Package 'sass'

January 24, 2021

Type Package

Version 0.3.1

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 7.1.1

SystemRequirements GNU make

Imports digest, fs, rlang, htmltools, R6, rappdirs

Suggests testthat, knitr, rmarkdown, withr, shiny

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 2021-01-24 17:10:02 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)
```

`output_template` *An intelligent (temporary) output file*

Description

Intended for use with `sass()`'s `output` argument for temporary file generation that is `cache` and `options` aware. In particular, this ensures that new redundant file(s) aren't generated on a `sass()` cache hit, and that the file's extension is suitable for the `sass_options()`'s `output_style`.

Usage

```
output_template(basename = "sass", dirname = basename, fileext = NULL)
```

Arguments

basename	a non-empty character vector giving the outfile name (without the extension).
dirname	a non-empty character vector giving the initial part of the directory name.
fileext	the output file extension. The default is ".min.css" for compressed and compact output styles; otherwise, its ".css".

Value

A function with two arguments: options and suffix. When called inside sass() with caching enabled, the caching key is supplied to suffix.

Examples

```
sass("body {color: red}", output = output_template())

func <- output_template(basename = "foo", dirname = "bar-")
func(suffix = "baz")
```

---

sass *Compile Sass to CSS*

Description

Compile Sass to CSS using LibSass.

Usage

```
sass(
  input = NULL,
  options = sass_options(),
  output = NULL,
  write_attachments = NA,
  cache = sass_cache_get(),
  cache_key_extra = NULL
)
```

Arguments

input	Accepts raw Sass, a named list of variables, a list of raw Sass and/or named variables, or a sass_layer() object. 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. May be a character string or output_template()

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.)

cache
: This can be a directory to use for the cache, a FileCache object created by sass_file_cache(), or FALSE or NULL for no caching.

cache_key_extra
: additional information to considering when computing the cache key. This should include any information that could possibly influence the resulting CSS that isn't already captured by input. For example, if input contains something like "@import sass_file.scss" you may want to include the file.mtime() of sass_file.scss (or, perhaps, a packageVersion() if sass_file.scss is bundled with an R package).

Value

If output = NULL, the function returns a string value of the compiled CSS. If output is specified, the compiled CSS is written to a file and the filename is returned.

Caching

By default, caching is enabled, meaning that sass() avoids the possibly expensive re-compilation of CSS whenever the same options and input are requested. Unfortunately, in some cases, options and input alone aren't enough to determine whether new CSS output must be generated. For example, changes in local file imports that aren't captured through sass_file()/sass_import(), may lead to a false-positive cache hit. For this reason, developers are encouraged to capture such information in cache_key_extra (possibly with packageVersion('myPackage') if shipping Sass with a package), and users may want to disable caching altogether during local development by calling options(sass.cache=FALSE).

In some cases when developing and modifying .scss files, sass() might not detect changes, and keep using cached .css files instead of rebuilding them. To be safe, if you are developing a theme with sass, it's best to turn off caching by calling options(sass.cache=FALSE).

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 computing a *cache key* from each sass() call's input, option, and cache_key_extra arguments. If an object with that hash already exists within the cache directory, its contents are used instead of performing the compilation. If it does not exist, then compilation is performed and usual and the results are stored in the cache.

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 can end up with stale results. To avoid this problem when developing sass code, it's best to disable caching with options(sass.cache=FALSE).

By default, the maximum size of the cache is 40 MB. If it grows past that size, the least-recently-used objects will be evicted from the cache to keep it under that size. Also by default, the maximum age of objects in the cache is one week. Older objects will be evicted from the cache.

To clear the default cache, call `sass_cache_get()$reset()`.

See Also

https://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; }"
))

# Compile a .scss file
example_file <- system.file("examples/example-full.scss", package = "sass")
sass(sass_file(example_file))

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

## Not run:
# ======================
# Caching 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);
}"

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

# But on subsequent calls, it should be very fast
```

```
system.time(sass(fib_sass))

# sass() can be called with cache=NULL; it will be slow
system.time(sass(fib_sass, cache = NULL))

# Clear the cache
sass_cache_get()$reset()

## End(Not run)

## Not run:
# Example of disabling cache by setting the default cache to NULL.

# Disable the default cache (save the original one first, so we can restore)
old_cache <- sass_cache_get()
sass_cache_set(NULL)
# Will be slow, because no cache
system.time(sass(fib_sass))

# Restore the original cache
sass_cache_set(old_cache)

## End(Not run)
```

`sass_cache_get` *Get default sass cache object for the current context*

Description

Get the default sass cache object, for the current context. The context depends on factors described below.

`sass_cache_get()` first checks the `sass.cache` option. If it is set to `NULL` or `FALSE`, then this function returns `NULL`. If it has been set to a string, it is treated as a directory name, and this function returns a `sass_file_cache()` object using that directory. If the option has been set to a `sass_file_cache()` object, then it will return that object.

In most cases, this function uses the user's cache directory, by calling `tools::R_user_dir("sass",which = "cache")` (for R 4.0 and above) or `rappdirs::user_cache_dir("R-sass")` (for older versions of R).

If this function is called from a Shiny application, it will also look for a subdirectory named app_cache/. If it exists, it will use a directory named app_cache/sass/ to store the cache.

When running a Shiny application in a typical R session, it will not create the app_cache/ subdirectory, but it will use it if present. This scopes the cache to the application.

