Package: tsitter (via r-universe)

May 23, 2026

Title Tree-Sitter Parsing Tools

Version 0.0.0.9000

Description Common tree-sitter parsing tools for R. It is meant to be used by other packages that specialize in particular languages and file formats.

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports cli, utils

Suggests magrittr, pillar, testthat (>= 3.0.0), tsjsonc, tstoml, withr

Remotes gaborcsardi/tsjsonc, gaborcsardi/tstoml

Additional_repositories https://github.com/r-lib/tsitter/releases/download

Encoding UTF-8

URL https://github.com/r-lib/tsitter, https://r-lib.github.io/tsitter/

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

Config/testthat/edition 3

Config/Needs/website r-lib/asciicast, tidyverse/tidytemplate

Biarch true

Config/roxygen2/version 8.0.0

Repository https://r-lib.r-universe.dev

Date/Publication 2026-05-18 11:07:36 UTC

RemoteUrl https://github.com/r-lib/tsitter

RemoteRef HEAD

RemoteSha 904bbd778988520aa88688f5d2dca312da2a8bf0

Contents

[[.ts_tree . . . 2
[[<-.ts_tree . . . 3
about . . . 4
as.character.ts_tree . . . 7
as.raw.ts_tree . . . 8
format.ts_tree . . . 9
print.ts_tree . . . 10
select-set . . . 11
ts_list_parsers . . . 12
ts_tree-brackets . . . 13
ts_tree_ast . . . 14
ts_tree_delete . . . 15
ts_tree_dom . . . 16
ts_tree_format . . . 17
ts_tree_insert . . . 19
ts_tree_mark_selection1 . . . 20
ts_tree_new . . . 21
ts_tree_query . . . 23
ts_tree_select . . . 24
ts_tree_select1 . . . 25
ts_tree_selection . . . 28
ts_tree_sexpr . . . 29
ts_tree_unserialize . . . 30
ts_tree_update . . . 31
ts_tree_write . . . 32

Index **34**

`[[.ts_tree` *Unserialize parts of a tree-sitter tree*

Description

```
tsitter:::doc_insert("ts_tree_double_bracket_description")
```

Usage

```
## S3 method for class 'ts_tree'
x[[i, ...]]
```

Arguments

`x`	`tsitter:::doc_insert("tsitter::ts_tree_double_bracket_param_x")`
`i`	`tsitter:::doc_insert("tsitter::ts_tree_double_bracket_param_i")`
`...`	`tsitter:::doc_insert("tsitter::ts_tree_double_bracket_param_dots")`

Details

tsitter:::doc_insert("tsitter::ts_tree_double_bracket_details") tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::ts_tree_double_bracket_return")

See Also

Other ts_tree generics: [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Other serialization functions: ts_tree_unserialize()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"a": 13, "b": [1, 2, 3], "c": "x"}')

tree

tree[[list("a")]]

# Last two elements of "b"
tree[[list("b", -(1:2))]]
```

[[<-.ts_tree *Edit parts of a tree-sitter tree*

Description

tsitter:::doc_insert("ts_tree_double_bracket_set_description")

Usage

```
## S3 replacement method for class 'ts_tree'
x[[i]] <- value
```

Arguments

x	tsitter:::doc_insert("tsitter::ts_tree_double_bracket_set_param_x")
i	tsitter:::doc_insert("tsitter::ts_tree_double_bracket_set_param_i")
value	tsitter:::doc_insert("tsitter::ts_tree_double_bracket_set_param_value")

Details

tsitter:::doc_insert("tsitter::ts_tree_double_bracket_set_details") tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::ts_tree_double_bracket_set_return")

See Also

Other ts_tree generics: [[.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"a": 13, "b": [1, 2, 3], "c": "x"}')

tree

tree[[list("a")]] <- 42
tree[[list("b", -1)]] <- ts_tree_deleted()

tree
```

---

about *About tsitter*

Description

tsitter is a common interface to tree-sitter parsers, implemented in other R packages. It has a common API to

- query,
- edit,
- format, and
- unserialize

tree-sitter parse trees.

Details

In this document I show examples with the `tsjsonc` package.

Create a tree-sitter tree:
Create a ts_tree (ts_tree_jsonc) object from a string:

```
txt <- r"(
// this is a comment
{
  "a": {
    "a1": [1, 2, 3],
    // comment
    "a2": "string"
  },
  "b": [
    {
      "b11": true,
      "b12": false
    },
    {
      "b21": false,
      "b22": false
    }
  ]
}
)"

json <- tsjsonc::ts_parse_jsonc(txt)
```

Pretty print a ts_tree object:

```
json
```

Select nodes of a tree:
Selecting nodes is the basis of editing and querying tree-sitter trees.
Select element by objects key:

```
ts_tree_select(json, "a")
```

Select element inside element:

```
ts_tree_select(json, "a", "a1")
```

Select element(s) of an array:

```
ts_tree_select(json, "a", "a1", 1:2)
```

Select multiple keys from an object:

```
ts_tree_select(json, "a", c("a1", "a2"))
```

Select nodes that match a tree-sitter query:

```
json |> ts_tree_select(query = "((pair value: (false) @val))")
```

