Package 'tsitter'

May 22, 2026

Title Tree-Sitter Parsing Tools

Version 0.1.0

Description Common tree-sitter (<https://tree-sitter.github.io/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

Config/Needs/check gaborcsardi/tsjsonc, gaborcsardi/tstoml

Config/Needs/coverage 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, gaborcsardi/tsjsonc, gaborcsardi/tstoml

Biarch true

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Gábor Csárdi [aut, cre], Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>), Tree-sitter authors [cph] (Tree-sitter C library, see AUTHORS file), ICU authors [cph] (ICU library, see AUTHORS file)

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Repository CRAN

Date/Publication 2026-05-22 09:20:02 UTC

Contents

`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:

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

Delete elements:
Delete selected elements:

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

Insert elements:
Insert element into an array:

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

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

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

Update elements:
Update existing element:

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

Inserts the element if some parents are missing:

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

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

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

```
ts_tree_write(json, stdout())
```

Formatting:
Format the whole document:

```
ts_tree_format(json)
```

Format part of the document:

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

Unserializing:
Unserialize a whole document:

```
ts_tree_unserialize(json)
```

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:

```
ts_tree_unserialize(ts_tree_select(json, "b"))
```

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

Format a `ts_tree` object for printing.

Usage

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

Arguments

`x`	`ts_tree` object.
`n`	Number of lines, or number of selections to print.
`...`	Currently ignored.

Details

This is the engine of `print.ts_tree()`, possibly useful to obtain a printed representation without doing the actual printing.

If there are selected nodes in the tree, those will be highlighted in the output. See `ts_tree_select()` to select nodes in a tree.

Value

Character vector of lines to print.

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

Print a ts_tree object to the screen.

Usage

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

Arguments

x	ts_tree object to print.
n	Number of lines, or number of selections to print.
...	Not used currently.

Details

Calls format.ts_tree() to format the ts_tree object, writes the formatted object to the standard output, and returns the original object invisibly.

Value

Invisibly returns the original ts_tree object.

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

The `ts_tree_select<-()` replacement function works similarly to the combination of `ts_tree_select()` and `ts_tree_update()`, but it might be more readable.

Usage

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

Arguments

`tree`	A `ts_tree` object as returned by `ts_tree_new()`.
`...`	Selection expressions, see `ts_tree_select()`.
`value`	An R expression to serialize or `ts_tree_deleted()`.

Details

The following two expressions are equivalent:

```
tree <- ts_tree_update(ts_tree_select(tree, <selectors>), value)
```

and

```
ts_tree_select(tree, <selectors>) <- value
```

`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

A `ts_tree` object with the selected parts updated.

`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_ast *Show the annotated syntax tree of a tree-sitter tree*

Description

`ts_tree_ast()` prints the annotated syntax tree of a ts_tree object. This syntax tree contains all tree-sitter nodes, and it shows the source code associated with each node, along with line numbers.

Usage

```
ts_tree_ast(tree)
```

Arguments

`tree`	A `ts_tree` object.

Details

The syntax tree and the DOM tree:

This syntax tree contains all nodes of the tree-sitter parse tree, including both named and unnamed nodes and comments. E.g. for a JSON(C) document it includes the pairs, brackets, braces, commas, colons, double quotes and string escape sequences as separate nodes.

See `tsitter::ts_tree_dom()` for a tree that shows the semantic structure of the parsed document, which may be different from the syntax tree.

Value

Character vector, the formatted annotated syntax tree, line by line. It has class cli_tree, from the cli package. It may contain ANSI escape sequences for coloring and hyperlinks.

See Also

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

Other ts_tree exploration: `[.ts_tree()`, `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

Use `ts_tree_select()` to select the elements to be deleted, and then call `ts_tree_delete()` to remove them from the tree.

Usage

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

Arguments

`tree`	A `ts_tree` object.
`...`	Extra arguments for methods.

Details

The formatting of the rest of the document is left as is.

If the tree does not have a selection, the tree corresponding to the empty document is returned, i.e. the whole content is deleted.

If the tree has a selection, but it is the empty selection, then the tree is returned unchanged.

For parsers that support comments, deleting elements that include comments typically delete the comments as well. Other comments are kept as is. See details in the manual of the specific parser.

Value

The modified `ts_tree` object with the selected elements removed.