With Shiny applications hosted on Shiny Server and Connect, it *will* create a app_cache/sass/ subdirectory, so that the cache is scoped to the application and will not interfere with another application's cache.

Usage

```
sass_cache_get()
```

Shiny Developer Mode

If Shiny Developer Mode is enabled (by setting `options(shiny.devmode = TRUE)` or calling `shiny::devmode(TRUE)`, the default global option value for `sass.cache` is updated to `FALSE` instead of `TRUE`, similar to `getOption("sass.cache",FALSE)`. This setting allows developers to make sure what is being returned from `sass()` is not an incorrect cache result.

See Also

`sass_cache_get_dir()`, `sass()`

---

sass_file_cache *Create a file cache object*

Description

This creates a file cache which is to be used by sass for caching generated .css files.

Usage

```
sass_file_cache(dir, max_size = 40 * 1024^2, max_age = Inf)
```

Arguments

`dir`	The directory in which to store the cached files.
`max_size`	The maximum size of the cache, in bytes. If the cache grows past this size, the least-recently-used objects will be removed until it fits within this size.
`max_age`	The maximum age of objects in the cache, in seconds. The default is one week.

Value

A FileCache object.

See Also

`sass_cache_get()`, `sass_cache_context_dir()`, FileCache

Examples

```
## Not run:
# Create a cache with the default settings
cache <- sass_file_cache(sass_cache_context_dir())

# Clear the cache
cache$reset()

## End(Not run)
```

`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 *Bundling Sass layers*

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_bundle()` to combine two or more layers or bundles objects into a Sass bundle; this ability to be merged is the main benefit of using Sass layers versus lower-level forms of sass input. At a later time, Sass layers may be removed from Sass bundles by referencing the same name that was used when creating the Sass bundle.

Usage

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

sass_bundle(...)

sass_bundle_remove(bundle, name)

is_sass_bundle(x)
```

Arguments

`defaults`	A suitable `as_sass()` `input`. Intended for declaring variables with `!default`. When layers are combined, defaults are merged in reverse order; that is, `sass_bundle(layer1,layer2)` will include `layer2$defaults` before `layer1$defaults`.
`declarations`	A suitable `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_bundle(layer1,layer2)` will include `layer1$declarations` before `layer2$declarations`.
`rules`	A suitable `as_sass()` `input`. Intended for actual CSS rules. These will be merged in forward order; that is, `sass_bundle(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`	Deprecated. Preserve meta information using a key in `sass_bundle(KEY = val)`. preserve simple metadata as layers are merged.
`...`	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 = ...)`.
`bundle`	Output value from `sass_layer()` or `sass_bundle()`
`name`	If a Sass layer name is contained in `name`, the matching Sass layer will be removed from the `bundle`
`x`	object to inspect

Functions

- `sass_layer`: Compose the parts of a single Sass layer. Object returned is a `sass_bundle()` with a single Sass layer
- `sass_bundle`: Collect `sass_bundle()` and/or `sass_layer()` objects. Unnamed Sass bundles will be concatenated together, preserving their internal name structures. Named Sass bundles will be condensed into a single Sass layer for easier removal from the returned Sass bundle.
- `sass_bundle_remove`: Remove a whole `sass_layer()` from a `sass_bundle()` object.
- `is_sass_bundle`: Check if `x` is a Sass bundle object

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_bundle(core, red_layer))
sass(sass_bundle(core, red_layer, sass_layer(green)))

# Example of merging layers and removing a layer
```

```
# Remember to name the layers that are removable
core_layers <- sass_bundle(core, red = red_layer, green = sass_layer(green))
core_layers # pretty printed for console
core_slim <- sass_bundle_remove(core_layers, "red")
sass(core_slim)


# 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")
)
```

sass_partial *Compile rules against a Sass Bundle or Sass Layer object*

Description

Replaces the rules for a `sass_layer()` object with new rules, and compile it. This is useful when (for example) you want to compile a set of rules using variables derived from a theme, but you do not want the resulting CSS for the entire theme – just the CSS for the specific rules passed in.

Usage

```
sass_partial(
  rules,
  bundle,
  options = sass_options(),
  output = NULL,
  write_attachments = NA,
  cache = sass_cache_get(),
  cache_key_extra = NULL
)
```

Arguments

`rules`	A set of sass rules, which will be used instead of the rules from `layer`.
`bundle`	A `sass_bundle()` or `sass_layer()` object.
`options`	Compiler options for Sass. Please specify options using `sass_options()`.
`output`	Specifies path to output file for compiled CSS. May be a character string or `output_template()`
`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.)
`cache`	This can be a directory to use for the cache, a FileCache object created by `sass_file_cache()`, or `FALSE` or `NULL` for no caching.
`cache_key_extra`	additional information to considering when computing the cache key. This should include any information that could possibly influence the resulting CSS that isn't already captured by `input`. For example, if `input` contains something like `"@import sass_file.scss"` you may want to include the `file.mtime()` of `sass_file.scss` (or, perhaps, a `packageVersion()` if `sass_file.scss` is bundled with an R package).

Examples

```
theme <- sass_layer(
  defaults = sass_file(system.file("examples/variables.scss", package = "sass")),
  rules = sass_file(system.file("examples/rules.scss", package = "sass"))
)

# Compile the theme
sass(theme)

# Sometimes we want to use the variables from the theme to compile other sass
my_rules <- ".someclass { background-color: $bg; color: $fg; }"
sass_partial(my_rules, theme)
```

Index