Delete elements:
Delete selected elements:

```
ts_tree_select(json, "a", "a1") |> ts_tree_delete()
```

Insert elements:
Insert element into an array:

```
ts_tree_select(json, "a", "a1") |> ts_tree_insert(at = 2, "new")
```

Inserting into an array reformats the array.
Insert element into an object, at the specified key:

```
ts_tree_select(json, "a") |>
  ts_tree_insert(key = "a0", at = 0, list("new", "element"))
```

Update elements:
Update existing element:

```
ts_tree_select(json, "a", c("a1", "a2")) |> ts_tree_update("new value")
```

Inserts the element if some parents are missing:

```
json <- ts_parse_jsonc(text = "{ \"a\": { \"b\": true } }")
json
```

```
ts_tree_select(json, "a", "x", "y") |> ts_tree_update(list(1,2,3))
```

Write out a document:
Use `stdout()` to write it to the screen instread of a file:

```
json |> ts_tree_write(stdout())
```

Formatting:
Format the whole document:

```
json |> ts_tree_format()
```

Format part of the document:

```
json |> ts_tree_select("a") |>
  ts_tree_format(options = list(format = "compact"))
```

Unserializing:
Unserialize a whole document:

```
json |> ts_tree_unserialize()
```

Note that `ts_tree_unserialize()` always returns a list, the first element of the list is the unserialized document.
Unserialize part(s) of the document:

```
json |> ts_tree_select("b") |> ts_tree_unserialize()
```

Again, `ts_tree_unserialize()` returns a list, with one element for each selected node.

Exploring a tree-sitter tree:
It is often useful to explore the structure of a (JSONC) tree-sitter tree, to help writing the right selection or tree-sitter queries.
Print the annotated syntax tree:

```
ts_tree_ast(json)
```

Print the document object model:

```
ts_tree_dom(json)
```

Print the structural summary of a tree:

```
ts_tree_sexpr(json)
```

Value

Not applicable.

Examples

```
# See above please.
```

---

as.character.ts_tree *The document of a tree-sitter tree as a character scalar*

Description

The document of a tree-sitter tree as a character scalar

Usage

```
## S3 method for class 'ts_tree'
as.character(x, ...)
```

Arguments

`x`	A `ts_tree` object.
`...`	Ignored.

Value

A character scalar containing the document of the tree.

See Also

`as.raw.ts_tree()` to get the document as a raw vector.

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"foo": 42, "bar": [1, 2, 3]}')

tree
as.character(tree)
```

---

as.raw.ts_tree *Raw bytes of a document of a tree-sitter tree*

Description

Raw bytes of a document of a tree-sitter tree

Usage

```
## S3 method for class 'ts_tree'
as.raw(x)
```

Arguments

x	A ts_tree object.

Value

A raw vector containing the bytes of the document of the tree.

See Also

`as.character.ts_tree()` to get the document as a character scalar.

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"foo": 42, "bar": [1, 2, 3]}')

tree
as.raw(tree)
```

format.ts_tree *Format tree-sitter trees*

Description

tsitter:::doc_insert("tsitter::format_description")

Usage

```
## S3 method for class 'ts_tree'
format(x, n = 10, ...)
```

Arguments

x	tsitter:::doc_insert("tsitter::format_param_x")
n	tsitter:::doc_insert("tsitter::format_param_n")
...	tsitter:::doc_insert("tsitter::format_param_dots")

Details

tsitter:::doc_insert("tsitter::format_details") tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::format_return")

See Also

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
json <- tsjsonc::ts_parse_jsonc(
  '{ "a": 1, "b": [10, 20, 30], "c": { "c1": true, "c2": 100 } }'
)
format(json)
```

print.ts_tree *Print a tree-sitter tree*

Description

tsitter:::doc_insert("tsitter::print_description")

Usage

```
## S3 method for class 'ts_tree'
print(x, n = 10, ...)
```

Arguments

x	tsitter:::doc_insert("tsitter::print_param_x")
n	tsitter:::doc_insert("tsitter::format_param_n")
...	tsitter:::doc_insert("tsitter::print_param_dots")

Details

tsitter:::doc_insert("tsitter::print_details")

Value

tsitter:::doc_insert("tsitter::print_return")

See Also

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
json <- tsjsonc::ts_parse_jsonc(
  '{ "a": 1, "b": [10, 20, 30], "c": { "c1": true, "c2": 100 } }'
)
print(json)
```

select-set *Edit parts of a tree-sitter tree*

Description

tsitter:::doc_insert("ts_tree_select_set_description")

Usage

```
ts_tree_select(tree, ...) <- value
```

Arguments

tree	tsitter:::doc_insert("ts_tree_select_set_param_tree")
...	tsitter:::doc_insert("ts_tree_select_set_param_dots")
value	tsitter:::doc_insert("ts_tree_select_set_param_value")

Details

tsitter:::doc_insert("tsitter::ts_tree_select_set_details") tsitter:::doc_extra()

ts_tree_deleted() is a special marker to delete elements from a ts_tree object with ts_tree_select<- or the double bracket operator.

Value

tsitter:::doc_insert("tsitter::ts_tree_select_set_return")

ts_tree_deleted() returns a marker object to be used at the right hand side of the ts_tree_select<- or the double bracket replacement functions, see examples below.

