

# Package ‘sketch’

October 6, 2021

**Type** Package

**Title** Interactive Sketches

**Version** 1.1.17

**Description** Creates static / animated / interactive visualisations embeddable in R Markdown documents. It implements an R-to-JavaScript transpiler and enables users to write JavaScript applications using the syntax of R.

**License** Apache License (>= 2.0)

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Imports** magrittr, rlang, purrr, rstudioapi, glue, htmltools,  
base64enc, jsonlite, shiny, methods

**Suggests** testthat, covr, knitr, rmarkdown,

**BugReports** <https://github.com/kcf-jackson/sketch>

**NeedsCompilation** no

**Author** Chun Fung Kwok [aut, cre] (<<https://orcid.org/0000-0002-0716-3879>>),  
Kate Saunders [ctb]

**Maintainer** Chun Fung Kwok <jkwok@svi.edu.au>

**Repository** CRAN

**Date/Publication** 2021-10-06 15:10:02 UTC

## R topics documented:

sketch-package	2
assets	3
basic_deparsers	3
bundle	4
combine_rules	4
compile_active	5
compile_data	5
compile_exprs	6
compile_r	6

convert_src . . . . .	7
default_2_deparers . . . . .	8
default_deparers . . . . .	8
default_processors . . . . .	9
deparse_Family . . . . .	9
deparse_js . . . . .	13
dp . . . . .	14
eng_sketch . . . . .	14
flatten_filelist . . . . .	15
get_dependencies . . . . .	15
html_tags . . . . .	16
insert_sketch . . . . .	17
is_call . . . . .	17
is_Family . . . . .	18
is_syntactic_literal . . . . .	21
let-declare-const . . . . .	22
license_info . . . . .	22
list-of-deparers . . . . .	23
load_Family . . . . .	23
local . . . . .	24
make_deparser . . . . .	25
make_processor . . . . .	25
make_rule . . . . .	26
parse_expr . . . . .	26
print.sketch_rule . . . . .	27
r-to-js-rules . . . . .	28
read_multilines . . . . .	28
rewrite . . . . .	29
runShinyApp . . . . .	30
safeguard . . . . .	30
source_active . . . . .	31
source_js . . . . .	32
source_r . . . . .	32
split_rules . . . . .	33
src . . . . .	34
test_sketch . . . . .	34
to_json . . . . .	35
websocket . . . . .	35

<b>Index</b>	<b>40</b>
--------------	-----------

---

sketch-package	<i>Interactive visualisation</i>
----------------	----------------------------------

---

## Description

Creates interactive visualisation embeddable in R Markdown documents. It implements an R-to-JavaScript transpiler and enables users to write JavaScript applications using the syntax of R.

---

assets	<i>Process assets in headers</i>
--------	----------------------------------

---

**Description**

Take a 'sketch' R file as input, extract and process the resources links as provided by the user with the '#!' header.

**Usage**

```
assets(file, ..., trace = FALSE)
```

**Arguments**

file	Character string; the file path.
...	(Optional) List of processors to pass to <a href="#">convert_src</a> .
trace	TRUE or FALSE; if TRUE, assets are extracted, but not processed.

**Examples**

```
file <- system.file("test_files/test_RMD.R", package = "sketch")
assets(file, trace = TRUE)
assets(file, trace = FALSE)
```

---

basic_deparsers	<i>A minimal list of deparsers for deparsing JavaScript</i>
-----------------	---

---

**Description**

A minimal list of deparsers for deparsing JavaScript

**Usage**

```
basic_deparsers()
```

**Note**

This is used as input to [deparsed\\_js](#), [compile\\_r](#) and [compile\\_exprs](#).

**Examples**

```
basic_deparsers()
```

---

bundle	<i>Bundle a list of files into a single JavaScript file</i>
--------	---

---

**Description**

Bundle a list of files into a single JavaScript file

**Usage**

```
bundle(fs)
```

**Arguments**

fs	A character vector; a list of R or JavaScript files. The R files will be transpiled to JavaScript before bundling.
----	--

**Examples**

```
library(sketch)
js <- bundle(c(src("dom"), src("io")))
cat(paste(readLines(js), collapse = "\n"))
```

---

combine_rules	<i>Combine rules for fast transpilation</i>
---------------	---

---

**Description**

This function turns an n-pass transpilation into k-pass, where n is the number of rules and k is the number of precedence groups.

**Usage**

```
combine_rules(rs, group = rep(1, length(rs)))
```

**Arguments**

rs	A list of rewriting rules (each of which is an output from <a href="#">make_rule</a> ).
group	A numeric vector; the precedence group. Rules with a higher precedence come before the the ones with lower precedence, and they are processed by the transpiler first. For rules with the same precedence, the natural order (in which they show up) determines which rules get processed first.

**Note**

The key insight about optimising the transpilation is that rewriting passes that do not interfere with each other can be combined, and it saves a full traversal of the parse tree.

---

compile_active	<i>Compile active file in 'RStudio'</i>
----------------	---

---

**Description**

Compile active file in 'RStudio'

**Usage**

```
compile_active(...)
```

**Arguments**

... Optional arguments to pass to compile\_r.

**Examples**

```
## Not run:  
# At 'RStudio', opens a 'sketch' R file in the editor, then  
# run the following:  
compile_active()  
  
## End(Not run)
```

---

compile_data	<i>Compile a data file into a JavaScript file</i>
--------------	---

---

**Description**

Compile a data file into a JavaScript file

**Usage**

```
compile_data(input, output = tempfile(), ...)
```

**Arguments**

input A character string; the path to the input file.  
output A character string; the path to the output file.  
... Extra arguments to be passed to [to\\_json](#).

