Package 'bslib'

December 16, 2022

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

Version 0.4.2

Description Simplifies custom 'CSS' styling of both 'shiny' and 'rmarkdown' via 'Bootstrap' 'Sass'. Supports 'Bootstrap' 3, 4 and 5 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 (>= 0.5.4), jsonlite, sass (>= 0.4.0), jquerylib (>= 0.1.3), rlang, cachem, memoise (>= 2.0.1), base64enc, mime

Suggests shiny (>= 1.6.0), rmarkdown (>= 2.7), thematic, knitr, testthat, withr, rappdirs, curl, magrittr, fontawesome, bsicons

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

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' 'card.R' 'deprecated.R' 'files.R' 'imports.R' 'layout.R' 'nav-items.R' 'nav-update.R' 'navs-legacy.R' 'navs.R' 'onLoad.R' 'page.R' 'precompiled.R' 'print.R' 'shiny-devmode.R' 'staticimports.R' 'utils-shiny.R' 'utils-tags.R' 'value-box.R' 'version-default.R' 'versions.R'

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

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

Config/testthat/edition 3

Config/Needs/routine desc, renv

Config/Needs/website brio, dplyr, glue, purrr, rprojroot, stringr, tidyr, DT, leaflet, plotly, htmlwidgets, shiny, ggplot2, svglite

Config/Needs/deploy BH, DT, ggplot2, hexbin, lattice, lubridate, dplyr, modelr, nycflights13, histoslider, reactable, rprojroot, rsconnect, plotly, leaflet, gt, shiny, htmlwidgets

NeedsCompilation no

Author Carson Sievert [aut, cre] (<https://orcid.org/0000-0002-4958-2844>),
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 2022-12-16 10:30:06 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_functions(theme, functions)

bs_add_mixins(theme, mixins)

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(). • 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)
functions	A character vector or sass::sass_file() containing functions definitions.
mixins	A character vector or sass::sass_file() containing mixin definitions.

Value

a modified bs_theme() object.

Functions

- bs_add_variables(): Add Bootstrap Sass variable defaults
- bs_add_rules(): Add additional Sass rules
- bs_add_functions(): Add additional Sass functions
- bs_add_mixins(): Add additional Sass 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_add_variables(
  bs_theme(bootswatch = "sketchy", primary = "orange"),
  "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(theme, "body-color" = "$success", .where = "declarations")
preview_button(theme)

# 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_add_rules(bs_theme(), 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, memoise = TRUE)
```

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.
memoise	whether or not to memoise (i.e., cache) func results for a short period of time. The default, TRUE, can have large performance benefits when many instances of the same themable widget are rendered. Note that you may want to avoid memoisation if func relies on side-effects (e.g., files on-disk) that need to change for each themable widget instance.

Value

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

References

https://rstudio.github.io/bslib/articles/custom-components.html

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). Defaults to the currently recommended version for new projects (currently Bootstrap 5).
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
bs4 <- bs_theme(version = 4)
bs_remove(bs4, c("_print", "_carousel"))

# Remove BS3 compatibility layer
bs_remove(bs4, "bs3compat")
```

`bs_theme` *Create a Bootstrap theme*

Description

Creates a Bootstrap theme object, where you can:

- 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 `...`.

To learn more about how to implement custom themes, as well as how to use them inside Shiny and R Markdown, 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,
  font_scale = 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,
  font_scale = NULL
)

is_bs_theme(x)
```

Arguments

version	The major version of Bootstrap to use (see versions() for possible values). Defaults to the currently recommended version for new projects (currently Bootstrap 5).
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.

font_scale	A scalar multiplier to apply to the base font size. For example, a value of 1.5 scales font sizes to 150% and a value of 0.8 scales to 80%. Must be a positive number.
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 = font_collection(font_google("Pacifico", local = FALSE), "Roboto", "sans-serif"))
```

References

https://rstudio.github.io/bslib/articles/bslib.html

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 <- bs_add_variables(theme, "my-class-color" = "red")
theme <- bs_add_rules(theme, ".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_get(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)
```

card *A Bootstrap card component*

Description

A general purpose container for grouping related UI elements together with a border and optional padding. To learn more about `card()`s, see this article.

Usage