See Also

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"a": 13, "b": [1, 2, 3], "c": "x"}')
tree

ts_tree_select(tree, "a") <- 42
ts_tree_select(tree, "b", -1) <- ts_tree_deleted()

tree
```

`ts_list_parsers` *List installed tree-sitter parsers*

Description

The ts package contains a common interface to several tree-sitter parsers, implemented in other R packages. `ts_list_parsers()` lists the available parsers installed in the system.

Usage

```
ts_list_parsers(lib_path = .libPaths())
```

Arguments

`lib_path`	Library paths to search for installed packages. Default is `base::.libPaths()`.

Details

To see tree-sitter parser packages that are available on CRAN, but not installed on your system, see the packages that depend on ts and have a name with a 'ts' prefix.

Here is an example that includes all tree-sitter parsers at this time:

```
ts_list_parsers()
```

Value

A data frame with columns:

- `package`: character, the name of the package.
- `version`: character, the version of the package.
- `title`: character, the title of the package.
- `library`: character, the library path where the package is installed.
- `loaded`: logical, whether the package is currently loaded.

Examples

```
ts_list_parsers()
```

ts_tree-brackets *Convert ts_tree object to a data frame*

Description

```
tsitter:::doc_insert("ts_tree_brackets_description")
```

Usage

```
## S3 method for class 'ts_tree'
x[i, j, drop = FALSE]
```

Arguments

x	`tsitter:::doc_insert("tsitter::ts_tree_brackets_param_x")`
i, j	`tsitter:::doc_insert("tsitter::ts_tree_brackets_param_ij")`
drop	`tsitter:::doc_insert("tsitter::ts_tree_brackets_param_drop")`

Details

```
tsitter:::doc_insert("tsitter::ts_tree_brackets_details")
```

Value

```
tsitter:::doc_insert("tsitter::ts_tree_brackets_return")
```

See Also

```
tsitter:::doc_seealso("[")
```

Other ts_tree exploration: `ts_tree_ast()`, `ts_tree_dom()`, `ts_tree_query()`, `ts_tree_sexpr()`

Examples

```
# Create a parse tree with tsjsonc -----------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"foo": 42, "bar": [1, 2, 3]}')

tree

tree[]
```

ts_tree_ast *Show the annotated syntax tree of a tree-sitter tree*

Description

tsitter:::doc_insert("ts_tree_ast_description")

tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_ast")

Usage

```
ts_tree_ast(tree)
```

Arguments

tree	tsitter:::doc_insert("tsitter::ts_tree_ast_param_tree")

Details

tsitter:::doc_insert("tsitter::ts_tree_ast_details") tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::ts_tree_ast_return")

See Also

ts_tree_dom() to show the document object model (DOM) of a ts_tree object.

tsitter:::doc_seealso("ts_tree_ast")

Other ts_tree exploration: ts_tree-brackets, ts_tree_dom(), ts_tree_query(), ts_tree_sexpr()

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"foo": 42, "bar": [1, 2, 3]}')

tree

ts_tree_ast(tree)

ts_tree_dom(tree)
```

```
# Create a parse tree with tstoml -------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  title = "TOML Example"
  [owner]
  name = "Tom Preston-Werner"
  dob = 1979-05-27T07:32:00-08:00
)")

tree

ts_tree_ast(tree)

ts_tree_dom(tree)
```

`ts_tree_delete` *Delete selected elements from a tree-sitter tree*

Description

`tsitter:::doc_insert("ts_tree_delete_description")`

`tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_delete")`

Usage

```
ts_tree_delete(tree, ...)
```

Arguments

`tree`	`tsitter:::doc_insert("tsitter::ts_tree_delete_param_tree")`
`...`	Extra arguments for methods.

Details

`tsitter:::doc_insert("tsitter::ts_tree_delete_details")` `tsitter:::doc_extra()`

Value

`tsitter:::doc_insert("tsitter::ts_tree_delete_return")`

See Also

`tsitter:::doc_seealso("ts_tree_delete")`

Other ts_tree generics: `[[.ts_tree()`, `[[<-.ts_tree()`, `format.ts_tree()`, `print.ts_tree()`, `select-set`, `ts_tree_ast()`, `ts_tree_dom()`, `ts_tree_format()`, `ts_tree_insert()`, `ts_tree_new()`, `ts_tree_query()`, `ts_tree_select()`, `ts_tree_sexpr()`, `ts_tree_unserialize()`, `ts_tree_update()`, `ts_tree_write()`

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc(
  "{ \"a\": //comment\ntrue, \"b\": [1, 2, 3] }"
)

tree

tree |> ts_tree_select("a")

tree |> ts_tree_select("a") |> ts_tree_delete()


# Create a parse tree with tstoml --------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  [servers]
  alpha = { ip = "127.0.0.1", dc = "eqdc10" }
  beta = { ip = "127.0.0.2", dc = "eqdc20" }
)")

tree

tree |> ts_tree_select("servers", TRUE, "dc")
tree |> ts_tree_select("servers", TRUE, "dc") |> ts_tree_delete()
```

`ts_tree_dom` *Print the document object model (DOM) of a tree-sitter tree*

Description

`tsitter:::doc_insert("ts_tree_dom_description")`

`tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_dom")`

Usage

