Package 'sass'

March 18, 2020

Type Package

Version 0.2.0

Title Syntactically Awesome Style Sheets ('Sass')

Description An 'SCSS' compiler, powered by the 'LibSass' library. With this, R developers can use variables, inheritance, and functions to generate dynamic style sheets. The package uses the 'Sass CSS' extension language, which is stable, powerful, and CSS compatible.

License MIT + file LICENSE

URL https://github.com/rstudio/sass

BugReports https://github.com/rstudio/sass/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

SystemRequirements GNU make

Imports digest, fs, rlang, htmltools

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Joe Cheng [aut],
Timothy Mastny [aut],
Richard Iannone [aut] (<https://orcid.org/0000-0003-3925-190X>),
Barret Schloerke [aut] (<https://orcid.org/0000-0001-9986-114X>),
Carson Sievert [aut, cre] (<https://orcid.org/0000-0002-4958-2844>),
RStudio [cph, fnd],
Sass Open Source Foundation [ctb, cph] (LibSass library),
Greter Marcel [ctb, cph] (LibSass library),
Mifsud Michael [ctb, cph] (LibSass library),
Hampton Catlin [ctb, cph] (LibSass library),
Natalie Weizenbaum [ctb, cph] (LibSass library),
Chris Eppstein [ctb, cph] (LibSass library),
Adams Joseph [ctb, cph] (json.cpp),
Trifunovic Nemanja [ctb, cph] (utf8.h)

Maintainer Carson Sievert <carson@rstudio.com>

Repository CRAN

Date/Publication 2020-03-18 15:50:09 UTC

R topics documented:

`as_sass` *List to Sass converter*

Description

Converts multiple types of inputs to a single Sass input string for `sass`.

Usage

```
as_sass(input)
```

Arguments

`input`	Either a • raw Sass string • named list containing variable names and values • Sass-like file name.

Details

Note that the LibSass compiler expects .sass files to use the Sass Indented Syntax.

Value

a single character value to be supplied to `sass`

See Also

Visit `https://sass-lang.com/documentation/file.SASS_REFERENCE.html#import` for more details.

Examples

```
# Example of regular Sass input
as_sass("body { color: \"blue\"; }")

# There is support for adding variables
as_sass(
  list(
    list(color = "blue"),
   "body { color: $color; }"
   )
)


# Add a file name
someFile <- tempfile("variables")

# Overwrite color to red
write("$color: \"red\";", someFile)

input <-
  as_sass(
    list(
      list(color = "blue"),
      sass_file(someFile),
      "body { color: $color; }"
      )
   )

input

# The final body color is red
sass(input)
```

`sass` *Compile Sass to CSS*

Description

Compile Sass to CSS using LibSass.

Usage

```
sass(
  input = NULL,
  options = sass_options(),
  output = NULL,
  cache_options = sass_cache_options(),
  write_attachments = NA
)
```

Arguments

input	Accepts raw Sass, a named list of variables, or a list of raw Sass and/or named variables. See as_sass and sass_import / sass_file for more details.
options	Compiler options for Sass. Please specify options using sass_options.
output	Specifies path to output file for compiled CSS.
cache_options	Caching options for Sass. Please specify options using sass_cache_options. Caching is turned off by default for interactive R sessions, and turned on for non-interactive ones.
write_attachments	If the input contains sass_layer objects that have file attachments, and output is not NULL, then copy the file attachments to the directory of output. (Defaults to NA, which merely emits a warning if file attachments are present, but does not write them to disk; the side-effect of writing extra files is subtle and potentially destructive, as files may be overwritten.)

Value

If output = NULL, the function returns a string value of the compiled CSS. If the output path is specified, the compiled CSS is written to that file and invisible() is returned.

See Also

http://sass-lang.com/guide

Examples

```
# raw Sass input
sass("foo { margin: 122px * .3; }")

# list of inputs, including named variables
sass(list(
  list(width = "122px"),
  "foo { margin: $width * .3; }"
))

# import a file
tmp_file <- tempfile()
writeLines("foo { margin: $width * .3; }", tmp_file)
sass(list(
  list(width = "122px"),
  sass_file(tmp_file)
))
```

sass_cache_options *Caching Options for Sass*

Description

Specifies whether caching is used with sass, and where on the file system the cached data should be stored. Used with sass.

