# Package 'bslib'

January 25, 2021

**Title** Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'

**Version** 0.2.4

**Description** Simplifies custom 'CSS' styling of both 'shiny' and 'rmarkdown' via 'Bootstrap' 'Sass'. Supports both 'Bootstrap' 3 and 4 as well as their various 'Bootswatch' themes. An interactive widget is also provided for previewing themes in real time.

**Depends** R (>= 2.10)

**Imports** grDevices, htmltools (>= 0.5.1), jsonlite, sass (>= 0.3.0), digest (>= 0.6.25), jquerylib (>= 0.1.3), rlang, magrittr

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

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Collate** 'bootswatch.R' 'bs-current-theme.R' 'bs-dependencies.R' 'bs-global.R' 'bs-remove.R' 'bs-theme-layers.R' 'utils.R' 'bs-theme-preview.R' 'bs-theme-update.R' 'bs-theme.R' 'deprecated.R' 'files.R' 'fonts.R' 'imports.R' 'onLoad.R' 'precompiled.R' 'shiny-devmode.R' 'version-default.R' 'versions.R' 'web-font.R'

**URL** https://rstudio.github.io/bslib/, https://github.com/rstudio/bslib

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

**NeedsCompilation** no

**Author** Carson Sievert [aut, cre],
Joe Cheng [aut],
RStudio [cph],
Bootstrap contributors [ctb] (Bootstrap library),
Twitter, Inc [cph] (Bootstrap library),
Javi Aguilar [ctb, cph] (Bootstrap colorpicker library),
Thomas Park [ctb, cph] (Bootswatch library),
PayPal [ctb, cph] (Bootstrap accessibility plugin)

**Maintainer** Carson Sievert <carson@rstudio.com>

**Repository** CRAN

**Date/Publication** 2021-01-25 08:00:02 UTC

# R topics documented:



---

`bootswatch_themes` *Obtain a list of all available bootswatch themes.*

---

## Description

Obtain a list of all available bootswatch themes.

## Usage

```
bootswatch_themes(version = version_default(), full_path = FALSE)
```

## Arguments

| | |
|---|---|
| `version` | the major version of Bootswatch. |
| `full_path` | whether to return a path to the installed theme. |

## Value

a character vector of Bootswatch themes.

bs_add_variables *Add low-level theming customizations*

## Description

Compared to higher-level theme customization available in `bs_theme()`, these functions are a more direct interface to Bootstrap Sass, and therefore, do nothing to ensure theme customizations are portable between major Bootstrap versions.

## Usage

```
bs_add_variables(
  theme,
  ...,
  .where = "defaults",
  .default_flag = identical(.where, "defaults")
)

bs_add_rules(theme, rules)

bs_add_declarations(theme, declarations)

bs_bundle(theme, ...)
```

## Arguments

| | |
|---|---|
| `theme` | a `bs_theme()` object. |
| `...` | • `bs_add_variables()`: Should be named Sass variables or values that can be passed in directly to the `defaults` argument of a `sass::sass_layer()`.<br>• `bs_bundle()`: Should be arguments that can be handled by `sass::sass_bundle()` to be appended to the `theme` |
| `.where` | Whether to place the variable definitions before other Sass `"defaults"`, after other Sass `"declarations"`, or after other Sass `"rules"`. |
| `.default_flag` | Whether or not to add a `!default` flag (if missing) to variable expressions. It's recommended to keep this as `TRUE` when `.where = "defaults"`. |
| `rules` | Sass rules. Anything understood by `sass::as_sass()` may be provided (e.g., a list, character vector, `sass::sass_file()`, etc) |
| `declarations` | Sass functions and mixins. |

## Value

a modified `bs_theme()` object.

## Functions

- bs_add_variables: Add Bootstrap Sass variable defaults
- bs_add_rules: Add additional Sass rules
- bs_add_declarations: Add Sass functions and mixins
- bs_bundle: Add additional sass::sass_bundle() objects to an existing theme.