```
ts_tree_dom(tree)
```

Arguments

`tree`	`tsitter:::doc_insert("tsitter::ts_tree_dom_param_tree")`

Details

`tsitter:::doc_insert("tsitter::ts_tree_dom_details")` `tsitter:::doc_extra()`

Value

`tsitter:::doc_insert("tsitter::ts_tree_dom_return")`

See Also

ts_tree_ast() to show the annotated syntax tree of a ts_tree object.

tsitter:::doc_seealso("ts_tree_dom")

Other ts_tree exploration: ts_tree-brackets, ts_tree_ast(), ts_tree_query(), ts_tree_sexpr()

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"foo": 42, "bar": [1, 2, 3]}')

tree

ts_tree_ast(tree)

ts_tree_dom(tree)



# Create a parse tree with tstoml --------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  title = "TOML Example"
  [owner]
  name = "Tom Preston-Werner"
  dob = 1979-05-27T07:32:00-08:00
)")

tree

ts_tree_ast(tree)

ts_tree_dom(tree)
```

---

ts_tree_format *Format the selected elements of a tree sitter tree for printing*

Description

tsitter:::doc_insert("ts_tree_format_description")

tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_format")

Usage

```
ts_tree_format(tree, options, ...)
```

Arguments

tree	tsitter:::doc_insert("tsitter::ts_tree_format_param_tree")
options	tsitter:::doc_insert("tsitter::ts_tree_format_param_options") See details in the manual of the specific parser.
...	Extra arguments for methods.

Details

tsitter:::doc_insert("tsitter::ts_tree_format_details") tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::ts_tree_format_return")

See Also

tsitter:::doc_seealso("ts_tree_format")

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{ "a":true, "b": [1,2,3] }')
tree

# Format whole document
tree |> ts_tree_format()

# Format each top element under the document node in one line
tree |> ts_tree_format() |>
  ts_tree_select(TRUE) |>
  ts_tree_format(options = list(format = "oneline"))



# Create a parse tree with tstoml --------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  [servers]
  alpha = { ip = "127.0.0.1", dc = "eqdc10" }
  beta = { ip = "127.0.0.2", dc = "eqdc20" }
)")

tree

tree |> ts_tree_format()
```

`ts_tree_insert` *Insert a new element into a tree-sitter tree*

Description

`tsitter:::doc_insert("ts_tree_insert_description")`

`tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_insert")`

Usage

```
ts_tree_insert(tree, new, key, at, options, ...)
```

Arguments

`tree`	`tsitter:::doc_insert("tsitter::ts_tree_insert_param_tree")`
`new`	`tsitter:::doc_insert("tsitter::ts_tree_insert_param_new")` The type of `new` depends on the parser and the method that implements the insertion. See details in the manual of the specific parser.
`key`	`tsitter:::doc_insert("tsitter::ts_tree_insert_param_key")` For example a JSON(C) object or a TOML table are keyed elements.
`at`	`tsitter:::doc_insert("tsitter::ts_tree_insert_param_at")` The interpretation of this argument depends on the method that implements the insertion. Typically the followings are supported: • `0` inserts at the beginning. • `Inf` inserts at the end. • A positive integer `n` inserts *after* the `n`-th element. • A character scalar inserts *after* the element with the given key, in keyed elements. See the details in the manual of the specific parser.
`options`	`tsitter:::doc_insert("tsitter::ts_tree_insert_param_options")` See details in the manual of the specific parser.
`...`	Extra arguments for methods.

Details

`tsitter:::doc_insert("tsitter::ts_tree_insert_details")` `tsitter:::doc_extra()`

Value

`tsitter:::doc_insert("tsitter::ts_tree_insert_return")`

See Also

```
tsitter:::doc_seealso("ts_tree_insert")
```

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{ "a": true, "b": [1, 2, 3] }')

tree |> ts_tree_select("b") |> ts_tree_insert(4, at = Inf)



# Create a parse tree with tstoml --------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  [servers]
  alpha = { ip = "127.0.0.1", dc = "eqdc10" }
  beta = { ip = "127.0.0.2", dc = "eqdc20" }
)")

tree |>
  ts_tree_select("servers", TRUE) |>
  ts_tree_insert(key = "active", TRUE)
```

---

ts_tree_mark_selection1

Helper function to decide which AST nodes to highlight for a selection (internal)

Description

This function are for packages implementing new parsers based on the ts package. It is very unlikely that you will need to call this function directly.

Usage

```
ts_tree_mark_selection1(tree, node)

## S3 method for class 'ts_tree'
ts_tree_mark_selection1(tree, node)
```

Arguments

tree	Tree-sitter tree.
node	Node id, integer scalar.

Details

In parsers where AST nodes do not correspond one-to-one to DOM nodes it is useful to highlight multiple AST nodes for a single selected DOM node. This generic function can be overridden in such parsers to return multiple AST node ids for a single selected (DOM) node id.

The default implementation simply returns the input node id.

Value

Integer vector of node ids to highlight.