**Examples**

```
file <- system.file("test_files/test_csv.csv", package = "sketch")  
readLines(compile_data(file))
```

---

compile\_exprs                    *Compile R code into JavaScript code*

---

**Description**

Compile R code into JavaScript code

**Usage**

```
compile_exprs(x, rules = default_rules(), deparsers = default_deparasers())
```

**Arguments**

x	A character string; the expression to transpile to JS.
rules	A list of rewriting rules. See <a href="#">make_rule</a> for more detail.
deparasers	A list of deparsers. See <a href="#">make_deparser</a> for more detail.

**Value**

A character string.

**Examples**

```
compile_exprs("R + Cpp", list(make_rule('Cpp', 'JS')))  
compile_exprs("math.add(a, b)", list(make_rule('math.add', '+')))
```

---

compile\_r                        *Compile an R file into a JavaScript file*

---

**Description**

Compile an R file into a JavaScript file

**Usage**

```
compile_r(  
  input,  
  output = "",  
  rules = default_rules(),  
  deparsers = default_deparasers()  
)
```

**Arguments**

input	A character string; the input file.
output	A character string; the output file. When the output is "", the result is printed to the standard output.
rules	A list of rewriting rules. See <a href="#">make_rule</a> for more detail.
deparasers	A list of deparasers. See <a href="#">make_deparser</a> for more detail.

**Value**

A character string; the output file path.

**Examples**

```
file <- system.file("test_files/test_source.R", package = "sketch")
readLines(file)
compile_r(input = file)
```

---

convert_src	<i>Convert an asset link into a 'shiny.tag' object</i>
-------------	--

---

**Description**

Convert an asset link into a 'shiny.tag' object

**Usage**

```
convert_src(x, processors = default_processors())
```

**Arguments**

x	A character string; the header line (without the prefix #!).
processors	A list of handlers for processing the '#!' header.

**Value**

A 'shiny.tag' object.

---

default\_2\_deparsers     *A list of deparsers with additional features*

---

**Description**

Support automatic variable declaration, automatic ‘return’ and shorthand notation for the DOM module.

**Usage**

```
default_2_deparsers()
```

**Note**

lifecycle: experimental

This is used as input to [compile\\_r](#) and [compile\\_exprs](#).

**Examples**

```
default_2_deparsers()
```

---

default\_deparsers     *A list of default deparsers for deparsing JavaScript*

---

**Description**

A list of default deparsers for deparsing JavaScript

**Usage**

```
default_deparsers()
```

**Note**

This is used as input to [compile\\_r](#) and [compile\\_exprs](#).

**Examples**

```
default_deparsers()
```

---

default\_processors     *A list of handlers for processing the '#!' header*

---

**Description**

A list of handlers for processing the '#!' header

**Usage**

```
default_processors()
```

**Note**

This is used as input to [assets](#).

**Examples**

```
default_processors()
```

---

deparse\_Family     *Deparsers (specialised)*

---

**Description**

Deparsers (specialised)

- Deparser for NULL
- Deparser for NA
- Deparser for NaN
- Deparser for calls
- Deparser for infix operators
- Deparser for brackets
- Deparser for the 'for' keyword
- Deparser for the 'if' keyword
- Deparser for the 'while' keyword
- Deparser for the "function" keyword
- Deparser for the "function" keyword with explicit return
- Deparser for return
- Deparser for assignments
- Deparser for assignments (automatic variable declaration)
- Deparser for the "next" keyword

Deparser for the "try" keyword  
Deparser for the "tryCatch" keyword  
Deparser for the "throw" keyword  
Deparser for the "list" operator  
Deparser for the "data.frame" operators  
Deparser for the "summarise" operators  
Deparser for the "mutate" operators  
Deparser for the "R6Class" function  
Deparser for the "new" operator  
Deparser for the "typeof" operator  
Deparser for the "export" operator  
Deparser for the "'async" and "await" operators  
Deparser for the "let" operator  
Deparser for the "const" operator  
Deparser for the "var" operator  
Deparser for the "dataURI" operator  
Deparser for the "ifelse" operator  
Deparser for the "lambda" operator  
Deparser for the "pipe" operator  
Deparser for the "assignment pipe" operator  
Deparser for the raw string operator  
Deparser for formula  
Deparser for the "add" operator  
Deparser for the "subtract" operator  
Deparser for the "extract" operator  
Deparser for the "extractAssign" operator  
Deparser for the "extract2" operator  
Deparser for the "extract2Assign" operator  
Deparser for the HTML tags  
Deparser for the d3.js 'attr' function  
Deparser for the d3.js 'style' function  
Deparser for '.macro'  
Deparser for '.data'

**Usage**

```
deparse_sym(ast, ...)  
deparse_NULL(ast, ...)  
deparse_NA(ast, ...)  
deparse_NaN(ast, ...)  
deparse_call(ast, ...)  
deparse_infix(ast, ...)  
deparse_wrap(ast, ...)  
deparse_for(ast, ...)  
deparse_if(ast, ...)  
deparse_while(ast, ...)  
deparse_function(ast, ...)  
deparse_function_with_return(ast, ...)  
deparse_return(ast, ...)  
deparse_assignment(ast, ...)  
deparse_assignment_auto(ast, ...)  
deparse_next(ast, ...)  
deparse_try(ast, ...)  
deparse_tryCatch(ast, ...)  
deparse_throw(ast, ...)  
deparse_list(ast, ...)  
deparse_df(ast, ...)  
deparse_df_summarise(ast, ...)  
deparse_df_mutate(ast, ...)  
deparse_R6Class(ast, ...)
```