See Also

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

ts_tree_select(tree, "a")

ts_tree_delete(ts_tree_select(tree, "a"))


# 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(tree, "servers", TRUE, "dc")
ts_tree_delete(ts_tree_select(tree, "servers", TRUE, "dc"))
```

---

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

Description

ts_tree_dom() prints the document object model (DOM) tree of a ts_tree object. This tree only includes semantic elements. E.g. for a JSON(C) document it includes objects, arrays and various value types, but not the syntax elements like brackets, commas or colons.

Usage

```
ts_tree_dom(tree)
```

Arguments

tree	A ts_tree object.

Details

The syntax tree and the DOM tree:

See ts_tree_ast() for the complete tree-sitter syntax tree that includes all nodes, including syntax elements like brackets and commas.

Value

Character vector, the formatted annotated syntax tree, line by line. It has class cli_tree, from the cli package. It may contain ANSI escape sequences for coloring and hyperlinks.

See Also

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

Other ts_tree exploration: [.ts_tree(), 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

(Re)format the selected elements of the document represented by a tree-sitter tree, if the tree-sitter parser supports formatting.

Usage

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

Arguments

`tree`	A `ts_tree` object.
`options`	A list of options for the formatting. See details in the manual of the specific parser.
`...`	Extra arguments for methods.

Details

If `tree` does not have a selection, then the whole document is formatted.

If `tree` has an empty selection, then it is returned unchanged.

Some parsers support options to customize the formatting. See details in the manual of the specific parser.

Value

A `ts_tree` object representing the reformatted document.

See Also

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
ts_tree_format(tree)

# Format each top element under the document node in one line
```

```
ts_tree_format(
  ts_tree_select(ts_tree_format(tree), TRUE),
  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

ts_tree_format(tree)
```

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

Description

Insert a new element into each selected element.

Usage

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

Arguments

`tree`	A `ts_tree` object.
`new`	The new element to insert. 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`	The key of the new element, if inserting into a keyed element. For example a JSON(C) object or a TOML table are keyed elements.
`at`	The position to insert the new element 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`	A list of options for the insertion. See details in the manual of the specific parser.
`...`	Extra arguments for methods.

Details

It is not always possible to insert a new element into a selected element. For example in a JSONC document you can only insert a new element into an array or an object, but not into scalar elements. If the insertion is not possible, an error is raised.

If `tree` does not have a selection, the new element is inserted into at the top level.

If `tree` has an empty selection, then it is returned unchanged, i.e. no new element is inserted.

Value

A `ts_tree` object representing the modified parse tree.

See Also

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] }')

ts_tree_insert(ts_tree_select(tree, "b"), 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" }
)")

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

`ts_tree_mark_selection1` *Helper function to decide which AST nodes to highlight for a selection (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_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.

The result is a `ts_tree` object. A `ts_tree` object may be queried, edited, formatted, written to file, etc. using `ts_tree` methods.

Usage

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

Arguments

`language`	Language of the file or string, a `ts_language` object, e.g. the return value of `tsjsonc::ts_language_jsonc()`.
`file`	Path of a file to parse. Use either `file` or `text`, but not both.
`text`	String to parse. Use either `file` or `text`, but not both.
`ranges`	Can be used to parse part(s) of the input. It must be a data frame with integer columns `start_row`, `start_col`, `end_row`, `end_col`, `start_byte`, `end_byte`, in this order.
`fail_on_parse_error`	Logical, whether to error if there are parse errors in the document. Default is `TRUE`.
`...`	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()`.

Value

A `ts_tree` object representing the parse tree of the input. You can use the single bracket `[` operator to convert it to a data frame.

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()`.

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

ts_tree_format(json)



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

toml

ts_tree_format(toml)
```

---

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

Description

Use tree-sitter's query language to find nodes in a tree-sitter tree.

Usage

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

Arguments

`tree`	A `ts_tree` object.
`query`	Character string, the tree-sitter query to run.

Details

You probably need to know some details about the specific tree-sitter parser you are using, to write effective queries. See the documentation of the parser package you are using for details about the node types and the query language support. See links below.

Value

A list with entries `patterns` and `matched_captures`.

`patterns` contains information about all patterns in the queries and it is a data frame with columns: `id`, `name`, `pattern`, `match_count`.

`matched_captures` contains information about all matches, and it has columns `id`, `pattern`, `match`, `start_byte`, `end_byte`, `start_row`, `start_column`, `end_row`, `end_column`, `name`, `code`.

The `pattern` column of `matched_captured` refers to the `id` column of `patterns`.

See Also

`ts_tree_select()` to select the nodes matching a query.

Other ts_tree exploration: `[.ts_tree()`, `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 } }'
)
ts_tree_query(json, "(number) @number")
```

---

ts_tree_select *Select elements of a tree-sitter tree*

Description

This function is the heart of ts. To edit a tree-sitter tree, you first need to select the parts you want to delete or update.