Examples

```
# This is an internal generic for parser implementations, see the
# tsjsonc and tstoml packages for examples of methods implementing
# custom behavior.
```

---

ts_tree_new *Create tree-sitter tree from file or string*

Description

This is the main function to create a tree-sitter parse tree, using a ts parser implemented in another package. `tsitter:::doc_insert("ts_tree_new_description")`

`tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers())`

Usage

```
ts_tree_new(
  language,
  file = NULL,
  text = NULL,
  ranges = NULL,
  fail_on_parse_error = TRUE,
  ...
)
```

Arguments

language	`tsitter:::doc_insert("tsitter::ts_tree_new_param_language", "ts")`
file	`tsitter:::doc_insert("tsitter::ts_tree_new_param_file", "ts")`
text	`tsitter:::doc_insert("tsitter::ts_tree_new_param_text", "ts")`
ranges	`tsitter:::doc_insert("tsitter::ts_tree_new_param_ranges", "ts")`
fail_on_parse_error	`tsitter:::doc_insert("tsitter::ts_tree_new_param_fail_on_parse_error", "ts")`
...	Additional arguments for methods.

Details

A package that implements a tree-sitter parser provides a function that creates a `ts_language` object for that parser. E.g. tsjsonc has `tsjsonc::ts_language_jsonc()`. You need to use the returned `ts_language` object as the `language` argument of `ts_tree_new()`.

`tsitter:::doc_insert("ts_tree_new_details") tsitter:::doc_extra()`

Value

`tsitter:::doc_insert("tsitter::ts_tree_new_return", "ts")`

See Also

The tree-sitter parser packages typically include shortcuts to create parse trees from strings and file, e.g. `tsjsonc::ts_parse_jsonc()` and `tsjsonc::ts_read_jsonc()`.

`tsitter:::doc_seealso("ts_tree_new")`

Other ts_tree generics: `[[.ts_tree()`, `[[<-.ts_tree()`, `format.ts_tree()`, `print.ts_tree()`, `select-set`, `ts_tree_ast()`, `ts_tree_delete()`, `ts_tree_dom()`, `ts_tree_format()`, `ts_tree_insert()`, `ts_tree_query()`, `ts_tree_select()`, `ts_tree_sexpr()`, `ts_tree_unserialize()`, `ts_tree_update()`, `ts_tree_write()`

Examples

```
# JSONC example, needs the tsjsonc package -----------------------------
json <- ts_tree_new(
  tsjsonc::ts_language_jsonc(),
  text = '{ "a": 1, "b": 2 }'
)

json

json |> ts_tree_format()



# TOML example, needs the tstoml package -------------------------------
toml <- ts_tree_new(
  tstoml::ts_language_toml(),
  text = '[section]\nkey = "value"\nnumber = 42\n'
)

toml

toml |> ts_tree_format()
```

ts_tree_query *Run tree-sitter queries on tree-sitter trees*

Description

tsitter:::doc_insert("ts_tree_query_description") tsitter:::format_rd_parser_list(tsitter:::ts_list_
"ts_tree_query")

Usage

```
ts_tree_query(tree, query)
```

Arguments

tree	tsitter:::doc_insert("tsitter::ts_tree_query_param_tree")
query	tsitter:::doc_insert("tsitter::ts_tree_query_param_query")

Details

tsitter:::doc_insert("tsitter::ts_tree_query_details") tsitter:::doc_tabs("ts_tree_query_details_exa
tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::ts_tree_query_return")

See Also

ts_tree_select() to select the nodes matching a query.

tsitter:::doc_seealso("ts_tree_query")

Other ts_tree exploration: ts_tree-brackets, ts_tree_ast(), ts_tree_dom(), ts_tree_sexpr()

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_select(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Select all numbers in a JSONC document -----------------------------------
json <- tsjsonc::ts_parse_jsonc(
  '{ "a": 1, "b": [10, 20, 30], "c": { "c1": true, "c2": 100 } }'
)
json |> ts_tree_query("(number) @number")
```

ts_tree_select *Select elements of a tree-sitter tree*

Description

tsitter:::doc_insert("tsitter::ts_tree_select_description")

tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_select")

Usage

```
ts_tree_select(tree, ..., refine = FALSE)
```

Arguments

tree	tsitter:::doc_insert("ts_tree_select_param_tree")
...	tsitter:::doc_insert("ts_tree_select_param_dots")
refine	tsitter:::doc_insert("ts_tree_select_param_refine")

Details

tsitter:::doc_insert("tsitter::ts_tree_select_details") tsitter:::doc_extra()

Value

tsitter:::doc_insert("tsitter::ts_tree_select_return")

See Also

tsitter:::doc_seealso("ts_tree_select")

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_sexpr(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# ------------------------------------------------------------------------
# Create a JSONC tree, needs the tsjsonc package
json <- ts_tree_new(
  tsjsonc::ts_language_jsonc(),
  text = '{ "a": 1, "b": 2, "c": { "d": 3, "e": 4 } }'
)

json |> ts_tree_select("c", "d")
```

```
# ---------------------------------------------------------------------
# Create a TOML tree, needs the tstoml package
toml <- ts_tree_new(
  tstoml::ts_language_toml(),
  text = tstoml::toml_example_text()
)

toml |> ts_tree_select("servers", TRUE, "ip")
```

`ts_tree_select1` *Select nodes from a tree-sitter tree (internal)*

Description

This function is for packages implementing new parsers based on the ts package. It is very unlikely that you will need to call this function directly.