deparse\_new(ast, ...)  
deparse\_typeof(ast, ...)  
deparse\_export(ast, ...)  
deparse\_async\_await(ast, ...)  
deparse\_let(ast, ...)  
deparse\_const(ast, ...)  
deparse\_var(ast, ...)  
deparse\_dataURI(ast, ...)  
deparse\_ifelse(ast, ...)  
deparse\_lambda(ast, ...)  
deparse\_pipe(ast, ...)  
deparse\_assignment\_pipe(ast, ...)  
deparse\_raw\_string(ast, ...)  
deparse\_formula(ast, ...)  
deparse\_add(ast, ...)  
deparse\_subtract(ast, ...)  
deparse\_extract(ast, ...)  
deparse\_extractAssign(ast, ...)  
deparse\_extract2(ast, ...)  
deparse\_extract2Assign(ast, ...)  
deparse\_html\_tags(ast, ...)  
deparse\_d3\_attr(ast, ...)  
deparse\_d3\_style(ast, ...)  
deparse\_macro(ast, ...)

```
deparse_data(ast, ...)
```

**Arguments**

`ast`            A language object.  
`...`            The contextual information to be passed on to the next call.

**Value**

A character string.

**Note**

At the moment, the ``.macro`` / `deparse_macro`` function must be used with the `compile_exprs`` call. This is currently an experimental feature.

At the moment, the ``.data`` / `deparse_data`` function must be used with the `compile_exprs`` call. This is currently an experimental feature.

---

deparse\_js            *Expression deparsing for JavaScript*

---

**Description**

This is the "master" deparser that dispatches the "worker" deparsers based on the type of the input.

**Usage**

```
deparse_js(ast, deparsers)
```

**Arguments**

`ast`            language object.  
`deparsers`     A list of "typed" deparsers.

**Value**

A character string.

**Examples**

```

expr_1 <- parse_expr("R.extract(x, 3, )")
deparse_js(expr_1, basic_deparsers())
deparse_js(expr_1, default_deparsers())

expr_2 <- parse_expr("R.data_frame(x = 1, y = 2)")
deparse_js(expr_2, basic_deparsers())
deparse_js(expr_2, default_deparsers())

expr_3 <- parse_expr("lambda(x, x + 1)")
deparse_js(expr_3, basic_deparsers())

```

---

dp	<i>Constructor function to combine low-level deparsers</i>
----	--

---

**Description**

Constructor function to combine low-level deparsers

**Usage**

```
dp(...)
```

**Arguments**

... character strings indicating the features needed of the deparsers. The supported features are "basic", "r", "auto" and "dom" corresponding to the basic deparsers, the R support, the automatic variable declaration and return, and the dom short-hand notation.

**Note**

lifecycle: experimental

---

eng_sketch	<i>A language engine for 'sketch'</i>
------------	---------------------------------------

---

**Description**

This supports the use of 'sketch' code chunk in an R Markdown document.

**Usage**

```
eng_sketch(options)
```

**Arguments**

options            A list of chunk options.

**Examples**

```
# The following line makes `sketch::eng_sketch` available to `knitr::knit_engines`.
# It is usually used in the 'setup' code chunk of an R Markdown document
knitr::knit_engines$set(sketch = sketch::eng_sketch)
```

---

flatten\_filelist            *Flatten a list of files and directories into a list of files*

---

**Description**

Flatten a list of files and directories into a list of files

**Usage**

```
flatten_filelist(fs, pattern = NULL, ...)
```

**Arguments**

fs                    A character vector; a list of files.

pattern              An optional regular expression to pass to ‘list.files’ for filtering files while expanding a directory into a list of files.

...                   Additional parameters to pass to ‘list.files’.

**Examples**

```
modules_dir <- system.file("modules", package = "sketch")
flatten_filelist(modules_dir)
```

---

get\_dependencies            *Extract the content of the ‘load\_script’ headers of a sketch R file*

---

**Description**

Extract the content of the ‘load\_script’ headers of a sketch R file

**Usage**

```
get_dependencies(app_script, local_only = TRUE)
```

### Arguments

app\_script      A character string; the path to the sketch R file.  
local\_only      TRUE / FALSE; if TRUE, exclude the ones that are web link.

### Examples

```
sample_file <- system.file("test_files/test_sketch.R", package = "sketch")  
cat(readLines(sample_file), sep = "\n") # Preview the file content  
get_dependencies(sample_file)
```

---

html\_tags

*HTML templates*

---

### Description

A list of 'shiny.tag' objects describing a HTML template. The list must have signature / structure of a named list: [head = [shiny.tag], body = [shiny.tag]]

### Usage

```
default_tags(local = TRUE)
```

```
basic_tags()
```

### Arguments

local            TRUE / FALSE. If TRUE, the R base module is loaded from the local file stored in the package, otherwise, the module is served via a content delivery network (CDN).

### Examples

```
str(default_tags())
```

```
str(basic_tags())
```

---

insert_sketch	<i>Insert a 'sketch' app into an R Markdown document</i>
---------------	--

---

**Description**

Insert a 'sketch' app into an R Markdown document

**Usage**

```
insert_sketch(file, id, output_dir = NULL, render = TRUE, ...)
```

**Arguments**

file	A character string; the path to the 'sketch' file.
id	A character string; an unique identifier for the 'sketch' file. Needed only when output_dir is not NULL.
output_dir	A character string; a separate directory to save the 'sketch' app. Default to be NULL, which embeds the app in the Rmd file.
render	TRUE or FALSE; if TRUE, call <a href="#">doRenderTags</a> ; if FALSE, return the 'shiny.tag' object.
...	(Optional) Other attributes to pass to iframes. Also supports the 'rules', 'deparers' and 'debug' options to pass to 'source_r'.

**Value**

An HTML string if render is TRUE, or a 'shiny.tag' object if render is FALSE.