```
card(
  ...,
  full_screen = FALSE,
  height = NULL,
  class = NULL,
  wrapper = card_body
)
```

Arguments

`...`	Unnamed arguments can be any valid child of an htmltools tag (which includes card items such as `card_body()`. Named arguments become HTML attributes on returned UI element.
`full_screen`	If `TRUE`, an icon will appear when hovering over the card body. Clicking the icon expands the card to fit viewport size. Consider pairing this feature with `card_body_fill()` to get output that responds to changes in the size of the card.
`height`	Any valid CSS unit (e.g., `height="200px"`).
`class`	Additional CSS classes for the returned UI element.
`wrapper`	A function (which returns a UI element) to call on unnamed arguments in `...` which are not already card item(s) (like `card_header()`, `card_body()`, etc.). Note that non-card items are grouped together into one `wrapper` call (e.g. given `card("a", "b", card_body("c"), "d")`, `wrapper` would be called twice, once with `"a"` and `"b"` and once with `"d"`). Consider setting `wrapper` to card_body_fill if the entire card wants responsive sizing or `NULL` to avoid wrapping altogether

Value

A `htmltools::div()` tag.

See Also

`card_body()` for putting stuff inside the card.

`navs_tab_card()` for cards with multiple tabs.

`layout_column_wrap()` for laying out multiple cards (or multiple columns inside a card).

Examples

```
library(htmltools)

if (interactive()) {
  card(
    full_screen = TRUE,
    card_header(
      "This is the header"
    ),
    card_body(
      p("This is the body."),
      p("This is still the body.")
    ),
    card_footer(
      "This is the footer"
    )
  )
}
```

---

card_body *Card items*

Description

Components designed to be provided as direct children of a `card()`. For a general overview of the `card()` API, see this article.

Usage

```
card_body(..., fill = FALSE, height = NULL, class = NULL)

card_body_fill(
  ...,
  gap = NULL,
  max_height = NULL,
  max_height_full_screen = max_height,
  min_height = NULL,
  class = NULL
)

card_title(..., container = htmltools::h5)

card_header(..., class = NULL, container = htmltools::div)

card_footer(..., class = NULL)
```

```
card_image(
  file,
  ...,
  href = NULL,
  border_radius = c("top", "bottom", "all", "none"),
  mime_type = NULL,
  class = NULL,
  height = NULL,
  width = NULL,
  container = card_body_fill
)

as.card_item(x)

is.card_item(x)
```

Arguments

`...`	Unnamed arguments can be any valid child of an htmltools tag. Named arguments become HTML attributes on returned UI element.
`fill`	whether to allow the `card_body()` to grow and shrink to fit its `card()`.
`height`	Any valid CSS unit (e.g., `height="200px"`).
`class`	Additional CSS classes for the returned UI element.
`gap`	A CSS length unit defining the gap (i.e., spacing) between elements provided to `...`.
`max_height, max_height_full_screen, min_height`	Any valid CSS length unit.
`container`	a function to generate an HTML element to contain the image.
`file`	a file path pointing an image. The image will be base64 encoded and provided to the `src` attribute of the `<img>`. Alternatively, you may set this value to `NULL` and provide the `src` yourself.
`href`	an optional URL to link to.
`border_radius`	where to apply `border-radius` on the image.
`mime_type`	the mime type of the `file`.
`width`	Any valid CSS unit (e.g., `width="100%"`).
`x`	an object to test (or coerce to) a card item.

Value

An `htmltools::div()` tag.

Functions

- `card_body()`: A general container for the "main content" of a `card()`.

- `card_body_fill()`: Similar to `card_body(fill = TRUE)`, but also marks the return element as a "fill container" (via `htmltools::bindFillRole()`) so that its immediate children are allowed to grow and shrink to fit.
- `card_title()`: Similar to `card_header()` but without the border and background color.
- `card_header()`: A header (with border and background color) for the `card()`. Typically appears before a `card_body()`.
- `card_footer()`: A header (with border and background color) for the `card()`. Typically appears after a `card_body()`.
- `card_image()`: Include static (i.e., pre-generated) images.
- `as.card_item()`: Mark an object as a card item. This will prevent the `card()` from putting the object inside a `wrapper` (i.e., a `card_body()`).

See Also

`card()` for creating a card component.

`navs_tab_card()` for cards with multiple tabs.

`layout_column_wrap()` for laying out multiple cards (or multiple columns inside a card).

`font_face` *Helpers for importing web fonts*

Description

`font_google()`, `font_link()`, and `font_face()` are all re-exported from the **sass** package (see `sass::font_face()` for details). For a quick example of how to use these functions with `bs_theme()`, see the examples section below.

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)
}
```

`layout_column_wrap` *A grid-like, column-first, layout*

Description

Wraps a 1d sequence of UI elements into a 2d grid. The number of columns (and rows) in the grid dependent on the column `width` as well as the size of the display. For more explanation and illustrative examples, see here