## References

https://getbootstrap.com/docs/4.4/getting-started/theming/

https://rstudio.github.io/sass/articles/sass.html#layering

## Examples

```
# Function to preview the styling a (primary) Bootstrap button
library(htmltools)
button <- tags$a(class = "btn btn-primary", href = "#", role = "button", "Hello")
preview_button <- function(theme) {
  if (interactive()) {
    browsable(tags$body(bs_theme_dependencies(theme), button))
  }
}

# Here we start with a theme based on a Bootswatch theme,
# then override some variable defaults
theme <- bs_theme(bootswatch = "sketchy", primary = "orange") %>%
  bs_add_variables(
    "body-bg" = "#EEEEEE",
    "font-family-base" = "monospace",
    "font-size-base" = "1.4rem",
    "btn-padding-y" = ".16rem",
    "btn-padding-x" = "2rem"
  )

preview_button(theme)

# If you need to set a variable based on another Bootstrap variable
theme %>%
  bs_add_variables("body-color" = "$success", .where = "declarations") %>%
  preview_button()

# Start a new global theme and add some custom rules that
# use Bootstrap variables to define a custom styling for a
# 'person card'
person_rules <- system.file("custom", "person.scss", package = "bslib")
theme <- bs_theme() %>% bs_add_rules(sass::sass_file(person_rules))
# Include custom CSS that leverages bootstrap Sass variables
person <- function(name, title, company) {
  tags$div(
    class = "person",
```

```
    h3(class = "name", name),
    div(class = "title", title),
    div(class = "company", company)
  )
}
if (interactive()) {
  browsable(shiny::fluidPage(
    theme = theme,
    person("Andrew Carnegie", "Owner", "Carnegie Steel Company"),
    person("John D. Rockefeller", "Chairman", "Standard Oil")
  ))
}
```

---

`bs_current_theme` *Obtain the currently active theme at render time*

---

## Description

Intended for advanced use by developers to obtain the currently active theme *at render time* and primarily for implementing themable widgets that can't otherwise be themed via `bs_dependency_defer()`.

## Usage

```
bs_current_theme(session = get_reactive_domain())
```

## Arguments

| | |
|---|---|
| `session` | The current Shiny session (if any). |

## Details

This function should generally only be called at print/render time. For example:

- Inside the `preRenderHook` of `htmlwidgets::createWidget()`.
- Inside of a custom print method that generates htmltools::tags.
- Inside of a `htmltools::tagFunction()`

Calling this function at print/render time is important because it does different things based on the context in which it's called:

- If a reactive context is active, `session$getCurrentTheme()` is called (which is a reactive read).
- If no reactive context is active, `shiny::getCurrentTheme()` is called (which returns the current app's `theme`, if relevant).
- If `shiny::getCurrentTheme()` comes up empty, then `bs_global_get()` is called, which is relevant for `rmarkdown::html_document()`, and possibly other static rendering contexts.

## Value

a bs_theme() object.

---

bs_dependency *Themeable HTML components*

---

## Description

Themeable HTML components use Sass to generate CSS rules from Bootstrap Sass variables, functions, and/or mixins (i.e., stuff inside of theme). bs_dependencies() makes it a bit easier to create themeable components by compiling sass::sass() (input) together with Bootstrap Sass inside of a theme, and packaging up the result into an htmlDependency().

Themable components can also be *dynamically* themed inside of Shiny (i.e., they may be themed in 'real-time' via bs_themer(), and more generally, update their styles in response to shiny::session's setCurrentTheme() method). Dynamically themeable components provide a "recipe" (i.e., a function) to bs_dependency_defer(), describing how to generate new CSS stylesheet(s) from a new theme. This function is called when the HTML page is first rendered, and may be invoked again with a new theme whenever shiny::session's setCurrentTheme() is called.

## Usage

```
bs_dependency(
  input = list(),
  theme,
  name,
  version,
  cache_key_extra = NULL,
  .dep_args = list(),
  .sass_args = list()
)

bs_dependency_defer(func)
```

## Arguments

| | |
|---|---|
| input | Sass rules to compile, using theme. |
| theme | A bs_theme() object. |
| name | Library name |
| version | Library version |
| cache_key_extra | Extra information to add to the sass cache key. It is useful to add the version of your package. |
| .dep_args | A list of additional arguments to pass to htmltools::htmlDependency(). Note that package has no effect and script must be absolute path(s). |
| .sass_args | A list of additional arguments to pass to sass::sass_partial(). |

func — a *non-anonymous* function, with a *single* argument. This function should accept a `bs_theme()` object and return a single `htmlDependency()`, a list of them, or `NULL`.

