# Package: sass (via r-universe)

November 2, 2026

**Type** Package

**Title** Syntactically Awesome Style Sheets ('Sass')

**Version** 0.4.10.9000

**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://rstudio.github.io/sass/, https://github.com/rstudio/sass

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

**Imports** fs (>= 1.2.4), htmltools (>= 0.5.1), R6, rappdirs, rlang (>= 0.4.10)

**Suggests** curl, knitr, rmarkdown, shiny, testthat, withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**SystemRequirements** GNU make

**Config/roxygen2/version** 8.1.1

**Config/pak/sysreqs** cmake make libuv1-dev

**Repository** https://posit-dev-shinycoreci.r-universe.dev

**Date/Publication** 2026-11-02 16:53:02 UTC

**RemoteUrl** https://github.com/rstudio/sass

**RemoteRef** HEAD

**RemoteSha** 9ca326fbbaceeb9e607ece4806cf589b76ddade9

# Contents

as_sass . . . 2
font_google . . . 3
output_template . . . 7
sass . . . 8
sass_cache_get . . . 11
sass_file_cache . . . 11
sass_import . . . 12
sass_layer . . . 13
sass_options . . . 15
sass_partial . . . 17

**Index** **19**

---

`as_sass` *Convert an R object into Sass code*

---

## Description

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

## Usage

```
as_sass(input)
```

## Arguments

| | |
|---|---|
| `input` | Any of the following:<br>• A character vector containing Sass code.<br>• A named list containing variable names and values.<br>• A `sass_file()`, `sass_layer()`, and/or `sass_bundle()`.<br>• A `list()` containing any of the above. |

## Value

a single character value to be supplied to `sass()`.

## References

https://sass-lang.com/documentation/at-rules/import

## 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)
```

---

font_google *Helpers for importing web fonts*

---

## Description

Include font file(s) when defining a Sass variable that represents a CSS `font-family` property.

## Usage

```
font_google(
  family,
  local = TRUE,
  cache = sass_file_cache(sass_cache_context_dir()),
  wght = NULL,
  ital = NULL,
```

```
  display = c("swap", "auto", "block", "fallback", "optional")
)

font_link(family, href)

font_face(
  family,
  src,
  weight = NULL,
  style = NULL,
  display = c("swap", "auto", "block", "fallback", "optional"),
  stretch = NULL,
  variant = NULL,
  unicode_range = NULL
)

font_collection(..., default_flag = TRUE, quote = TRUE)

is_font_collection(x)
```

## Arguments

| | |
|---|---|
| family | A character string with a *single* font family name. |
| local | Whether or not download and bundle local (woff2) font files. |
| cache | A sass_file_cache() object (or, more generally, a file caching class with $get_file() and $set_file() methods). Set this argument to FALSE or NULL to disable caching. |
| wght | One of the following:<br>• NULL, the default weight for the family.<br>• A character string defining an axis range<br>• A numeric vector of desired font weight(s). |
| ital | One of the following:<br>• NULL, the default font-style for the family.<br>• 0, meaning font-style: normal<br>• 1, meaning font-style: italic<br>• c(0, 1), meaning both normal and italic |
| display | A character vector for the font-display @font-face property. |
| href | A URL resource pointing to the font data. |
| src | A character vector for the src @font-face property. Beware that is character strings are taken verbatim, so careful quoting and/or URL encoding may be required. |
| weight | A character (or numeric) vector for the font-weight @font-face property. |
| style | A character vector for the font-style @font-face property. |
| stretch | A character vector for the font-stretch @font-face property. |

| | |
|---|---|
| variant | A character vector for the font-variant @font-face property. |
| unicode_range | A character vector for unicode-range @font-face property. |
| ... | a collection of font_google(), font_link(), font_face(), and/or character vector(s) (i.e., family names to include in the CSS font-family properly). Family names are automatically quoted as necessary. |
| default_flag | whether or not to include a !default when converted to a Sass variable with as_sass(). |
| quote | whether or not to attempt automatic quoting of family names. |
| x | test whether x is a font_collection(), font_google(), font_link(), or font_face() object. |

## Details

These helpers **must be used the named list approach to variable definitions**, for example:

```
list(
  list("font-variable" = font_google("Pacifico")),
  list("body{font-family: $font-variable}")
)
```

## Value

a sass_layer() holding an htmltools::htmlDependency() which points to the font files.

## Font fallbacks

By default, font_google() downloads, caches, and serves the relevant font file(s) locally. By locally serving files, there's a guarantee that the font can render in any client browser, even when the client doesn't have internet access. However, when importing font files remotely (i.e., font_google(..., local = FALSE) or font_link()), it's a good idea to provide fallback font(s) in case the remote link isn't working (e.g., maybe the end user doesn't have an internet connection). To provide fallback fonts, use font_collection(), for example:

```
pacifico <- font_google("Pacifico", local = FALSE)
as_sass(list(
  list("font-variable" = font_collection(pacifico, "system-ui")),
  list("body{font-family: $font-variable}")
))
```

## Default flags

These font helpers encourage best practice of adding a !default to Sass variable definitions, but the flag may be removed via font_collection() if desired.

```
as_sass(list("font-variable" = pacifico))
#> $font-variable: Pacifico !default;
as_sass(list("font-variable" = font_collection(pacifico, default_flag = F)))
#> $font-variable: Pacifico;
```

### Serving non-Google fonts locally

Non-Google fonts may also be served locally with `font_face()`, but it requires downloading font file(s) and pointing `src` to the right location on disk. If you want `src` to be a relative file path (you almost certainly do), then you'll need to mount that resource path using something like `shiny::addResourcePath()` (for a shiny app) or `servr::httd()` (for static HTML).

## References

https://developers.google.com/fonts/docs/css2

https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face

https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Web_fonts

## Examples

```
library(htmltools)

my_font <- list("my-font" = font_google("Pacifico"))
hello <- tags$body(
  "Hello",
  tags$style(
    sass(
      list(
        my_font,
        list("body {font-family: $my-font}")
      )
    )
  )
)

if (interactive()) {
  browsable(hello)
}

# Three different yet equivalent ways of importing a remotely-hosted Google Font
a <- font_google("Crimson Pro", wght = "200..900", local = FALSE)
b <- font_link(
  "Crimson Pro",
  href = "https://fonts.googleapis.com/css2?family=Crimson+Pro:wght@200..900"
)
url <- "https://fonts.gstatic.com/s/crimsonpro/v13/q5uDsoa5M_tv7IihmnkabARboYF6CsKj.woff2"
c <- font_face(
  family = "Crimson Pro",
  style = "normal",
  weight = "200 900",
  src = paste0("url(", url, ") format('woff2')")
)
```

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,
  path = tempdir()
)
```

## Arguments

| | |
|---|---|
| basename | a non-empty character string giving the outfile name (without the extension). |
| dirname | a non-empty character string 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". |
| path | the output file's root directory path. |

## 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_get(),
  output = NULL,
  write_attachments = NA,
  cache = sass_cache_get(),
  cache_key_extra = NULL
)
```

## Arguments

| | |
|---|---|
| input | Any of the following:<br>• A character vector containing Sass code.<br>• A named list containing variable names and values.<br>• A sass_file(), sass_layer(), and/or sass_bundle().<br>• A list() containing any of the above. |
| options | Compiler 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` *Retrieve the default file cache*

---

## Description

When caching is enabled, this function returns a `sass_file_cache()` object that `sass()`'s cache argument uses (by default) for caching Sass compilation. When caching is disabled (either by setting the `sass.cache` option to `FALSE`, `NULL`, or via `shiny::devmode()`), this function returns `NULL` (effectively telling `sass()` to not cache by default).

## Usage

```
sass_cache_get()
```

## Details

When caching is enabled, then this function returns a `sass_file_cache()` object that (by default) uses `sass_cache_context_dir()` for its directory. The directory can also be customized by providing the `sass.cache` option with either a filepath (as a string) or a full-blown `sass_file_cache()` object.

## 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/at-rules/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.

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

## 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 provide a way to package Sass variables, rules, functions, and mixins in a structured and composable way that follows best Sass practices. Most importantly, when multiple `sass_layer()` are combined into a `sass_bundle()`, variable `defaults` for later layers are placed *before* earlier layers, effectively 'new' defaults through all the 'old' defaults.

## Usage

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

sass_layer_file(file)

sass_bundle(...)

sass_bundle_remove(bundle, name)

is_sass_bundle(x)
```