**Examples**

```
# In an R code chunk of an R Markdown document
file <- system.file("test_files/test_RMD.R", package = "sketch")
insert_sketch(file, style = "width:500px; height:500px;", render = FALSE)
```

---

is_call	<i>Predicate for calls</i>
---------	----------------------------

---

**Description**

Predicate for calls

**Usage**

```
is_call(x, name = NULL, n = NULL, ns = NULL)
```

**Arguments**

x	An object to test. If a formula, the right-hand side is extracted.
name	An optional name that the call should match. It is passed to <code>sym()</code> before matching. This argument is vectorised and you can supply a vector of names to match. In this case, <code>is_call()</code> returns TRUE if at least one name matches.
n	An optional number of arguments that the call should match.
ns	The namespace of the call. If NULL, the namespace doesn't participate in the pattern-matching. If an empty string "" and x is a namespaced call, <code>is_call()</code> returns FALSE. If any other string, <code>is_call()</code> checks that x is namespaced within ns.  Can be a character vector of namespaces, in which case the call has to match at least one of them, otherwise <code>is_call()</code> returns FALSE.

**Note**

This function is imported from 'rlang'.

---

is\_Family

*Predicate for symbols, i.e. symbols or syntactic literals*

---

**Description**

Predicate for symbols, i.e. symbols or syntactic literals

Predicate for infix operators

Predicate for brackets

Predicate for the 'for' keyword

Predicate for the 'if' keyword

Predicate for the 'while' keyword

Predicate for the "function" keyword

Predicate for return

Predicate for assignments

Predicate for assignments

Predicate for the "break" keyword

Predicate for the "next" keyword

Predicate for the "try" keyword

Predicate for the "tryCatch" keyword

Predicate for the "throw" keyword

Predicate for the "list" operator

Predicate for the "data.frame" operators

Predicate for the "summarise" operators

Predicate for the "mutate" operators  
Predicate for the "R6Class" function  
Predicate for the "new" operator  
Predicate for the "typeof" operator  
Predicate for the "export" operator  
Predicate for the "async" and "await" operators  
Predicate for the "let" operator  
Predicate for the "const" operator  
Predicate for the "var" operator  
Predicate for the "dataURI" operator  
Predicate for the "ifelse" operator  
Predicate for the "lambda" operator  
Predicate for the "pipe" operator  
Predicate for the "assignment pipe" operator  
Predicate for the raw string operator  
Predicate for formula  
Predicate for the "add" operator  
Predicate for the "subtract" operator  
Predicate for the "extract" operator  
Predicate for the "extractAssign" operator  
Predicate for the "extract2" operator  
Predicate for the "extract2Assign" operator  
Predicate for the HTML tags  
Predicate for the d3.js 'attr' function  
Predicate for the d3.js 'style' function  
Predicate for '.macro'  
Predicate for '.data'

**Usage**

`is_sym(ast)`

`is_infix(ast)`

`is_wrap(ast)`

`is_call_for(ast)`

`is_call_if(ast)`

is\_call\_while(ast)  
is\_call\_function(ast)  
is\_call\_return(ast)  
is\_call\_assignment(ast)  
is\_call\_assignment\_auto(ast)  
is\_call\_break(ast)  
is\_call\_next(ast)  
is\_call\_try(ast)  
is\_call\_tryCatch(ast)  
is\_call\_throw(ast)  
is\_call\_list(ast)  
is\_call\_df(ast)  
is\_call\_df\_summarise(ast)  
is\_call\_df\_mutate(ast)  
is\_call\_R6Class(ast)  
is\_call\_new(ast)  
is\_call\_typeof(ast)  
is\_call\_export(ast)  
is\_call\_async\_await(ast)  
is\_call\_let(ast)  
is\_call\_const(ast)  
is\_call\_var(ast)  
is\_call\_dataURI(ast)  
is\_call\_ifelse(ast)

`is_call_lambda(ast)`  
`is_call_pipe(ast)`  
`is_call_assignment_pipe(ast)`  
`is_call_raw_string(ast)`  
`is_call_formula(ast)`  
`is_call_add(ast)`  
`is_call_subtract(ast)`  
`is_call_extract(ast)`  
`is_call_extractAssign(ast)`  
`is_call_extract2(ast)`  
`is_call_extract2Assign(ast)`  
`is_html_tags(ast)`  
`is_d3_attr(ast)`  
`is_d3_style(ast)`  
`is_macro(ast)`  
`is_data(ast)`

**Arguments**

`ast`            A language object.

---

`is_syntactic_literal`    *Predicate for syntactic literal*

---

**Description**

Predicate for syntactic literal

**Usage**

`is_syntactic_literal(x)`

**Arguments**

x                    An object to test.

**Note**

This function is imported from 'rlang'.

---

let-declare-const      *Empty functions*

---

**Description**

These functions do nothing. It is created to ensure the keywords 'let' and 'declare' are defined.

**Usage**

```
let(...)  
declare(...)  
const(...)
```

**Arguments**

...                    Any arguments

**Examples**

```
let (x)  
let (x = 1, y = 2)  
declare (x1, x2, x3)
```

---

license\_info            *License information*

---

**Description**

License information

**Usage**

```
license_info(x)
```

**Arguments**

x                    A character string; name of the library / assets.

**Value**

A named list containing the license information and the link from which the information is extracted.

**Examples**

```
license_info("mathjs")  
license_info("p5")
```

---

list-of-deparsers	<i>Low-level lists of deparsers</i>
-------------------	-------------------------------------

---

**Description**

Support of R functionalities

**Usage**

```
dp_r_support()  
  
dp_auto()  
  
dp_dom()  
  
dp_d3()  
  
dp_macro()
```

---

load_Family	<i>Header functions</i>
-------------	-------------------------

---

**Description**

Header functions

**Usage**

```
load_library(package, ...)  
  
load_script(src, ...)  
  
load_data(x, cache = tempfile(), ...)
```

**Arguments**

package	A character string; name of a JavaScript library.
...	Additional arguments to pass to header processor.
src	A character string; the full web/local path to a JavaScript library.
x	A character string; the full path to the file containing the data.
cache	A character string; the full path to the cache file.