## Value

`bs_dependency()` returns an `htmltools::htmlDependency()` and `bs_dependency_defer()` returns an `htmltools::tagFunction()`

## References

https://rstudio.github.io/bslib/articles/theming.html#themable-components-1

## Examples

```
## Not run:

myWidgetVersion <- "1.2.3"

myWidgetDependency <- function() {
  list(
    bs_dependency_defer(myWidgetCss),
    htmlDependency(
      name = "mywidget-js",
      version = myWidgetVersion,
      src = system.file(package = "mypackage", "js"),
      script = "mywidget.js"
    )
  )
}

myWidgetCSS <- function(theme) {
  if (!is_bs_theme(theme)) {
    return(
      htmlDependency(
        name = "mywidget-css",
        version = myWidgetVersion,
        src = system.file(package = "mypackage", "css"),
        stylesheet = "mywidget.css"
      )
    )
  }

  # Compile mywidget.scss using the variables and defaults from the theme
  # object.
  sass_input <- sass::sass_file(system.file(package = "mypackage", "scss/mywidget.scss"))

  bs_dependency(
    input = sass_input,
    theme = theme,
    name = "mywidget",
```

```
    version = myWidgetVersion,
    cache_key_extra = utils::packageVersion("mypackage")
  )
}

# Note that myWidgetDependency is not defined inside of myWidget. This is so
# that, if `myWidget()` is called multiple times, Shiny can tell that the
# function objects are identical and deduplicate them.
myWidget <- function(id) {
  div(
    id = id,
    span("myWidget"),
    myWidgetDependency()
  )
}

## End(Not run)
```

---

`bs_get_variables` *Retrieve Sass variable values from the current theme*

---

## Description

Useful for retriving a variable from the current theme and using the value to inform another R function.

## Usage

```
bs_get_variables(theme, varnames)

bs_get_contrast(theme, varnames)
```

## Arguments

| | |
|---|---|
| `theme` | a `bs_theme()` object. |
| `varnames` | a character string referencing a Sass variable in the current theme. |

## Value

a character string containing a CSS/Sass value. If the variable(s) are not defined, their value is `NA`.

## Examples

```
vars <- c("body-bg", "body-color", "primary", "border-radius")
bs_get_variables(bs_theme(), varnames = vars)
bs_get_variables(bs_theme(bootswatch = "darkly"), varnames = vars)
```

```
bs_get_contrast(bs_theme(), c("primary", "dark", "light"))

library(htmltools)
div(
  class = "bg-primary",
  style = css(
    color = bs_get_contrast(bs_theme(), "primary")
  )
)
```

---

bs_global_theme *Global theming*

---

## Description

bs_global_theme() creates a new (global) Bootstrap Sass theme which bs_theme_dependencies() (or sass_partial()) can consume (their theme argument defaults to bs_global_get(), which get the current global theme).

## Usage

```
bs_global_theme(
  version = version_default(),
  bootswatch = NULL,
  bg = NULL,
  fg = NULL,
  primary = NULL,
  secondary = NULL,
  success = NULL,
  info = NULL,
  warning = NULL,
  danger = NULL,
  base_font = NULL,
  code_font = NULL,
  heading_font = NULL,
  ...
)

bs_global_set(theme = bs_theme())

bs_global_get()

bs_global_clear()

bs_global_add_variables(
  ...,
  .where = "defaults",
```

```
  .default_flag = identical(.where, "defaults")
)

bs_global_add_rules(...)

bs_global_bundle(...)

bs_global_theme_update(
  ...,
  bootswatch = NULL,
  bg = NULL,
  fg = NULL,
  primary = NULL,
  secondary = NULL,
  success = NULL,
  info = NULL,
  warning = NULL,
  danger = NULL,
  base_font = NULL,
  code_font = NULL,
  heading_font = NULL
)
```