## Arguments

| | |
|---|---|
| `functions` | `as_sass()` input intended for Sass functions. Functions are placed before `defaults` so that variable definitions may make use of functions. |

| | |
|---|---|
| defaults | as_sass() input intended for variable defaults. These variable defaults after placed after functions but before mixins. When multiple layers are combined in a sass_bundle(), defaults are merged in reverse order; that is, sass_bundle(layer1, layer2) will include layer2$defaults before layer1$defaults. |
| mixins | as_sass() input intended for Sass mixins. Mixins are placed after defaults, but before rules. |
| rules | as_sass() input intended for Sass rules. Rules are placed last (i.e., after functions, defaults, and mixins). |
| html_deps | An HTML dependency (or a list of them). This dependency gets attached to the return value of sass()/as_sass(). |
| 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. |
| declarations | Deprecated, use functions or mixins instead. |
| tags | Deprecated. Preserve meta information using a key in sass_bundle(KEY = val). preserve simple metadata as layers are merged. |
| file | file path to a .scss file. |
| ... | 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_layer_file(): Read in a .scss file with parse special /*-- scss:(functions|defaults|rules|mixins) --* comments as relevant sections of a 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 (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

Specify compiler `options` for `sass()`. To customize options, either provide `sass_options()` directly to a `sass()` call or set options globally via `sass_options_set()`. When `shiny::devmode()` is enabled, `sass_options_get()` defaults `source_map_embed` and `source_map_contents` to `TRUE`.

## 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
)

sass_options_get(...)

sass_options_set(...)
```

## 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`.

...
: arguments to `sass_options()`. For `sass_options_set()`, the following values are also acceptable:
  - `NULL`, clearing the global options.
  - Return value of `sass_options_get()`.
  - Return value of `sass_options_set()`.

## Value

List of Sass compiler options to be used with `sass()`. For `sass_options_set()`, any previously set global options are returned.

## Examples

```
x <- "foo { margin: 122px * .001; }"
sass(x)

# Provide options directly to sass()
sass(x, options = sass_options(precision = 1, output_style = "compact"))

# Or set some option(s) globally
old_options <- sass_options_set(precision = 1)
sass(x)

# Specify local options  while also respecting global options
sass(x, options = sass_options_get(output_style = "compact"))

# Restore original state
sass_options_set(old_options)
```

---

# 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_get(),
  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 `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


as_sass, 2
as_sass(), *5*, *13, 14*

file.mtime(), *8*, *18*
FileCache, *8*, *12*, *18*
font_collection *(*font_google*)*, 3
font_collection(), *5*
font_face *(*font_google*)*, 3
font_google, 3
font_link *(*font_google*)*, 3

htmltools::htmlDependency(), *5*

is_font_collection *(*font_google*)*, 3
is_sass_bundle *(*sass_layer*)*, 13

list(), *2*, *8*

output_template, 7
output_template(), *8*, *18*

packageVersion(), *8*, *18*

sass, 8
sass(), *2*, *7*, *9*, *11*, *14*, *16, 17*
sass_bundle *(*sass_layer*)*, 13
sass_bundle(), *2*, *8*, *18*
sass_bundle_remove *(*sass_layer*)*, 13
sass_cache_context_dir(), *11, 12*
sass_cache_get, 11
sass_cache_get(), *12*
sass_cache_get_dir(), *11*
sass_file *(*sass_import*)*, 12
sass_file(), *2*, *8, 9*
sass_file_cache, 11
sass_file_cache(), *4*, *8*, *11*, *18*
sass_import, 12
sass_import(), *9*
sass_layer, 13
sass_layer(), *2*, *5*, *8*, *17, 18*
sass_layer_file *(*sass_layer*)*, 13
sass_options, 15
sass_options(), *7, 8*, *17, 18*
sass_options_get *(*sass_options*)*, 15
sass_options_set *(*sass_options*)*, 15
sass_partial, 17
shiny::addResourcePath(), *6*
shiny::devmode(), *11*