Usage

```
layout_column_wrap(
  width,
  ...,
  fixed_width = FALSE,
  heights_equal = c("all", "row"),
  fill = TRUE,
  height = NULL,
  height_mobile = NULL,
  gap = NULL,
  class = NULL
)
```

Arguments

`width` The desired width of each card, which can be any of the following:

- A (unit-less) number between 0 and 1.
  - This should be specified as `1/num`, where `num` represents the number of desired columns.
- A CSS length unit
  - Either the minimum (when `fixed_width=FALSE`) or fixed width (`fixed_width=TRUE`).
- `NULL`

- Allows power users to set the `grid-template-columns` CSS property manually, either via a `style` attribute or a CSS stylesheet.

`...`	Unnamed arguments should be UI elements (e.g., `card()`) Named arguments become attributes on the containing htmltools::tag element.
`fixed_width`	Whether or not to interpret the `width` as a minimum (`fixed_width=FALSE`) or fixed (`fixed_width=TRUE`) width when it is a CSS length unit.
`heights_equal`	If `"all"` (the default), every card in every row of the grid will have the same height. If `"row"`, then every card in *each* row of the grid will have the same height, but heights may vary between rows.
`fill`	whether or not the grid items should grow to fill the row height.
`height`	Any valid CSS unit (e.g., `height="200px"`).
`height_mobile`	Any valid CSS unit to use for the height when on mobile devices (or narrow windows).
`gap`	A CSS length unit defining the gap (i.e., spacing) between elements provided to `....`
`class`	Additional CSS classes for the returned UI element.

Examples

```
x <- card("A simple card")
# Always has 2 columns (on non-mobile)
layout_column_wrap(1/2, x, x, x)
# Has three columns when viewport is wider than 750px
layout_column_wrap("250px", x, x, x)
```

---

nav *Navigation items*

Description

Create nav item(s) for use inside nav containers (e.g., `navs_tab()`, `navs_bar()`, etc).

Usage

```
nav(title, ..., value = title, icon = NULL)

nav_menu(title, ..., value = title, icon = NULL, align = c("left", "right"))

nav_content(value, ..., icon = NULL)

nav_item(...)

nav_spacer()
```

Arguments

`title`	A title to display. Can be a character string or UI elements (i.e., tags).
`...`	Depends on the function: • For `nav()` and `nav_content()`: UI elements (i.e., tags) to display when the item is active. • For `nav_menu()`: a collection of nav items (e.g., `nav()`, `nav_item()`). • For `nav_item()`: UI elements (i.e., tags) to place directly in the navigation panel (e.g., search forms, links to external content, etc).
`value`	A character string to assign to the nav item. This value may be supplied to the relevant container's `selected` argument in order to show particular nav item's content immediately on page load. This value is also useful for programmatically updating the selected content via `nav_select()`, `nav_hide()`, etc (updating selected tabs this way is often useful for showing/hiding panels of content via other UI controls like `shiny::radioButtons()` – in this scenario, consider using `nav_content()` with `navs_hidden()`).
`icon`	Optional icon to appear next to the nav item's `title`.
`align`	horizontal alignment of the dropdown menu relative to dropdown toggle.

Value

A nav item that may be passed to a nav container (e.g. `navs_tab()`).

Functions

- `nav()`: Content to display when the given item is selected.
- `nav_menu()`: Create a menu of nav items.
- `nav_content()`: Create nav content for use inside `navs_hidden()` (for creating custom navigation controls via `navs_select()`),
- `nav_item()`: Place arbitrary content in the navigation panel (e.g., search forms, links to external content, etc.)
- `nav_spacer()`: Adding spacing between nav items.

See Also

`navs_tab()`, `nav_select()`.

---

navs_tab *Navigation containers*

Description

Render a collection of `nav()` items into a container.