## Arguments

| | |
|---|---|
| version | The major version of Bootstrap to use (see versions() for possible values). |
| bootswatch | The name of a bootswatch theme (see bootswatch_themes() for possible values). When provided to bs_theme_update(), any previous Bootswatch theme is first removed before the new one is applied (use bootswatch = "default" to effectively remove the Bootswatch theme). |
| bg | A color string for the background. |
| fg | A color string for the foreground. |
| primary | A color to be used for hyperlinks, to indicate primary/default actions, and to show active selection state in some Bootstrap components. Generally a bold, saturated color that contrasts with the theme's base colors. |
| secondary | A color for components and messages that don't need to stand out. (Not supported in Bootstrap 3.) |
| success | A color for messages that indicate an operation has succeeded. Typically green. |
| info | A color for messages that are informative but not critical. Typically a shade of blue-green. |
| warning | A color for warning messages. Typically yellow. |
| danger | A color for errors. Typically red. |
| base_font | The default typeface. |
| code_font | The typeface to be used for code. Be sure this is monospace! |
| heading_font | The typeface to be used for heading elements. |

| | |
|---|---|
| ... | arguments passed along to bs_add_variables(). |
| theme | a bs_theme() object. |
| .where | Whether to place the variable definitions before other Sass "defaults", after other Sass "declarations", or after other Sass "rules". |
| .default_flag | Whether or not to add a !default flag (if missing) to variable expressions. It's recommended to keep this as TRUE when .where = "defaults". |

## Value

functions that modify the global theme (e.g., bs_global_set()) invisibly return the previously set theme. bs_global_get() returns the current global theme.

## See Also

bs_theme(), bs_theme_preview()

## Examples

```
# Remember the global state now (so we can restore later)
theme <- bs_global_get()
# Use Bootstrap 3 (globally) with some theme customization
bs_global_theme(3, bg = "#444", fg = "#e4e4e4", primary = "#e39777")
if (interactive()) bs_theme_preview(with_themer = FALSE)
# If no global theme is active, bs_global_get() returns NULL
bs_global_clear()
bs_global_get()
# Restore the original state
bs_global_set(theme)
```

---

# bs_remove *Remove or retrieve Sass code from a theme*

---

## Description

Remove or retrieve Sass code from a theme

## Usage

```
bs_remove(theme, ids = character(0))

bs_retrieve(theme, ids = character(0), include_unnamed = TRUE)
```

### Arguments

| | |
|---|---|
| theme | a bs_theme() object. |
| ids | a character vector of ids |
| include_unnamed | whether or not to include unnamed sass::sass_layer()s (e.g., Bootstrap Sass variables, functions, and mixins). |

### Value

a modified bs_theme() object.

### Examples

```
# Remove CSS rules for print and carousels
bs_theme(version = 4) %>%
  bs_remove(c("_print", "_carousel"))

# Remove BS3 compatibility layer
bs_theme(version = 4) %>%
  bs_remove("bs3compat")
```

---

## bs_theme *Create a Bootstrap theme*

---

### Description

Creates a Bootstrap theme object which can be:

- Used in any HTML page powered by shiny::bootstrapLib() (e.g., shiny::fluidPage(), shiny::bootstrapPage(), etc).
- Used in any output format powered by rmarkdown::html_document() (or rmarkdown::html_document_base()).
- Used more generally in any htmltools::tags via bs_theme_dependencies().

These functions (i.e., bs_theme() or bs_theme_update()) allow you to do the following common Bootstrap customization(s):

- Choose a (major) Bootstrap version.
- Choose a Bootswatch theme (optional).
- Customize main colors and fonts via explicitly named arguments (e.g., bg, fg, primary, etc).
- Customize other, lower-level, Bootstrap Sass variable defaults via . . .
  - See all Bootstrap 4 variables
  - See all Bootstrap 3 variables

For less common theming customization(s), you can modify theme objects to:

- Add additional Sass/CSS rules (see `bs_add_rules()` and `sass_partial()`).
- Leverage (new) Sass functions and mixins in those rules (see `bs_add_declarations()`)

These lower-level theming tools build on the concept of a `sass::sass_layer()`. To learn more, see here.

## Usage

```
bs_theme(
  version = version_default(),
  bootswatch = NULL,
  ...,
  bg = NULL,
  fg = NULL,
  primary = NULL,
  secondary = NULL,
  success = NULL,
  info = NULL,
  warning = NULL,
  danger = NULL,
  base_font = NULL,
  code_font = NULL,
  heading_font = NULL
)

bs_theme_update(
  theme,
  ...,
  bootswatch = NULL,
  bg = NULL,
  fg = NULL,
  primary = NULL,
  secondary = NULL,
  success = NULL,
  info = NULL,
  warning = NULL,
  danger = NULL,
  base_font = NULL,
  code_font = NULL,
  heading_font = NULL
)

is_bs_theme(x)
```