---

local	<i>A helper function to enable debugger option</i>
-------	--

---

**Description**

A helper function to enable debugger option

**Usage**

```
local(x, from_local = TRUE)
```

**Arguments**

x	TRUE / FALSE; whether to attach a debugging console to the sketch application.
from_local	TRUE / FALSE; whether to load the debugger console from the local package. If FALSE, the console will be loaded from a Content Delivery Network (CDN) link.

**Note**

Use 'from\_local=TRUE' for self-contained applications, and 'from\_local=FALSE' for reduced file size.

**Examples**

```
# This function is designed to be used in the configuration header, e.g.  
# config(debug = local(TRUE), rules = basic_rules(), deparsers = basic_deparsers())
```

```
local(TRUE)
```

---

make_deparser	<i>A constructor for a "typed" deparser</i>
---------------	---

---

**Description**

A constructor for a "typed" deparser

**Usage**

```
make_deparser(predicate_fun, deparse_fun)
```

**Arguments**

predicate\_fun A function that takes a "lang" object and return a logical.

deparse\_fun A function that takes a "lang" object and return a character string.

**Value**

A list; a deparser ready to be dispatched by "type".

**Examples**

```
str(make_deparser(predicate_fun = rlang::is_call, deparse_fun = deparse))
```

---

make_processor	<i>Make a handle to process header</i>
----------------	--

---

**Description**

Make a handle to process header

**Usage**

```
make_processor(pred, fun)
```

**Arguments**

pred A function, taking a string and returning a logical.

fun A function, taking a string and returning a 'shiny.tag' object.

**Value**

A header processor / handler.

---

make_rule	<i>Make a AST transformation rule</i>
-----------	---------------------------------------

---

**Description**

Make a AST transformation rule

**Usage**

```
make_rule(from, to)
```

**Arguments**

from	A character string.
to	A character string.

**Value**

A function that takes a language object and returns a language object.

**Examples**

```
library(sketch)

rule_1 <- make_rule("pi", "Math.PI")
expr <- rlang::parse_expr("2 * (3 + pi)")

rule_1(expr) # this works but is not the preferred usage
rewrite(expr, list(rule_1)) # this is preferred

rule_2 <- make_rule("+", "Math.add")
rewrite(expr, list(rule_1, rule_2))
```

---

parse_expr	<i>Parse R code</i>
------------	---------------------

---

**Description**

Parse R code

**Usage**

```
parse_expr(x)
```

**Arguments**

- x                   Text containing expressions to parse\_expr for parse\_expr() and parse\_exprs(). Can also be an R connection, for instance to a file. If the supplied connection is not open, it will be automatically closed and destroyed.

**Note**

This function is imported from 'rlang'.

**Examples**

```
parse_expr("x <- 1 + 1")
```

---

print.sketch\_rule        *Print function for 'sketch\_rule' objects*

---

**Description**

Print function for 'sketch\_rule' objects

**Usage**

```
## S3 method for class 'sketch_rule'  
print(x, ...)
```

**Arguments**

- x                   A 'sketch\_rule' object.  
...                 (Unused) Optional arguments.

**Examples**

```
library(sketch)  
rule_1 <- make_rule("+", "Math.add")  
print(rule_1)
```

---

r-to-js-rules	<i>Mapping R operators into JavaScript operators</i>
---------------	--

---

**Description**

Mapping R operators into JavaScript operators

**Usage**

```
basic_rules()
default_rules()
```

**Note**

These functions are used as inputs to [compile\\_r](#) and [compile\\_exprs](#).

**References**

R operators: <https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Operators>

R infix and prefix operators: <https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Infix-and-prefix-operators>

JavaScript operators: [https://www.w3schools.com/js/js\\_operators.asp](https://www.w3schools.com/js/js_operators.asp)

**Examples**

```
basic_rules()
default_rules()
```

---

read_multilines	<i>Read one or more lines from the console for the first successful parse</i>
-----------------	---

---

**Description**

`read_multilines` reads one or more lines from the terminal (in interactive use).

**Usage**

```
read_multilines(prompt = "")
```

**Arguments**

prompt	the string printed when prompting the user for input. Should usually end with a space " ".
--------	--

## Details

This function repeatedly calls `readline` until enough inputs are provided to reach a successful parse.

This can only be used in an interactive session.

## Examples

```
## Not run:
# In an interactive session
read_multilines()
1 + 2 # expect immediate success

read_multilines()
1 +
2 +
3 # expect success here

## End(Not run)
```

---

rewrite

*Interface for AST rewriting*

---

## Description

Interface for AST rewriting

## Usage

```
rewrite(ast, rules)
```

## Arguments

`ast` A language object.  
`rules` A list of functions, each of which is the output from `'make_rule'`.

## Value

A language object.

## Examples

```
library(sketch)

rewrite(
  ast = rlang::parse_expr("2 * (3 + pi)"),
  rules = list(make_rule("pi", "Math.PI"))
)
```

```

rewrite(
  ast = rlang::parse_expr("2 + pi"),
  rules = list(
    make_rule("pi", "Math.PI"),
    make_rule("+", "Math.add")
  )
)

```

---

runShinyApp	<i>Run 'Shiny' Application</i>
-------------	--------------------------------

---

### Description

Run 'Shiny' Application

### Usage

```
runShinyApp()
```

### Examples

```

## Not run:
runShinyApp()

## End(Not run)

```

---

safeguard	<i>Perform pre-transpilation check</i>
-----------	--

---

### Description

Perform pre-transpilation check

### Usage

```
safeguard(ast, rules, deparsers)
```

### Arguments

ast	A language object.
rules	A list of functions; the rewriting rules, each of which is the output from <a href="#">make_rule</a> .
deparsers	A list of functions; the deparsers, each of which is the output from <a href="#">make_deparser</a> .