Usage

```
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.ts_tree_selector_default'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.NULL'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.ts_tree_selector_ids'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.ts_tree_selector_tsquery'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.character'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.integer'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.numeric'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.ts_tree_selector_regex'
ts_tree_select1(tree, node, slt)

## S3 method for class 'ts_tree.logical'
ts_tree_select1(tree, node, slt)
```

Arguments

tree	A ts_tree object as returned by ts_tree_new().
node	Integer scalar, the node id to select from.
slt	A selector object, see details in ts_tree_select().

Details

A parser package may implement methods for this generic to change the behavior of ts_tree_select() for a certain selector type, or even add new selector types.

Each new method should be named as

```
ts_tree_select.<ts_tree_class>.<selector_class>
```

The ts package implement deault methods for the selector types described in the ts_tree_select() manual page.

ts_tree_selector_default **selector:**

Method: ts_tree_select1.ts_tree.ts_tree_selector_default

This method is used to select the default element(s), when there is no selected element. E.g. when starting a new selection from the root of the DOM tree.

The default implementation returns the ids of all children of the document root in the AST, except comments. If there are no such children, it returns the id of the document root of the AST itself (always id 1).

NULL **selector:**

Method: ts_tree_select1.ts_tree.NULL

This method is used for the NULL selector, that is supposed to clear the selection. You probably do not need to override this method. The default implementation returns an empty integer vector.

ts_tree_selector_ids **selector:**

Method: ts_tree_select1.ts_tree.ts_tree_selector_ids

This method is used to select nodes by their ids directly. You probably do not need to override this method. The default implementation returns the ids stored in the selector.

Note:
This behaviour may change in the future to select only nodes in the subtree of the current node.

ts_tree_selector_tsquery **selector:**

Method: ts_tree_select1.ts_tree.ts_tree_selector_tsquery

This method is used to select nodes matching a tree-sitter query. You probably do not need to override this method. The default implementation returns the ids stored in the selector.

Note:
This behaviour may change in the future to select only nodes in the subtree of the current node.

`character` **(character vector) selector:**

Method: `ts_tree_select1.ts_tree.character`

This method is used when the selector is a character vector. The default implementation selects DOM children of `node` whose names are in the character vector. If not all children o `node` are named, it returns an empty integer vector. (E.g. in a JSONC document it returns an empty integer vector when nodes is an array.)

`integer` **(integer vector) selector:**

Method: `ts_tree_select1.ts_tree.integer`

This method is used when the selector is an integer vector. The default implementation selects DOM children of `node` by position. Positive indices count from the start, negative indices count from the end. Zero indices are not allowed and an error is raised if any are used.

`numeric` **(numeric, double vector) selector:**

Method: `ts_tree_select1.ts_tree.numeric`

This method is used when the selector is a numeric (double) vector. It currrently coerces the numeric vector to integer and calls the integer method.

`ts_tree_selector_regex` **(regular expression) selector:**

Method: `ts_tree_select1.ts_tree.ts_tree_selector_regex`

This method is used when the selector is a regular expression. The default implementation selects DOM children of `node` whose names match the regular expression. If not all children o `node` are named, it returns an empty integer vector. (E.g. in a JSONC document it returns an empty integer vector when nodes is an array.)

`logical` **(logical vector) selector:**

Method: `ts_tree_select1.ts_tree.logical`

This method is used when the selector is a logical vector. The default implementation only supports scalar `TRUE`, which selects all DOM children of `node`. Other values raise an error.

Value

Must return an integer vector of selected node ids.

Examples

```
# This is an internal generic for parser implementations, see the
# tsjsonc and tstoml packages for examples of methods implementing
# selector types.
```

`ts_tree_selection` *Helper functions for tree-sitter tree selections (internal)*

Description

These functions are for packages implementing new parsers based on the ts package. It is very unlikely that you will need to call these functions directly.

Usage

```
ts_tree_selection(tree, default = TRUE)

ts_tree_selected_nodes(tree, default = TRUE)
```

Arguments

`tree`	A `ts_tree` object as returned by `ts_tree_new()`.
`default`	Logical, whether to return the default selection if there is no explicit selection, or `NULL`.

Details

`ts_tree_selection()` returns the current selection, as a list of selectors.

`ts_tree_selected_nodes()` returns the ids of the currently selected nodes.

Value

`ts_tree_selection()` returns a list of selection records.

`ts_tree_selected_nodes()` returns the ids of the currently selected nodes.