Usage

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

Arguments

`tree`	A `ts_tree` object as returned by `ts_tree_new()`.
`...`	Selection expressions, see details.
`refine`	Logical, whether to refine the current selection or start a new selection.

Details

The selection process is iterative. Selection expressions (selectors) are applied one by one, and each selector selects nodes from the currently selected nodes. For each selector, it is applied individually to each currently selected node, and the results are concatenated.

The selection process starts from the root of the DOM tree, the document node (see `ts_tree_dom()`), unless `refine = TRUE` is set, in which case it starts from the current selection.

See the various types of selection expressions below.

Selectors:

All elements: `TRUE`*:*
Selects all child nodes of the current nodes.

Specific keys: character vector:
Selects child nodes with the given names from nodes with named children. If a node has no named children, it selects nothing from that node.

By position: integer vector:
Selects child nodes by position. Positive indices count from the start, negative indices count from the end. Zero indices are not allowed.

Matching keys: regular expression:
A character scalar named `regex` can be used to select child nodes whose names match the given regular expression, from nodes with named children. If a node has no named children, it selects nothing from that node.

Tree sitter query matches:
A character scalar named `query` can be used to select nodes matching a tree-sitter query. See `ts_tree_query()` for details on tree-sitter queries.
Instead of a character scalar this can also be a two-element list, where the first element is the query string and the second element is a character vector of capture names to select. In this case only nodes matching the given capture names will be selected.

Explicit node ids:
You can use `I(c(...))` to select nodes by their ids directly. This is for advanced use cases only.

Refining selections:

If the `refine` argument of `ts_tree_select()` is `TRUE`, then the selection starts from the already selected elements (all of them simultanously), instead of starting from the document element.

The `ts_tree_select<-()` replacement function:

The `ts_tree_select<-()` replacement function works similarly to the combination of `ts_tree_select()` and `ts_tree_update()`, but it might be more readable.

The `[[` and `[[<-` operators:

The `[[` operator works similarly to the combination of `ts_tree_select()` and `ts_tree_unserialize()`, but it might be more readable.

The `[[<-` operator works similarly to the combination of `ts_tree_select()` and `ts_tree_update()`, (and also to the replacement function `ts_tree_select<-()`), but it might be more readable.

Value

A `ts_tree` object with the selected parts.

See Also

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 } }'
)

ts_tree_select(json, "c", "d")



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

ts_tree_select(toml, "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

Show the structure of a tree-sitter tree as an S-expression.

Usage

```
ts_tree_sexpr(tree)
```

Arguments

`tree`	A `ts_tree` object.

Details

This function returns a nested list representation of the syntax tree, where each node is represented as a list with its type and children.

Value

A string representing the S-expression of the syntax tree.

See Also

Other ts_tree exploration: `[.ts_tree()`, `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

Unserialize the selected elements of a `ts_tree` object, i.e. convert them to R objects.

Usage

```
ts_tree_unserialize(tree)
```

Arguments

`tree`	A `ts_tree` object.

Details

If no elements are selected in the tree, then the whole document is unserialized.

If the tree has an empty selection, then an empty list is returned.

The `[[` operator:

The `[[` operator works similarly to the combination of `ts_tree_select()` and `ts_tree_unserialize()`, but it might be more readable.

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

List of R objects, with one entry for each selected element.

See Also

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

ts_tree_unserialize(ts_tree_select(tree, c("b", "c")))