## Arguments

| | |
|---|---|
| `version` | The major version of Bootstrap to use (see `versions()` for possible values). |
| `bootswatch` | The name of a bootswatch theme (see `bootswatch_themes()` for possible values). When provided to `bs_theme_update()`, any previous Bootswatch theme |

| | |
|---|---|
| | is first removed before the new one is applied (use `bootswatch = "default"` to effectively remove the Bootswatch theme). |
| `...` | arguments passed along to `bs_add_variables()`. |
| `bg` | A color string for the background. |
| `fg` | A color string for the foreground. |
| `primary` | A color to be used for hyperlinks, to indicate primary/default actions, and to show active selection state in some Bootstrap components. Generally a bold, saturated color that contrasts with the theme's base colors. |
| `secondary` | A color for components and messages that don't need to stand out. (Not supported in Bootstrap 3.) |
| `success` | A color for messages that indicate an operation has succeeded. Typically green. |
| `info` | A color for messages that are informative but not critical. Typically a shade of blue-green. |
| `warning` | A color for warning messages. Typically yellow. |
| `danger` | A color for errors. Typically red. |
| `base_font` | The default typeface. |
| `code_font` | The typeface to be used for code. Be sure this is monospace! |
| `heading_font` | The typeface to be used for heading elements. |
| `theme` | a `bs_theme()` object. |
| `x` | an object. |

## Value

a `sass::sass_bundle()` (list-like) object.

## Colors

Colors may be provided in any format that `htmltools::parseCssColors()` can understand. To control the vast majority of the ('grayscale') color defaults, specify both the `fg` (foreground) and `bg` (background) colors. The `primary` and `secondary` theme colors are also useful for accenting the main grayscale colors in things like hyperlinks, tabset panels, and buttons.

## Fonts

Use `base_font`, `code_font`, and `heading_font` to control the main typefaces. These arguments set new defaults for the relevant `font-family` CSS properties, but don't necessarily import the relevant font files. To both set CSS properties *and* import font files, consider using the various `font_face()` helpers.

Each *_font argument may be collection of character vectors, `font_google()`s, `font_link()`s and/or `font_face()`s. Note that a character vector can have:

- A single unquoted name (e.g., `"Source Sans Pro"`).
- A single quoted name (e.g., `"'Source Sans Pro'"`).
- A comma-separated list of names w/ individual names quoted as necessary. (e.g. `c("Open Sans","'Source Sans Pro'","'Helvetica Neue',Helvetica,sans-serif")`)

Since `font_google(...,local = TRUE)` guarantees that the client has access to the font family, meaning it's relatively safe to specify just one font family, for instance:

```
bs_theme(base_font = font_google("Pacifico", local = TRUE))
```

However, specifying multiple "fallback" font families is recommended, especially when relying on remote and/or system fonts being available, for instance. Fallback fonts are useful not only for handling missing fonts, but also for handling a Flash of Invisible Text (FOIT) which can be quite noticeable with remote web fonts on a slow internet connection.

```
bs_theme(base_font = list(font_google("Pacifico", local = FALSE), "Roboto", "sans-serif")
```

## References

https://getbootstrap.com/docs/4.4/getting-started/theming/

https://rstudio.github.io/sass/

## See Also

bs_add_variables(), bs_theme_preview()

## Examples

```
theme <- bs_theme(
  # Controls the default grayscale palette
  bg = "#202123", fg = "#B8BCC2",
  # Controls the accent (e.g., hyperlink, button, etc) colors
  primary = "#EA80FC", secondary = "#48DAC6",
  base_font = c("Grandstander", "sans-serif"),
  code_font = c("Courier", "monospace"),
  heading_font = "'Helvetica Neue', Helvetica, sans-serif",
  # Can also add lower-level customization
  "input-border-color" = "#EA80FC"
)
if (interactive()) {
  bs_theme_preview(theme)
}

# Lower-level bs_add_*() functions allow you to work more
# directly with the underlying Sass code
theme <- theme %>%
  bs_add_variables("my-class-color" = "red") %>%
  bs_add_rules(".my-class { color: $my-class-color }")
```

bs_theme_dependencies *Compile Bootstrap Sass with (optional) theming*

## Description

bs_theme_dependencies() compiles Bootstrap Sass into CSS and returns it, along with other HTML dependencies, as a list of htmltools::htmlDependency()s. Most users won't need to call this function directly as Shiny & R Markdown will perform this compilation automatically when handed a bs_theme(). If you're here looking to create a themeable component, see bs_dependency().