Usage

```
navs_tab(..., id = NULL, selected = NULL, header = NULL, footer = NULL)

navs_pill(..., id = NULL, selected = NULL, header = NULL, footer = NULL)

navs_pill_list(
  ...,
  id = NULL,
  selected = NULL,
  header = NULL,
  footer = NULL,
  well = TRUE,
  fluid = TRUE,
  widths = c(4, 8)
)

navs_hidden(..., id = NULL, selected = NULL, header = NULL, footer = NULL)

navs_bar(
  ...,
  title = NULL,
  id = NULL,
  selected = NULL,
  position = c("static-top", "fixed-top", "fixed-bottom"),
  header = NULL,
  footer = NULL,
  bg = NULL,
  inverse = "auto",
  collapsible = TRUE,
  fluid = TRUE
)

navs_tab_card(
  ...,
  id = NULL,
  selected = NULL,
  title = NULL,
  header = NULL,
  footer = NULL,
  height = NULL,
  full_screen = FALSE,
  wrapper = card_body
)

navs_pill_card(
  ...,
  id = NULL,
  selected = NULL,
```

```
  title = NULL,
  header = NULL,
  footer = NULL,
  height = NULL,
  placement = c("above", "below"),
  full_screen = FALSE,
  wrapper = card_body
)
```

Arguments

...	a collection of nav() items.
id	a character string used for dynamically updating the container (see nav_select()).
selected	a character string matching the value of a particular nav() item to selected by default.
header	UI element(s) (tags) to display *above* the nav content.
footer	UI element(s) (tags) to display *below* the nav content.
well	TRUE to place a well (gray rounded rectangle) around the navigation list.
fluid	TRUE to use fluid layout; FALSE to use fixed layout.
widths	Column widths of the navigation list and tabset content areas respectively.
title	A (left-aligned) title to place in the card header/footer. If provided, other nav items are automatically right aligned.
position	Determines whether the navbar should be displayed at the top of the page with normal scrolling behavior ("static-top"), pinned at the top ("fixed-top"), or pinned at the bottom ("fixed-bottom"). Note that using "fixed-top" or "fixed-bottom" will cause the navbar to overlay your body content, unless you add padding, e.g.: tags$style(type="text/css", "body {padding-top: 70px;}")
bg	a CSS color to use for the navbar's background color.
inverse	Either TRUE for a light text color or FALSE for a dark text color. If "auto" (the default), the best contrast to bg is chosen.
collapsible	TRUE to automatically collapse the navigation elements into a menu when the width of the browser is less than 940 pixels (useful for viewing on smaller touch-screen device)
height	Any valid CSS unit (e.g., height="200px").
full_screen	If TRUE, an icon will appear when hovering over the card body. Clicking the icon expands the card to fit viewport size. Consider pairing this feature with card_body_fill() to get output that responds to changes in the size of the card.
wrapper	A function (which returns a UI element) to call on unnamed arguments in ... which are not already card item(s) (like card_header(), card_body(), etc.). Note that non-card items are grouped together into one wrapper call (e.g. given card("a", "b", card_body("c"), "d"), wrapper would be called twice, once with "a" and "b" and once with "d"). Consider setting wrapper to card_body_fill if the entire card wants responsive sizing or NULL to avoid wrapping altogether
placement	placement of the nav items relative to the content.

See Also

nav(), nav_select().

Examples

```
library(shiny)

nav_items <- function(prefix) {
  list(
    nav("a", paste(prefix, ": tab a content")),
    nav("b", paste(prefix, ": tab b content")),
    nav_item(
    tags$a(icon("github"), "Shiny", href = "https://github.com/rstudio/shiny", target = "_blank")
    ),
    nav_spacer(),
    nav_menu(
      "Other links", align = "right",
      nav("c", paste(prefix, ": tab c content")),
      nav_item(
     tags$a(icon("r-project"), "RStudio", href = "https://rstudio.com", target = "_blank")
      )
    )
  )
}

if (interactive()) {
  shinyApp(
    page_navbar(
      title = "page_navbar()",
      bg = "#0062cc",
      !!!nav_items("page_navbar()"),
      footer = div(
        style = "width:80%; margin: 0 auto",
        h4("navs_tab()"),
        navs_tab(!!!nav_items("navs_tab()")),
        h4("navs_pill()"),
        navs_pill(!!!nav_items("navs_pill()")),
        h4("navs_tab_card()"),
        navs_tab_card(!!!nav_items("navs_tab_card()")),
        h4("navs_pill_card()"),
        navs_pill_card(!!!nav_items("navs_pill_card()")),
        h4("navs_pill_list()"),
        navs_pill_list(!!!nav_items("navs_pill_list()"))
      )
    ),
    function(...) { }
  )
}
```

nav_select *Dynamically update nav containers*

Description

Functions for dynamically updating nav containers (e.g., select, insert, and remove nav items). These functions require an `id` on the nav container to be specified.