ts_tree_unserialize(ts_tree_select(tree, "b"))
```

---

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

Description

Replace all selected elements with a new element.

Usage

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

Arguments

tree	A ts_tree object.
new	The new element to replace the selected elements with. The type of new depends on the parser and the method that implements the insertion. See details in the manual of the specific parser. See details in the manual of the specific parser.
options	A list of options for the update.
...	Extra arguments for methods.

Details

If the tree does not have a selection, the new element replaces the whole document.

If the tree has an empty selection, the new element is inserted at the position of where the selected elements would be.

Value

The modified `ts_tree` object with the selected elements replaced by the new element.

See Also

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"
    }
  }
)")

ts_tree_update(ts_tree_select(tree, "version"), "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" }
)")

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

---

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

Description

Writes the document of a ts `ts_tree` object to a file or connection.

Usage

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

Arguments

tree	A ts_tree object as returned by ts_tree_new().
file	Character string, connection, or NULL. The file or connection to write to. By default it writes to the same file that was used in ts_tree_new(), if tree was read from a file.

Details

If tree was created from a file, then ts_tree_write() by default writes it back to the same file. Otherwise, the file argument must be specified.

To write to a connection, pass a connection object to the file argument. If the connection is opened in binary mode, the raw bytes are written using base::writeBin(). Otherwise, the raw bytes are converted to characters using the system encoding before writing using base::rawToChar().

Use file = stdout() to write to the standard output, i.e. to the console in an interactive R session.

Value

Invisibly returns NULL.

See Also

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
ts_tree_write(ts_tree_format(tree), "example.json")
```

[.ts_tree *Convert ts_tree object to a data frame*

Description

Create a data frame for the syntax tree of a JSON document, by indexing a ts_tree object with single brackets. This is occasionally useful for exploration and debugging.

Usage

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

Arguments

x	A ts_tree object.
i, j	Incides, passed to the regular data.frame indexing method, see Extract.
drop	Passed to the regular data.frame indexing method, see Extract.

Details

A tree-sitter tree object has at least four classes:

- ts_tree_<parser_name>, e.g. ts_tree_tsjsonc,
- ts_tree,
- tbl, from the pillar package, for better printing when converted to a data frame, and
- data.frame, since it is a data frame internally.

The ts_tree class has custom format() and print() methods, that show (part of) the underlying document, and also the selected elements, if any.

It is sometimes useful to treat a tree ts_tree object as a data frame, and drop the ts_tree classes. This can be done by indexing with single brackets, e.g. tree[]. This returns a data frame with one row per token, and various columns with information about the tokens. See details in the 'Value' section or this page.

Value

A data frame with one row per token, and columns:

- id: integer, the id of the token. The (root) document node has id 1.
- parent: integer, the id of the parent token. The root token has parent NA
- field_name: character, the field name of the token in its parent.
- type: character, the type of the token.
- code: character, the actual code of the token.
- start_byte, end_byte: integer, the byte positions of the token in the input.
- start_row, start_column, end_row, end_column: integer, the position of the token in the input.
- is_missing: logical, whether the token is a missing token added by the parser to recover from errors.
- has_error: logical, whether the token has a parse error.
- children: list of integer vectors, the ids of the children tokens.
- dom_type: character, the type of the node in the DOM tree. See ts_tree_dom(). Nodes that are not part of the DOM tree have NA_character_ here.
- dom_children: list of integer vectors, the ids of the children in the DOM tree. See ts_tree_dom().
- dom_parent: integer, the parent of the node in the DOM tree. See ts_tree_dom(). Nodes that are not part of the DOM tree and the document node have have NA_integer_ here.

Other, undocumented columns may also be present, these are considered internal and may change without notice.

See Also

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` *Unserialize parts of a tree-sitter tree*

Description

The `[[` operator works similarly to the combination of `ts_tree_select()` and `ts_tree_unserialize()`, but it might be more readable.

Usage

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

Arguments

`x`	A `ts_tree` object.
`i`	Selection expressions in a list, see details in `ts_tree_select()`.
`...`	Additional arguments, passed to `ts_tree_select()`.

Details

The following two expressions are equivalent:

```
ts_tree_unserialize(ts_tree_select(tree, <selectors>))
```

and

```
tree[[list(<selectors>)]]
```

The `[[<-` replacement operator:

The `[[<-` operator works similarly to the combination of `ts_tree_select()` and `ts_tree_update()`, (and also to the replacement function `ts_tree_select<-()`), but it might be more readable.

Value

List of R objects, with one entry for each selected element.

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

The `[[<-` operator works similarly to the combination of `ts_tree_select()` and `ts_tree_update()`, (and also to the replacement function `ts_tree_select<-()`), but it might be more readable.

Usage

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

Arguments

`x`	A `ts_tree` object.
`i`	A list with selection expressions, see `ts_tree_select()` for details.
`value`	An R expression to serialize or `ts_tree_deleted()`.

Details

The following two expressions are equivalent:

```
tree <- ts_tree_update(ts_tree_select(tree, <selectors>), value)
```

and

```
tree[[list(<selectors>)]] <- value
```

Value

The modified `ts_tree` object.

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
```

Index