**Value**

TRUE when the check is complete.

**Examples**

```
# Expect no warning
safeguard(parse_expr("a <- 3"),
           rules = default_rules(),
           deparsers = default_deparasers())

# Expect a warning (as `max` is reserved to be a function by the transpiler)
safeguard(parse_expr("max <- 3"),
           rules = default_rules(),
           deparsers = default_deparasers())
```

---

source_active	<i>Source active file in 'RStudio'</i>
---------------	--

---

**Description**

Source active file in 'RStudio'

**Usage**

```
source_active(...)
```

**Arguments**

... Optional arguments to pass to source\_r.

**Examples**

```
## Not run:
# At 'RStudio', opens a 'sketch' R file in the editor, then
# run the following:
source_active() # This launches the default HTML viewer.

## End(Not run)
```

---

source_js	<i>Serve a compiled 'sketch' JavaScript file</i>
-----------	--

---

### Description

Serve a compiled 'sketch' JavaScript file

### Usage

```
source_js(file, debug = FALSE, asset_tags = default_tags(), launch_browser)
```

### Arguments

file	A character string; path to the compiled JS file.
debug	TRUE or FALSE; if TRUE, a console for debugging is attached to your app.
asset_tags	An optional list of 'shiny.tag' objects to be added to the html template. The list must have signature / structure of a named list: [head = [shiny.tag], body = [shiny.tag]], containing the head and body elements, each of which is a list of shiny.tag object.
launch_browser	A character string; "viewer" or "browser", which calls 'rstudioapi::viewer' and 'utils::browseURL' respectively; use NULL to suppress display.

### Examples

```
## Not run:
file <- system.file("test_files/test_source.js", package = "sketch")
# The next line launches the default HTML browser
source_js(file, debug = TRUE, launch_browser = "browser")

## End(Not run)
```

---

source_r	<i>Source a 'sketch' R file</i>
----------	---------------------------------

---

### Description

This function compiles a 'sketch' R file, resolves the dependencies and serves it in the viewer.

### Usage

```
source_r(file, debug = FALSE, launch_browser, asset_tags = default_tags(), ...)
```

**Arguments**

file	A character string; path to the R file.
debug	TRUE or FALSE; if TRUE, a console for debugging is attached to your app.
launch_browser	A character string; "viewer" or "browser", which calls 'rstudioapi::viewer' and 'utils::browseURL' respectively; use NULL to suppress display.
asset_tags	An optional list of 'shiny.tag' objects to be added to the html template. The list must have signature / structure of a named list: [head = [shiny.tag], body = [shiny.tag]],
...	Additional arguments to pass to 'compile_r'.

**Examples**

```
## Not run:
file <- system.file("test_files/test_source.R", package = "sketch")
# The next line launches the default HTML browser
source_r(file, debug = TRUE, launch_browser = "browser")

## End(Not run)
```

---

split\_rules

*Split rules for customisation*


---

**Description**

This function is the left-inverse of 'combine\_rules', i.e. `split_rules(combine_rules(rs, group)) = rs` for any variable 'group'. It is created to facilitate the addition or removal of rewriting rules.

**Usage**

```
split_rules(rs)
```

**Arguments**

rs	A list of (grouped) rewriting rules. Note that a list of n rules without grouping is a list of n groups of single rule.
----	---

src *Get the source link of a JavaScript library*

---

**Description**

Get the source link of a JavaScript library

**Usage**

```
src(x)
```

**Arguments**

x                    A character string; name of the JavaScript library

**Value**

A character string; the path to the library.

**Examples**

```
src("mathjs")  
src("p5")
```

---

test\_sketch *Test a sketch application*

---

**Description**

Test a sketch application

**Usage**

```
test_sketch(app_script, test_script, port = 9454, ...)
```

**Arguments**

app\_script        A character string; the file path to the app.  
test\_script       A character string; the file path to the tests.  
port               An integer to pass to `'websocket$new()'`.  
...                Additional arguments to pass to [source\\_r](#).

**Value**

A "websocket" object.

**Examples**

```
## Not run:
app_file <- system.file("test_files/test_testthat_app.R", package = "sketch")
test_file <- system.file("test_files/test_testthat_test.R", package = "sketch")
# This following command will launch the default browser
res <- test_sketch(app_file, test_file)

## End(Not run)
```

---

to\_json

*Convert a file into a JavaScript expression*


---

**Description**

It supports csv and json by default and lets users provide custom handlers if other file formats are used.

**Usage**

```
to_json(input, as_data_frame, read_fun, ...)
```

**Arguments**

input	A character string; the path to the input file.
as_data_frame	TRUE or FALSE; whether the data are loaded as a data-frame.
read_fun	A function to load the input file. Default settings are provided for CSV files and JSON files. The function has to load a data file into an object that can be handled by 'jsonlite::toJSON'. Possible choices include 'utils::read_delim', 'readr::read_csv2', etc.
...	Extra arguments to be passed to 'read_fun'.

---

websocket

*Websocket for 'sketch' applications*


---

**Description**

This combines the \*-Server family of functions in 'httpuv' with the transpilation functionality provided by 'sketch'.

### Public fields

`app` A list of functions that define the application.

`server` A server handle to be used by 'stopServer'.