Usage

```
sass_cache_options(
  cache = getOption("sass.cache", !interactive()),
  cache_dir = getOption("sass.cache_dir", tempdir())
)
```

Arguments

cache	Logical value indicating whether caching is performed. If no value is provided, the R option sass.cache is consulted; if the option is not set, then caching is performed only if the R session is not interactive.
cache_dir	The directory which will be used to store cache files. If no value is provided, the R option sass.cache_dir is consulted; if the option is not set, then tempdir is used. Note that this means that caches will not persist beyond the current R session by default, since tempdir() is unique to each session.

Details

If caching is enabled, sass() will attempt to bypass the compilation process by reusing output from previous sass() calls that used equivalent inputs. This mechanism works by using a hashing algorithm to derive a *cache key* from each sass() call's input and options arguments. If a file named *CACHE_KEY*.sasscache.css exists within the cache directory, its contents are used instead of performing the compilation. If the file does not exist, then compilation is performed and usual and the results are stored at that file path for next time.

If a file that is included using sass_file changes on disk (i.e. its last-modified time changes), its previous cache entries will effectively be invalidated (not removed from disk, but they'll no longer be matched). However, if a file imported using sass_file itself imports other sass files using @import, changes to those files are invisible to the cache and you will end up with stale results.

If a cache directory is explicitly specified (either via the cache_dir argument or via the sass.cache_dir R option), note that this package does not do any cleanup of that directory. If disk space is a concern, you will need to delete older entries from that directory yourself.

Examples

```
# Very slow to compile
fib_sass <- "@function fib($x) {
  @if $x <= 1 {
    @return $x
```

```
  }
  @return fib($x - 2) + fib($x - 1);
}

body {
  width: fib(27);
}"

# Use a custom cache dir for the purposes of this example. Normally,
# you'd want to set the caching behavior using options().
temp_cache_dir <- tempfile("sass_cache_dir")
dir.create(temp_cache_dir)
cache_opts <- sass_cache_options(TRUE, cache_dir = temp_cache_dir)

# The first time this runs it will be very slow
system.time(sass(fib_sass, cache_options = cache_opts))

# But on subsequent calls, it should be very fast
system.time(sass(fib_sass, cache_options = cache_opts))
```

`sass_import` *Sass Import*

Description

Create an import statement to be used within your Sass file. See https://sass-lang.com/documentation/file.SASS_REFERENCE.html#import for more details.

Usage

```
sass_import(input, quote = TRUE)

sass_file(input)
```

Arguments

`input`	Character string to be placed in an import statement.
`quote`	Logical that determines if a double quote is added to the import value. Defaults to `TRUE`.

Details

`sass_file` adds extra checks to make sure an appropriate file path exists given the input value.

Value

Fully defined Sass import string.

Examples

```
sass_import("foo")
sass_import("$foo", FALSE)

tmp_scss_file <- tempfile(fileext = ".scss")
writeLines("$color: red; body{ color: $color; }", tmp_scss_file)
sass_file(tmp_scss_file)
sass(sass_file(tmp_scss_file))
```

---

sass_layer *Sass layer objects*

Description

Sass layers are a way to group a set of related Sass variable definitions, function/mixin declarations, and CSS rules into a single object. Use `sass_layer()` to create these objects, and `sass_layer_merge()` to combine two or more layer objects into a single layer; this ability to be merged is the main benefit of using Sass layers versus lower-level forms of sass input.

Usage

```
sass_layer_merge(...)

sass_layer(
  defaults = "",
  declarations = "",
  rules = "",
  html_deps = NULL,
  file_attachments = character(0),
  tags = character(0)
)
```

Arguments

`...`	A collection of `sass_layer()`s and/or objects that `as_sass()` understands. Arguments should be provided in reverse priority order: defaults, declarations, and rules in later layers will take precedence over those of previous layers. Non-layer values will be converted to layers by calling `sass_layer(rules = ...)`.
`defaults`	A suitable `sass::as_sass()` `input`. Intended for declaring variables with `!default`. When layers are combined, defaults are merged in reverse order; that is, `sass_layer_merge(layer1,lay` will include `layer2$defaults` before `layer1$defaults`.
`declarations`	A suitable `sass::as_sass()` `input`. Intended for function and mixin declarations, and variable declarations without `!default`; not intended for actual CSS rules. These will be merged in forward order; that is, `sass_layer_merge(layer1,layer2)` will include `layer1$declarations` before `layer2$declarations`.