## Usage

```
bs_theme_dependencies(
  theme,
  sass_options = sass::sass_options(output_style = "compressed"),
  cache = sass::sass_cache_get(),
  jquery = jquerylib::jquery_core(3),
  precompiled = get_precompiled_option("bslib.precompiled", default = TRUE)
)
```

## Arguments

| | |
|---|---|
| theme | a bs_theme() object. |
| sass_options | a sass::sass_options() object. |
| 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. |
| jquery | a jquerylib::jquery_core() object. |
| precompiled | Before compiling the theme object, first look for a precompiled CSS file for the theme_version(). If precompiled = TRUE and a precompiled CSS file exists for the theme object, it will be fetched immediately and not compiled. At the moment, we only provide precompiled CSS for "stock" builds of Bootstrap (i.e., no theming additions, bootswatch themes, or non-default sass_options). |

## Value

a list of HTML dependencies containing Bootstrap CSS, Bootstrap JavaScript, and jquery. This list may contain additional HTML dependencies if bundled with the theme.

## Sass caching and precompilation

If Shiny Developer Mode is enabled (by setting options(shiny.devmode = TRUE) or calling shiny::devmode(TRUE)), both **sass** caching and **bslib** precompilation are disabled by default; that is, a call to bs_theme_dependencies(theme) expands to bs_theme_dependencies(theme,cache = F,precompiled = F)). This is useful for local development as enabling caching/precompilation may produce incorrect results if local changes are made to bslib's source files.

## See Also

bs_theme(), bs_dependency()

## Examples

```
# Function to preview the styling a (primary) Bootstrap button
library(htmltools)
button <- tags$a(class = "btn btn-primary", href = "#", role = "button", "Hello")
preview_button <- function(theme) {
  if (interactive()) {
    browsable(tags$body(bs_theme_dependencies(theme), button))
  }
}

# Latest Bootstrap
preview_button(bs_theme())
# Bootstrap 3
preview_button(bs_theme(3))
# Bootswatch 4 minty theme
preview_button(bs_theme(4, bootswatch = "minty"))
# Bootswatch 4 sketchy theme
preview_button(bs_theme(4, bootswatch = "sketchy"))
```

---

# bs_theme_preview *Preview the currently set theme*

---

## Description

Launches an example shiny app via `run_with_themer()` and `bs_theme_dependencies()`. Useful for getting a quick preview of the current theme setting as well as an interactive GUI for tweaking some of the main theme settings.

## Usage

```
bs_theme_preview(theme = bs_theme(), ..., with_themer = TRUE)
```

## Arguments

| | |
|---|---|
| theme | a bs_theme() object. |
| ... | passed along to shiny::runApp(). |
| with_themer | whether or not to run the app with run_with_themer(). |

## Details

The app that this launches is subject to change.

## Value

nothing, this function is called for its side-effects (launching an application).

## See Also

`run_with_themer()`

## Examples

```
theme <- bs_theme(bg = "#6c757d", fg = "white", primary = "orange")
if (interactive()) bs_theme_preview(theme)
```

---

# font_face *Import web fonts into a* `bs_theme()`

---

## Description

When used with any of the main font settings in `bs_theme()` (e.g., `base_font`, `code_font`, `heading_font`), these font objects ensure relevant font file resources are included with the theme. A particular font object should define an single font `family` — if you need multiple families, a `list()` of font objects may be provided to `bs_theme()`.

## Usage

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

font_link(family, href)

font_google(
  family,
  local = TRUE,
  cache = sass_file_cache(dir = cache_context_dir(), max_size = 100 * 1024^2),
  wght = NULL,
  ital = NULL,
  display = c("swap", "auto", "block", "fallback", "optional")
)
```

## Arguments

| | |
|---|---|
| family | A character string with a *single* font family name. |
| 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. |
| display | the font-display @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. |
| href | A URL resource pointing to the font data. |
| local | Whether or not download and bundle local (woff) font files. |
| cache | A sass::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 |

## Value

a list with a special class.

## Local fonts

With local (i.e., self-hosted) fonts, clients (i.e., end users) can render fonts without an internet connection. By default, google_font() will automatically download, cache, and serve font files locally. Non-Google fonts may also be served locally, but you'll have to download and serve local file using something like shiny::addResourcePath() (or similar) and provide the relevant files to a font_face() definiton.

## Remote fonts

With remotely hosted fonts, clients (i.e., end users) need an internet connection to render the fonts. Remote fonts can be implemented using font_google(...,local = FALSE) (hosted via Google), font_link() (hosted via href URL), or font_face() (hosted via src URL).

## References

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

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

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

## Examples