`log` A character vector that keep tracks of all the commands sent to the browser session.

`ws` A WebSocket channel to handle the communication between the R session and the browser session.

`in_handler` A function to handle instructions sent by the browser session.

`out_handler` A function to handle instruction sent to the browser session.

`env` An environment to store variables temporarily.

`port` An integer; the TCP port number.

`message` TRUE or FALSE; whether to display a prompt when a server is started and when it is stopped.

`connected` TRUE or FALSE; whether a connection has been established. One should ways start the WebSocket server before visiting the web page that connects to the server.

`started` TRUE or FALSE; whether a server has been started. Use the startServer method to start a server.

### Methods

#### Public methods:

- `websocket$startServer()`
- `websocket$stopServer()`
- `websocket$listServers()`
- `websocket$stopAllServers()`
- `websocket$sketch_mode()`
- `websocket$new_app()`
- `websocket$new()`
- `websocket$clone()`

**Method** `startServer()`: Start a WebSocket server

*Usage:*

```
websocket$startServer()
```

**Method** `stopServer()`: Stop a WebSocket server

*Usage:*

```
websocket$stopServer()
```

**Method** `listServers()`: List all running WebSocket servers

*Usage:*

```
websocket$listServers()
```

**Method** `stopAllServers()`: Stop all running WebSocket servers

*Usage:*

```
websocket$stopAllServers()
```

**Method** `sketch_mode()`: Enter sketch mode, in which all commands go through the transpiler before reaching the browser session.

*Usage:*

```
websocket$sketch_mode()
```

**Method** `new_app()`: Create a blank HTML page with interactive access. This function is designed for newcomers.

*Usage:*

```
websocket$new_app(
  preamble = list(library = c(), script = c(), data = c()),
  ...
)
```

*Arguments:*

`preamble` (Optional) A named list; the preamble to include. Use the name 'lib' for arguments to `load_library`, 'script' for arguments to `load_script` and 'data' for arguments to `load_data`. Note that the "dom" and "websocket" modules are required and loaded by default.

... Extra parameters to pass to [source\\_r](#).

*Returns:* The (invisible) temporary file path to the app.

**Method** `new()`: Initialise a WebSocket connection

*Usage:*

```
websocket$new(in_handler, out_handler, message = TRUE, port = 9454)
```

*Arguments:*

`in_handler` A function to handle incoming message, default to be [print](#) which only displays the message without any processing.

`out_handler` A function to handle outgoing message, default to be [compile\\_exprs](#) which transpiles R commands into JavaScript commands.

`message` TRUE or FALSE; whether to display a prompt when a server is started and when it is stopped.

`port` An integer; the TCP port number.

*Returns:* A 'websocket' object.

*Examples:*

```
\dontrun{
# Launch a WebSocket server
ws <- websocket$new()
ws$startServer()
ws$listServers() # Check that a server is running

# Launch a 'sketch' application with WebSocket functionality
file <- system.file("test_files/test_websocket.R", package = "sketch")
source_r(file, debug = TRUE) # Launch the default browser
}
```

```

# Enter sketch mode to send commands to the application
ws$sketch_mode()
# Within sketch mode
print("1234")
x <- 10
print(x + 1)
q()

# Back to normal mode, inspect the log and stop the server
ws$log
ws$stopServer()
ws$listServers() # Confirm no server is running
}

```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
websocket$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```

## -----
## Method `websocket$new`
## -----

## Not run:
# Launch a WebSocket server
ws <- websocket$new()
ws$startServer()
ws$listServers() # Check that a server is running

# Launch a 'sketch' application with WebSocket functionality
file <- system.file("test_files/test_websocket.R", package = "sketch")
source_r(file, debug = TRUE) # Launch the default browser

# Enter sketch mode to send commands to the application
ws$sketch_mode()
# Within sketch mode
print("1234")
x <- 10
print(x + 1)
q()

# Back to normal mode, inspect the log and stop the server
ws$log
ws$stopServer()
ws$listServers() # Confirm no server is running

```

