Package: treesitter.c (via r-universe)

June 7, 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

Roxygen list(markdown = TRUE)

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

Repository https://sounkou-bioinfo.r-universe.dev

Date/Publication 2026-02-07 20:54:49 UTC

RemoteUrl https://github.com/sounkou-bioinfo/treesitter.c

RemoteRef HEAD

RemoteSha 1018d57467ccdf0c6aa819b63f844d8082712bb9

Contents

`fake_libc_path` *Get the path to the installed fake_libc headers*

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` *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.*

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 *Extract function names (declarations and definitions) from a root*

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 *Extract global variable names from a parsed tree root*

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

root	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

root	A tree-sitter root node.

Value

Data frame describing struct members, including bitfields.

get_struct_nodes *Extract struct names from a parsed tree root*

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 *Extract members of unions*

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 *Extract union names from a parsed header*

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 *tree-sitter language for C*

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

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_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

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

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

file	Path to a header file to preprocess.
cc	(Character) Compiler command to use. If NULL, resolved via r_cc().
ccflags	(Character) Additional flags to pass to the compiler.
...	Arguments passed on to preprocess_headers dir Directory where header files will be searched. recursive Logical; whether to search recursively. pattern 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

`dir`	Directory where header files will be searched.
`recursive`	Logical; whether to search recursively.
`pattern`	File name pattern(s) used to identify header files.
`cc`	Compiler string; passed to `preprocess_header`.
`ccflags`	Compiler flags; passed to `preprocess_header`.
`...`	Arguments passed on to `parse_r_include_headers` `preprocess` Run the C preprocessor (using R's configured CC) on header files before parsing. Defaults to `FALSE`. `include_dirs` 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