```
# If you have an internet connection, running the following code
# will download, cache, and import the relevant Google Font files
# for local use
theme <- bs_theme(
  base_font = font_google("Fira Sans"),
  code_font = font_google("Fira Code"),
  heading_font = font_google("Fredoka One")
)
if (interactive()) {
  bs_theme_preview(theme)
}

# 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')")
)
theme <- bs_theme(base_font = c)
if (interactive()) {
  bs_theme_preview(theme)
}
```

---

# run_with_themer *Theme customization UI*

---

## Description

A 'real-time' theme customization UI that you can use to easily make common tweaks to Bootstrap variables and immediately see how they would affect your app's appearance. There are two ways you can launch the theming UI. For most Shiny apps, just use `run_with_themer()` in place of `shiny::runApp()`; they should take the same arguments and work the same way. Alternatively,

you can call the `bs_themer()` function from inside your server function (or in an R Markdown app that is using `runtime: shiny`, you can call this from any code chunk). Note that this function is only intended to be used for development!

## Usage

```
run_with_themer(appDir = getwd(), ..., gfonts = TRUE, gfonts_update = FALSE)

bs_themer(gfonts = TRUE, gfonts_update = FALSE)
```

## Arguments

| | |
|---|---|
| `appDir` | The application to run. This can be a file or directory path, or a `shiny::shinyApp()` object. See `shiny::runApp()` for details. |
| `...` | Additional parameters to pass through to `shiny::runApp()`. |
| `gfonts` | whether or not to detect Google Fonts and wrap them in `font_google()` (so that their font files are automatically imported). |
| `gfonts_update` | whether or not to update the internal database of Google Fonts. |

## Details

To help you utilize the changes you see in the preview, this utility prints `bs_theme()` code to the R console.

## Value

nothing. These functions are called for their side-effects.

## Limitations

Currently, this utility only works with Bootstrap 4. We hope to add Bootstrap 3 compatibility in the future. Also, the color picker currently doesn't render correctly on IE11.

It also only works with Shiny apps and R Markdown apps that use the Shiny runtime. It's not possible to perform real-time preview for static R Markdown documents.

Note that only CSS generated with `bs_dependency_defer()` will be instantly reflected in theme preview.

## Examples

```
library(shiny)

# Initialize Bootstrap 4 with Bootstrap 3 compatibility shim
theme <- bs_theme(version = 4, bg = "black", fg = "white")

ui <- fluidPage(
  theme = theme,
  h1("Heading 1"),
  h2("Heading 2"),
  p(
```

```
    "Paragraph text;",
    tags$a(href = "https://www.rstudio.com", "a link")
  ),
  p(
    actionButton("cancel", "Cancel"),
    actionButton("continue", "Continue", class = "btn-primary")
  ),
  tabsetPanel(
    tabPanel("First tab",
      "The contents of the first tab"
    ),
    tabPanel("Second tab",
      "The contents of the second tab"
    )
  )
)

if (interactive()) {
  run_with_themer(shinyApp(ui, function(input, output) {}))
}
```

---

`theme_bootswatch` *Obtain a theme's Bootswatch theme name*

---

## Description

Obtain a theme's Bootswatch theme name

## Usage

```
theme_bootswatch(theme)
```

## Arguments

| | |
|---|---|
| `theme` | a `bs_theme()` object. |

## Value

the Bootswatch theme named used (if any) in the `theme`.

## theme_version *Obtain a theme's Bootstrap version*

### Description

Obtain a theme's Bootstrap version

### Usage

```
theme_version(theme)
```

### Arguments

| | |
|---|---|
| theme | a bs_theme() object. |

### Value

the major version of Bootstrap used in the theme.

## versions *Available Bootstrap versions*

### Description

Available Bootstrap versions

### Usage

```
versions()

version_default()
```

### Value

A list of the Bootstrap versions available.

# Index