```
## End(Not run)
```

# Index

assets, [3, 9](#)

basic\_deparsers, [3](#)  
basic\_rules (r-to-js-rules), [28](#)  
basic\_tags (html\_tags), [16](#)  
bundle, [4](#)

combine\_rules, [4](#)  
compile\_active, [5](#)  
compile\_data, [5](#)  
compile\_exprs, [3, 6, 8, 28, 37](#)  
compile\_r, [3, 6, 8, 28](#)  
const (let-declare-const), [22](#)  
convert\_src, [3, 7](#)

declare (let-declare-const), [22](#)  
default\_2\_deparsers, [8](#)  
default\_deparsers, [8](#)  
default\_processors, [9](#)  
default\_rules (r-to-js-rules), [28](#)  
default\_tags (html\_tags), [16](#)  
deparse\_add (deparse\_Family), [9](#)  
deparse\_assignment (deparse\_Family), [9](#)  
deparse\_assignment\_auto  
(deparse\_Family), [9](#)  
deparse\_assignment\_pipe  
(deparse\_Family), [9](#)  
deparse\_async\_await (deparse\_Family), [9](#)  
deparse\_call (deparse\_Family), [9](#)  
deparse\_const (deparse\_Family), [9](#)  
deparse\_d3\_attr (deparse\_Family), [9](#)  
deparse\_d3\_style (deparse\_Family), [9](#)  
deparse\_data (deparse\_Family), [9](#)  
deparse\_dataURI (deparse\_Family), [9](#)  
deparse\_df (deparse\_Family), [9](#)  
deparse\_df\_mutate (deparse\_Family), [9](#)  
deparse\_df\_summarise (deparse\_Family), [9](#)  
deparse\_export (deparse\_Family), [9](#)  
deparse\_extract (deparse\_Family), [9](#)  
deparse\_extract2 (deparse\_Family), [9](#)  
deparse\_extract2Assign  
(deparse\_Family), [9](#)  
deparse\_extractAssign (deparse\_Family),  
[9](#)  
deparse\_Family, [9](#)  
deparse\_for (deparse\_Family), [9](#)  
deparse\_formula (deparse\_Family), [9](#)  
deparse\_function (deparse\_Family), [9](#)  
deparse\_function\_with\_return  
(deparse\_Family), [9](#)  
deparse\_html\_tags (deparse\_Family), [9](#)  
deparse\_if (deparse\_Family), [9](#)  
deparse\_ifelse (deparse\_Family), [9](#)  
deparse\_infix (deparse\_Family), [9](#)  
deparse\_js, [3, 13](#)  
deparse\_lambda (deparse\_Family), [9](#)  
deparse\_let (deparse\_Family), [9](#)  
deparse\_list (deparse\_Family), [9](#)  
deparse\_macro (deparse\_Family), [9](#)  
deparse\_NA (deparse\_Family), [9](#)  
deparse\_NaN (deparse\_Family), [9](#)  
deparse\_new (deparse\_Family), [9](#)  
deparse\_next (deparse\_Family), [9](#)  
deparse\_NULL (deparse\_Family), [9](#)  
deparse\_pipe (deparse\_Family), [9](#)  
deparse\_R6Class (deparse\_Family), [9](#)  
deparse\_raw\_string (deparse\_Family), [9](#)  
deparse\_return (deparse\_Family), [9](#)  
deparse\_subtract (deparse\_Family), [9](#)  
deparse\_sym (deparse\_Family), [9](#)  
deparse\_throw (deparse\_Family), [9](#)  
deparse\_try (deparse\_Family), [9](#)  
deparse\_tryCatch (deparse\_Family), [9](#)  
deparse\_typeof (deparse\_Family), [9](#)  
deparse\_var (deparse\_Family), [9](#)  
deparse\_while (deparse\_Family), [9](#)  
deparse\_wrap (deparse\_Family), [9](#)  
doRenderTags, [17](#)  
dp, [14](#)

- dp\_auto (list-of-deparsers), 23
- dp\_d3 (list-of-deparsers), 23
- dp\_dom (list-of-deparsers), 23
- dp\_macro (list-of-deparsers), 23
- dp\_r\_support (list-of-deparsers), 23
- eng\_sketch, 14
- flatten\_filelist, 15
- get\_dependencies, 15
- html\_tags, 16
- insert\_sketch, 17
- is\_call, 17
- is\_call\_add (is\_Family), 18
- is\_call\_assignment (is\_Family), 18
- is\_call\_assignment\_auto (is\_Family), 18
- is\_call\_assignment\_pipe (is\_Family), 18
- is\_call\_async\_await (is\_Family), 18
- is\_call\_break (is\_Family), 18
- is\_call\_const (is\_Family), 18
- is\_call\_dataURI (is\_Family), 18
- is\_call\_df (is\_Family), 18
- is\_call\_df\_mutate (is\_Family), 18
- is\_call\_df\_summarise (is\_Family), 18
- is\_call\_export (is\_Family), 18
- is\_call\_extract (is\_Family), 18
- is\_call\_extract2 (is\_Family), 18
- is\_call\_extract2Assign (is\_Family), 18
- is\_call\_extractAssign (is\_Family), 18
- is\_call\_for (is\_Family), 18
- is\_call\_formula (is\_Family), 18
- is\_call\_function (is\_Family), 18
- is\_call\_if (is\_Family), 18
- is\_call\_ifelse (is\_Family), 18
- is\_call\_lambda (is\_Family), 18
- is\_call\_let (is\_Family), 18
- is\_call\_list (is\_Family), 18
- is\_call\_new (is\_Family), 18
- is\_call\_next (is\_Family), 18
- is\_call\_pipe (is\_Family), 18
- is\_call\_R6Class (is\_Family), 18
- is\_call\_raw\_string (is\_Family), 18
- is\_call\_return (is\_Family), 18
- is\_call\_subtract (is\_Family), 18
- is\_call\_throw (is\_Family), 18
- is\_call\_try (is\_Family), 18
- is\_call\_tryCatch (is\_Family), 18
- is\_call\_typeof (is\_Family), 18
- is\_call\_var (is\_Family), 18
- is\_call\_while (is\_Family), 18
- is\_d3\_attr (is\_Family), 18
- is\_d3\_style (is\_Family), 18
- is\_data (is\_Family), 18
- is\_Family, 18
- is\_html\_tags (is\_Family), 18
- is\_infix (is\_Family), 18
- is\_macro (is\_Family), 18
- is\_sym (is\_Family), 18
- is\_syntactic\_literal, 21
- is\_wrap (is\_Family), 18
- let (let-declare-const), 22
- let-declare-const, 22
- license\_info, 22
- list-of-deparsers, 23
- load\_data (load\_Family), 23
- load\_Family, 23
- load\_library (load\_Family), 23
- load\_script (load\_Family), 23
- local, 24
- make\_deparser, 6, 7, 25, 30
- make\_processor, 25
- make\_rule, 4, 6, 7, 26, 30
- parse\_expr, 26
- print, 37
- print.sketch\_rule, 27
- r-to-js-rules, 28
- read\_multilines, 28
- rewrite, 29
- runShinyApp, 30
- safeguard, 30
- sketch (sketch-package), 2
- sketch-package, 2
- source\_active, 31
- source\_js, 32
- source\_r, 32, 34, 37
- split\_rules, 33
- src, 34
- sym(), 18
- test\_sketch, 34
- to\_json, 5, 35

websocket, [35](#)