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"a": 13, "b": [1, 2, 3], "c": "x"}')
tree <- ts_tree_select(tree, "b", -1)
ts_tree_selection(tree)
```

ts_tree_sexpr *Show the syntax tree of a tree-sitter tree*

Description

tsitter:::doc_insert("ts_tree_sexpr_description") tsitter:::format_rd_parser_list(tsitter:::ts_list_
"ts_tree_sexpr")

Usage

```
ts_tree_sexpr(tree)
```

Arguments

tree	tsitter:::doc_insert("tsitter::ts_tree_sexpr_param_tree")

Details

tsitter:::doc_insert("ts_tree_sexpr_details")

Value

tsitter:::doc_insert("tsitter::ts_tree_sexpr_return")

See Also

Other ts_tree exploration: ts_tree-brackets, ts_tree_ast(), ts_tree_dom(), ts_tree_query()

Other ts_tree generics: [[.ts_tree(), [[<-.ts_tree(), format.ts_tree(), print.ts_tree(), select-set, ts_tree_ast(), ts_tree_delete(), ts_tree_dom(), ts_tree_format(), ts_tree_insert(), ts_tree_new(), ts_tree_query(), ts_tree_select(), ts_tree_unserialize(), ts_tree_update(), ts_tree_write()

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc(
  "{ \"a\": //comment\ntrue, \"b\": [1, 2, 3] }"
)

ts_tree_sexpr(tree)



# Create a parse tree with tstoml --------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  [servers]
  alpha = { ip = "127.0.0.1", dc = "eqdc10" }
  beta = { ip = "127.0.0.2", dc = "eqdc20" }
)")
```

```
ts_tree_sexpr(tree)
```

`ts_tree_unserialize` *Unserialize selected elements of a tree-sitter tree*

Description

```
tsitter:::doc_insert("ts_tree_unserialize_description")
tsitter:::format_rd_parser_list(tsitter:::ts_list_parsers(), "ts_tree_unserialize")
```

Usage

```
ts_tree_unserialize(tree)
```

Arguments

`tree`	`tsitter:::doc_insert("tsitter::ts_tree_unserialize_param_tree")`

Details

`tsitter:::doc_insert("tsitter::ts_tree_unserialize_details")` `tsitter:::doc_extra()`
For the details on how the selected elements are mapped to R objects, see the documentation of the methods in the parser packages. The methods in the installed parser packages are linked below.

Value

```
tsitter:::doc_insert("tsitter::ts_tree_unserialize_return")
```

See Also

```
tsitter:::doc_seealso("ts_tree_unserialize")
```

Other ts_tree generics: `[[.ts_tree()`, `[[<-.ts_tree()`, `format.ts_tree()`, `print.ts_tree()`, `select-set`, `ts_tree_ast()`, `ts_tree_delete()`, `ts_tree_dom()`, `ts_tree_format()`, `ts_tree_insert()`, `ts_tree_new()`, `ts_tree_query()`, `ts_tree_select()`, `ts_tree_sexpr()`, `ts_tree_update()`, `ts_tree_write()`

Other serialization functions: `[[.ts_tree()`

Examples

```
# Create a parse tree with tsjsonc ------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"a": 13, "b": [1, 2, 3], "c": "x"}')

tree

tree |> ts_tree_select(c("b", "c")) |> ts_tree_unserialize()
```

```
tree |> ts_tree_select("b") |> ts_tree_unserialize()
```

`ts_tree_update` *Replace selected elements with a new element in a tree-sitter tree*

Description

```
tsitter:::doc_insert("ts_tree_update_description")
```

Usage

```
ts_tree_update(tree, new, options, ...)
```

Arguments

`tree`	`tsitter:::doc_insert("tsitter::ts_tree_update_param_tree")`
`new`	`tsitter:::doc_insert("tsitter::ts_tree_update_param_new")` The type of `new` depends on the parser and the method that implements the insertion. See details in the manual of the specific parser.
`options`	`tsitter:::doc_insert("tsitter::ts_tree_update_param_options")` See details in the manual of the specific parser.
`...`	Extra arguments for methods.

Details

```
tsitter:::doc_insert("tsitter::ts_tree_update_details") tsitter:::doc_extra()
```

Value

```
tsitter:::doc_insert("tsitter::ts_tree_update_return")
```

See Also

```
tsitter:::doc_seealso("ts_tree_update")
```

Other ts_tree generics: `[[.ts_tree()`, `[[<-.ts_tree()`, `format.ts_tree()`, `print.ts_tree()`, `select-set`, `ts_tree_ast()`, `ts_tree_delete()`, `ts_tree_dom()`, `ts_tree_format()`, `ts_tree_insert()`, `ts_tree_new()`, `ts_tree_query()`, `ts_tree_select()`, `ts_tree_sexpr()`, `ts_tree_unserialize()`, `ts_tree_write()`

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc(r"(
  {
    "name": "example",
    "version": "1.0.0",
    "dependencies": {
      "tsjsonc": "^0.1.0"
    }
  }
)")

tree |> ts_tree_select("version") |> ts_tree_update("2.0.0")


# Create a parse tree with tstoml --------------------------------------
tree <- tstoml::ts_parse_toml(r"(
  [package]
  name = "example"
  version = "1.0.0"
  depdendencies = { tstoml = "0.1.0" }
)")

tree |> ts_tree_select("package", "version") |> ts_tree_update("2.0.0")
```

---

ts_tree_write *Write a tree-sitter tree to a file*

Description

```
tsitter:::doc_insert("ts_tree_write_description")
```

Usage

```
ts_tree_write(tree, file = NULL)
```

Arguments

tree	tsitter:::doc_insert("tsitter::ts_tree_write_param_tree")
file	tsitter:::doc_insert("tsitter::ts_tree_write_param_file")

Details

```
tsitter:::doc_insert("tsitter::ts_tree_write_details") tsitter:::doc_extra()
```

Value