rules — A suitable sass::as_sass() input. Intended for actual CSS rules. These will be merged in forward order; that is, sass_layer_merge(layer1,layer2) will include layer1$rules before layer2$rules.

html_deps — An HTML dependency (or a list of them).

file_attachments — A named character vector, representing file assets that are referenced (using relative paths) from the sass in this layer. The vector names should be a relative path, and the corresponding vector values should be absolute paths to files or directories that exist; at render time, each value will be copied to the relative path indicated by its name. (For directories, the *contents* of the source directory will be copied into the destination directory; the directory itself will not be copied.) You can also omit the name, in which case that file or directory will be copied directly into the output directory.

tags — A character vector with zero or more elements. Can be used to preserve simple metadata as layers are merged.

Examples

```
blue <- list(color = "blue !default")
red <- list(color = "red !default")
green <- list(color = "green !default")

# a sass_layer() by itself is not very useful, it just defines some
# SASS to place before (defaults) and after (declarations, rules)
core <- sass_layer(defaults = blue, rules = "body { color: $color; }")
core
sass(core)

# However, by stacking sass_layer()s, we have ability to place
# SASS both before and after some other sass (e.g., core)
# Here we place a red default _before_ the blue default and export the
# color SASS variable as a CSS variable _after_ the core
red_layer <- sass_layer(red, rules = ":root{ --color: #{$color}; }")
sass(sass_layer_merge(core, red_layer))
sass(sass_layer_merge(core, red_layer, sass_layer(green)))


# File attachment example: Create a checkboard pattern .png, then
# use it from a sass layer

tmp_png <- tempfile(fileext = ".png")
grDevices::png(filename = tmp_png, width = 20, height = 20,
  bg = "transparent", antialias = "none")
par(mar = rep_len(0,4), xaxs = "i", yaxs = "i")
plot.new()
rect(c(0,0.5), c(0,0.5), c(0.5,1), c(0.5,1), col = "#00000044", border=NA)
dev.off()

layer <- sass_layer(
  rules = ".bg-check { background-image: url(images/demo_checkboard_bg.png) }",
  file_attachments = c("images/demo_checkboard_bg.png" = tmp_png)
```

```
)

output_path <- tempfile(fileext = ".css")
sass(layer, output = output_path, write_attachments = TRUE)
```

`sass_options` *Compiler Options for Sass*

Description

Set compiler options for Sass. Used with `sass`.

Usage

```
sass_options(
  precision = 5,
  output_style = "expanded",
  indented_syntax = FALSE,
  include_path = "",
  source_comments = FALSE,
  indent_type = "space",
  indent_width = 2,
  linefeed = "lf",
  output_path = "",
  source_map_file = "",
  source_map_root = "",
  source_map_embed = FALSE,
  source_map_contents = FALSE,
  omit_source_map_url = FALSE
)
```

Arguments

`precision`	Number of decimal places.
`output_style`	Bracketing and formatting style of the CSS output. Possible styles: `"nested"`, `"expanded"`, `"compact"`, and `"compressed"`.
`indented_syntax`	Enables the compiler to parse Sass Indented Syntax in strings. Note that the compiler automatically overrides this option to `TRUE` or `FALSE` for files with .sass and .scss file extensions respectively.
`include_path`	Vector of paths used to resolve `@import`. Multiple paths are possible using a character vector of paths.
`source_comments`	Annotates CSS output with line and file comments from Sass file for debugging.
`indent_type`	Specifies the indent type as `"space"` or `"tab"`.
`indent_width`	Number of tabs or spaces used for indentation. Maximum 10.

`linefeed` Specifies how new lines should be delimited. Possible values: `"lf"`, `"cr"`, `"lfcr"`, and `"crlf"`.

`output_path` Specifies the location of the output file. Note: this option will not write the file on disk. It is only for internal reference with the source map.

`source_map_file`
Specifies the location for Sass to write the source map.

`source_map_root`
Value will be included as source root in the source map information.

`source_map_embed`
Embeds the source map as a data URI.

`source_map_contents`
Includes the contents in the source map information.

`omit_source_map_url`
Disable the inclusion of source map information in the output file. Note: must specify `output_path` when `TRUE`.

Value

List of Sass compiler options to be used with `sass`.

Examples

```
sass(
  "foo { margin: 122px * .3; }",
  options = sass_options(output_style = "compact")
)
```

Index