Usage

```
nav_select(id, selected = NULL, session = getDefaultReactiveDomain())

nav_insert(
  id,
  nav,
  target = NULL,
  position = c("after", "before"),
  select = FALSE,
  session = getDefaultReactiveDomain()
)

nav_remove(id, target, session = getDefaultReactiveDomain())

nav_show(id, target, select = FALSE, session = getDefaultReactiveDomain())

nav_hide(id, target, session = getDefaultReactiveDomain())
```

Arguments

`id`	a character string used to identify the nav container.
`selected`	a character string used to identify a particular `nav()` item.
`session`	a shiny session object (the default should almost always be used).
`nav`	a `nav()` item.
`target`	The `value` of an existing `nav()` item, next to which tab will be added. If removing: the `value` of the `nav()` item that you want to remove.
`position`	Should `nav` be added before or after the target?
`select`	Should `nav` be selected upon being inserted?

See Also

`nav()`, `navs_tab()`.

Examples

```
can_browse <- function() interactive() && require("shiny")

# Selecting a tab
if (can_browse()) {
  shinyApp(
    page_fluid(
      radioButtons("item", "Choose", c("A", "B")),
      navs_hidden(
        id = "container",
        nav_content("A", "a"),
        nav_content("B", "b")
      )
    ),
    function(input, output) {
      observe(nav_select("container", input$item))
    }
  )
}

# Inserting and removing
if (can_browse()) {
  ui <- page_fluid(
    actionButton("add", "Add 'Dynamic' tab"),
    actionButton("remove", "Remove 'Foo' tab"),
    navs_tab(
      id = "tabs",
      nav("Hello", "hello"),
      nav("Foo", "foo"),
      nav("Bar", "bar tab")
    )
  )
  server <- function(input, output) {
    observeEvent(input$add, {
      nav_insert(
        "tabs", target = "Bar", select = TRUE,
        nav("Dynamic", "Dynamically added content")
      )
    })
    observeEvent(input$remove, {
      nav_remove("tabs", target = "Foo")
    })
  }
  shinyApp(ui, server)
}
```

---

page *Create a Bootstrap page*

Description

These functions are small wrappers around shiny's page constructors (i.e., `shiny::fluidPage()`, `shiny::navbarPage()`, etc) that differ in two ways:

- The `theme` parameter defaults bslib's recommended version of Bootstrap (for new projects).
- The return value is rendered as an static HTML page when printed interactively at the console.

Usage

```
page(..., title = NULL, theme = bs_theme(), lang = NULL)

page_fluid(..., title = NULL, theme = bs_theme(), lang = NULL)

page_fixed(..., title = NULL, theme = bs_theme(), lang = NULL)

page_fill(..., padding = 0, title = NULL, theme = bs_theme(), lang = NULL)

page_navbar(
  ...,
  title = NULL,
  id = NULL,
  selected = NULL,
  position = c("static-top", "fixed-top", "fixed-bottom"),
  header = NULL,
  footer = NULL,
  bg = NULL,
  inverse = "auto",
  collapsible = TRUE,
  fluid = TRUE,
  theme = bs_theme(),
  window_title = NA,
  lang = NULL
)
```

Arguments

`...`	The contents of the document body.
`title`	The browser window title (defaults to the host URL of the page)
`theme`	One of the following: • `NULL` (the default), which implies a "stock" build of Bootstrap 3. • A `bslib::bs_theme()` object. This can be used to replace a stock build of Bootstrap 3 with a customized version of Bootstrap 3 or higher. • A character string pointing to an alternative Bootstrap stylesheet (normally a css file within the www directory, e.g. `www/bootstrap.css`).
`lang`	ISO 639-1 language code for the HTML page, such as "en" or "ko". This will be used as the lang in the `<html>` tag, as in `<html lang="en">`. The default (NULL) results in an empty string.

padding	Padding to use for the body. This can be a numeric vector (which will be interpreted as pixels) or a character vector with valid CSS lengths. The length can be between one and four. If one, then that value will be used for all four sides. If two, then the first value will be used for the top and bottom, while the second value will be used for left and right. If three, then the first will be used for top, the second will be left and right, and the third will be bottom. If four, then the values will be interpreted as top, right, bottom, and left respectively.
id	a character string used for dynamically updating the container (see `nav_select()`).
selected	a character string matching the `value` of a particular `nav()` item to selected by default.
position	Determines whether the navbar should be displayed at the top of the page with normal scrolling behavior (`"static-top"`), pinned at the top (`"fixed-top"`), or pinned at the bottom (`"fixed-bottom"`). Note that using `"fixed-top"` or `"fixed-bottom"` will cause the navbar to overlay your body content, unless you add padding, e.g.: `tags$style(type="text/css", "body {padding-top: 70px;}")`
header	UI element(s) (tags) to display *above* the nav content.
footer	UI element(s) (tags) to display *below* the nav content.
bg	a CSS color to use for the navbar's background color.
inverse	Either `TRUE` for a light text color or `FALSE` for a dark text color. If `"auto"` (the default), the best contrast to `bg` is chosen.
collapsible	`TRUE` to automatically collapse the navigation elements into a menu when the width of the browser is less than 940 pixels (useful for viewing on smaller touch-screen device)
fluid	`TRUE` to use fluid layout; `FALSE` to use fixed layout.
window_title	the browser window title. The default value, `NA`, means to use any character strings that appear in `title` (if none are found, the host URL of the page is displayed by default).

