members\_from\_root, [3](#)  
get\_enum\_nodes, [3](#)  
get\_function\_nodes, [4](#)  
get\_globals\_from\_root, [4](#)  
get\_globals\_with\_types\_from\_root, [5](#)  
get\_struct\_members, [5](#)  
get\_struct\_nodes, [6](#)  
get\_union\_members\_from\_root, [6](#)  
get\_union\_nodes, [7](#)

language, [7](#)

parse\_header\_text, [9](#)  
parse\_headers\_collect, [8](#), [10](#)  
parse\_r\_include\_headers, [10](#), [12](#)  
preprocess\_header, [11](#)  
preprocess\_headers, [11](#), [12](#)

r\_cc, [13](#)