```
tsitter:::doc_insert("tsitter::ts_tree_write_return")
```

See Also

```
tsitter:::doc_seealso("ts_tree_write")
```

Other ts_tree generics: `[[.ts_tree()`, `[[<-.ts_tree()`, `format.ts_tree()`, `print.ts_tree()`, `select-set`, `ts_tree_ast()`, `ts_tree_delete()`, `ts_tree_dom()`, `ts_tree_format()`, `ts_tree_insert()`, `ts_tree_new()`, `ts_tree_query()`, `ts_tree_select()`, `ts_tree_sexpr()`, `ts_tree_unserialize()`, `ts_tree_update()`

Examples

```
# Create a parse tree with tsjsonc -------------------------------------
tree <- tsjsonc::ts_parse_jsonc('{"foo": 42, "bar": [1, 2, 3]}')

# Format and write to file
tree |> ts_tree_format() |> ts_tree_write("example.json")
```

Index

∗ **serialization functions**
  [[.ts_tree, 2
  ts_tree_unserialize, 30
∗ **ts_tree exploration**
  ts_tree-brackets, 13
  ts_tree_ast, 14
  ts_tree_dom, 16
  ts_tree_query, 23
  ts_tree_sexpr, 29
∗ **ts_tree generics**
  [[.ts_tree, 2
  [[<-.ts_tree, 3
  format.ts_tree, 9
  print.ts_tree, 10
  select-set, 11
  ts_tree_ast, 14
  ts_tree_delete, 15
  ts_tree_dom, 16
  ts_tree_format, 17
  ts_tree_insert, 19
  ts_tree_new, 21
  ts_tree_query, 23
  ts_tree_select, 24
  ts_tree_sexpr, 29
  ts_tree_unserialize, 30
  ts_tree_update, 31
  ts_tree_write, 32
[.ts_tree *(*ts_tree-brackets*)*, 13
[[.ts_tree, 2
[[.ts_tree(), *4*, *9–11*, *14, 15*, *17, 18*, *20*, *22–24*, *29–31*, *33*
[[<-.ts_tree, 3

about, 4
as.character.ts_tree, 7
as.character.ts_tree(), *8*
as.raw.ts_tree, 8
as.raw.ts_tree(), *8*

base::.libPaths(), *12*

format.ts_tree, 9
format.ts_tree(), *3, 4*, *10, 11*, *14, 15*, *17, 18*, *20*, *22–24*, *29–31*, *33*

print.ts_tree, 10
print.ts_tree(), *3, 4*, *9*, *11*, *14, 15*, *17, 18*, *20*, *22–24*, *29–31*, *33*

select-set, 11

ts_list_parsers, 12
ts_tree-brackets, 13
ts_tree_ast, 14
ts_tree_ast(), *3, 4*, *9–11*, *13*, *15*, *17, 18*, *20*, *22–24*, *29–31*, *33*
ts_tree_delete, 15
ts_tree_delete(), *3, 4*, *9–11*, *14*, *17, 18*, *20*, *22–24*, *29–31*, *33*
ts_tree_deleted *(*select-set*)*, 11
ts_tree_dom, 16
ts_tree_dom(), *3, 4*, *9–11*, *13–15*, *18*, *20*, *22–24*, *29–31*, *33*
ts_tree_format, 17
ts_tree_format(), *3, 4*, *9–11*, *14, 15*, *17*, *20*, *22–24*, *29–31*, *33*
ts_tree_insert, 19
ts_tree_insert(), *3, 4*, *9–11*, *14, 15*, *17, 18*, *22–24*, *29–31*, *33*
ts_tree_mark_selection1, 20
ts_tree_new, 21
ts_tree_new(), *3, 4*, *9–11*, *14, 15*, *17, 18*, *20*, *22–24*, *26*, *28–31*, *33*
ts_tree_query, 23
ts_tree_query(), *3, 4*, *9–11*, *13–15*, *17, 18*, *20*, *22*, *24*, *29–31*, *33*
ts_tree_select, 24
ts_tree_select(), *3, 4*, *9–11*, *14, 15*, *17, 18*, *20*, *22, 23*, *26*, *29–31*, *33*
ts_tree_select1, 25
ts_tree_select<- *(*select-set*)*, 11

ts_tree_selected_nodes
(ts_tree_selection), 28
ts_tree_selection, 28
ts_tree_sexpr, 29
ts_tree_sexpr(), *3, 4*, *9–11*, *13–15*, *17, 18*, *20*, *22–24*, *30, 31*, *33*
ts_tree_unserialize, 30
ts_tree_unserialize(), *3, 4*, *9–11*, *14, 15*, *17, 18*, *20*, *22–24*, *29*, *31*, *33*
ts_tree_update, 31
ts_tree_update(), *3, 4*, *9–11*, *14, 15*, *17, 18*, *20*, *22–24*, *29, 30*, *33*
ts_tree_write, 32
ts_tree_write(), *3, 4*, *9–11*, *14, 15*, *17, 18*, *20*, *22–24*, *29–31*
tsjsonc, *22*
tsjsonc::ts_language_jsonc(), *22*
tsjsonc::ts_parse_jsonc(), *22*
tsjsonc::ts_read_jsonc(), *22*