See Also

`shiny::bootstrapPage()`

`shiny::fluidPage()`

`shiny::fixedPage()`

`shiny::fillPage()`

`shiny::navbarPage()`

`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

- Doesn't work with Bootstrap 3.
- Doesn't work with IE11.
- Only works inside Shiny apps and `runtime: shiny` R Markdown documents.
  - Can't be used with static R Markdown documents.
  - Can be used to some extent with `runtime: shiny_prerendered`, but only UI rendered through a `context="server"` may update in real-time.
- Doesn't work with '3rd party' custom widgets that don't make use of `bs_dependency_defer()` or `bs_current_theme()`.

Examples

```
library(shiny)

ui <- fluidPage(
  theme = bs_theme(bg = "black", fg = "white"),
  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.

value_box *Value box*

Description

An opinionated (card()-powered) box, designed for displaying a value and title. Optionally, a showcase can provide for context for what the value represents (for example, it could hold a bsicons::bs_icon(), or even a shiny::plotOutput()).

Usage

```
value_box(
  title,
  value,
  ...,
  showcase = NULL,
  showcase_layout = showcase_left_center(),
  full_screen = FALSE,
  theme_color = "primary",
  height = NULL,
  class = NULL
)

showcase_left_center(
  width = 0.3,
  max_height = "100px",
  max_height_full_screen = 0.67
```

```
)

showcase_top_right(
  width = 0.3,
  max_height = "75px",
  max_height_full_screen = 0.67
)
```

Arguments

`title, value`	a `htmltools::tag()` child to display above `value`. If a string a provided, it's automatically wrapped in a header tag.
`...`	Unnamed arguments may be any `htmltools::tag()` children to display below `value`.. Named arguments become attributes on the containing element.
`showcase`	a `htmltools::tag()` child to showcase (e.g., a `bsicons::bs_icon()`, a `plotly::plotlyOutput()`, etc).
`showcase_layout`	either `showcase_left_center()` or `showcase_top_right()`.
`full_screen`	If `TRUE`, an icon will appear when hovering over the card body. Clicking the icon expands the card to fit viewport size. Consider pairing this feature with `card_body_fill()` to get output that responds to changes in the size of the card.
`theme_color`	a theme color to use for the background color. Should match a name in the Bootstrap Sass variable `$theme-colors` (e.g., `"secondary"`, `"success"`, `"danger"`, etc)
`height`	Any valid CSS unit (e.g., `height="200px"`).
`class`	utility classes for customizing the appearance of the summary card. Use `bg-*` and `text-*` classes (e.g, `"bg-danger"` and `"text-light"`) to customize the background/foreground colors.
`width`	one of the following: • A proportion (i.e., a number between 0 and 1) of available width to allocate to the showcase. • A vector of length 2 valid CSS unit defining the width of each column (for `showcase_left_center()` the 1st unit defines the showcase width and for `showcase_top_right` the 2nd unit defines the showcase width). Note that any units supported by the CSS grid `grid-template-columns` property may be used (e.g., `fr` units).
`max_height, max_height_full_screen`	A proportion (i.e., a number between 0 and 1) or any valid CSS unit defining the showcase max_height.

See Also

`card()`

Examples

```
library(htmltools)

if (interactive()) {
  value_box(
    "KPI Title",
    h1(HTML("$1 <i>Billion</i> Dollars")),
    span(
      bsicons::bs_icon("arrow-up"),
      " 30% VS PREVIOUS 30 DAYS"
    ),
    showcase = bsicons::bs_icon("piggy-bank"),
    class = "bg-success"
  )
}
```

`versions` *Available Bootstrap versions*

Description

Available Bootstrap versions

Usage

```
versions()

version_default()
```

Value

A list of the Bootstrap versions available.

Index