Package 'brandr'

May 8, 2026

Title Brand Identity Management Using brand.yml Standard

Version 0.1.0

Description A system to facilitate brand identity management using the brand.yml standard, providing functions to consistently access and apply brand colors, typography, and other visual elements across your R projects.

License MIT + file LICENSE

URL https://danielvartan.github.io/brandr/, https://github.com/danielvartan/brandr/

BugReports https://github.com/danielvartan/brandr/issues/

Depends R (>= 4.3)

Imports checkmate (>= 2.3.2), cli (>= 3.6.3), colorspace (>= 2.1.1), dplyr (>= 1.1.4), here (>= 1.0.1), lifecycle (>= 1.0.4), ggplot2 (>= 3.5.1), grDevices (>= 4.3.0), yaml (>= 2.3.10)

Suggests bslib (>= 0.9.0), covr (>= 3.6.4), hexbin (>= 1.28.5), knitr (>= 1.49), magrittr (>= 2.0.3), palmerpenguins (>= 0.1.1), rmarkdown (>= 2.29), spelling (>= 2.3.1), testthat (>= 3.2.2), tidyr (>= 1.3.1)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

NeedsCompilation no

Author Daniel Vartanian [aut, cre, ccp, cph] (ORCID: <https://orcid.org/0000-0001-7782-759X>)

Maintainer Daniel Vartanian <danvartan@gmail.com>

Repository CRAN

Date/Publication 2025-03-10 16:50:13 UTC

Contents

`color_brand_sequential` *Brand color palettes*

Description

[Maturing]
`color_brand_*` functions are wrappers of `interpolate_colors()` for sequential, diverging, and qualitative brand color palettes. They serve as facilitators for getting the colors being used in each brand scale.

Usage

```
color_brand_sequential(n, alpha = NULL, direction = 1)

color_brand_diverging(n, alpha = NULL, direction = 1)

color_brand_qualitative(n, alpha = NULL, direction = 1)
```

Arguments

`n`	This parameter accepts two types of inputs: • If the value is an integer number and `type` is `"seq"` or `"div"`, the function will return a discrete color spectrum with `n` colors • If the value is an integer number and `type` is `"qual"`, the function will return `n` colors from the `colors` parameter, repeating them if necessary • If the value is a `numeric` vector between `0` and `1`, the function will return the color positions at `n` considering a continuous color spectrum ranging from `0` to `1`
`alpha`	(Optional) A number between `0` and `1`, indicating the transparency of the colors (Default: `NULL`).
`direction`	(Optional) A number (`1` or `-1`) indicating the direction of the colors. If `1`, the order remains the same. If `-1`, the order is reversed. (Default: `1`).

Details

Path to `_brand.yml`:

`brandr` will always look for a `_brand.yml` file in the root directory of your project. If the file is not found, an error message will be displayed. You can also set the path to the file manually using the `options()` function:

```
options(BRANDR_BRAND_YML = "PATH_TO_BRAND.YML")
```

Brand Color Scales:

To control the colors for each brand color scale, assign the desired hexadecimal color codes in a `character` vector to the following options:

- `BRANDR_COLOR_SEQUENTIAL`: For sequential color scales
- `BRANDR_COLOR_DIVERGING`: For diverging color scales
- `BRANDR_COLOR_QUALITATIVE`: For qualitative color scales

You can use `get_brand_color()` to get the hexadecimal color codes from the `_brand.yml` file.

Example:

```
options(
  BRANDR_COLOR_SEQUENTIAL =
    get_brand_color(c("primary", "secondary")),
  BRANDR_COLOR_DIVERGING =
    get_brand_color(c("primary", "white", "secondary")),
  BRANDR_COLOR_QUALITATIVE =
    get_brand_color(c("primary", "secondary", "tertiary"))
)
```

Value

A `character` vector with hexadecimal color codes.

See Also

Other color functions: `interpolate_colors()`

Examples

```
color_brand_sequential(5)
#> [1] "#390963" "#892B4F" "#DA4E3C" "#EA7220" "#FB9706" # Expected

color_brand_diverging(5)
#> [1] "#390963" "#9C84B1" "#FFFFFF" "#FDCB82" "#FB9706" # Expected

color_brand_qualitative(5)
#> [1] "#DA4E3C" "#390963" "#FB9706" "#DA4E3C" "#390963" # Expected

color_brand_qualitative(3, alpha = 0.5)
#> [1] "#DA4E3C80" "#39096380" "#FB970680" # Expected
```

get_brand_color *Get brand colors*

Description

[Maturing]

`get_brand_color()` retrieves color codes from the `_brand.yml` file.

Usage

```
get_brand_color(color, alpha = NULL)
```

Arguments

`color`	A `character` vector indicating the name of colors present in the `color` section of the `_brand.yml` file.
`alpha`	(Optional) A number between `0` and `1`, indicating the transparency of the colors (Default: `NULL`).

Details

Path to `_brand.yml`:

`brandr` will always look for a `_brand.yml` file in the root directory of your project. If the file is not found, an error message will be displayed. You can also set the path to the file manually using the `options()` function:

```
options(BRANDR_BRAND_YML = "PATH_TO_BRAND.YML")
```

Value

A `character` vector with hexadecimal color codes.

See Also

Other utility functions: `get_brand_color_mix()`, `get_brand_color_tint()`, `get_brand_font()`

Examples

```
get_brand_color("primary")
#> [1] "#DA4E3C" # Expected

get_brand_color("secondary")
#> [1] "#390963" # Expected

get_brand_color("tertiary")
#> [1] "#FB9706" # Expected

get_brand_color("tertiary", alpha = 0.5)
```

```
#> [1] "#FB970680" # Expected

get_brand_color(c("primary", "secondary"))
#> [1] "#DA4E3C" "#390963" # Expected

get_brand_color(c("red", "purple", "orange"))
#> [1] "#DA4E3C" "#390963" "#F06F20" # Expected
```

`get_brand_color_mix` *Get a mix of brand colors*

Description

[Maturing]

`get_brand_color_mix()` mixes two specific brand colors.

Usage

```
get_brand_color_mix(
  position = 500,
  color_1 = "primary",
  color_2 = "secondary",
  alpha = 0.5
)
```

Arguments

`position`	(Optional) A `numeric` vector indicating the position of the brand color in the range of tints. The range of positions is from `0` to `1000` (Default: `500`).
`color_1`, `color_2`	(Optional) A `character` string indicating the name of a color present in the `color` section of the `_brand.yml` file (Default: `primary`, `secondary`).
`alpha`	(Optional) A number between `0` and `1` indicating the alpha (transparency) of the color mix (Default: `0.5`).

Details

Path to `_brand.yml`:

`brandr` will always look for a `_brand.yml` file in the root directory of your project. If the file is not found, an error message will be displayed. You can also set the path to the file manually using the `options()` function:

```
options(BRANDR_BRAND_YML = "PATH_TO_BRAND.YML")
```

Value

A `character` vector with hexadecimal color codes.

See Also

Other utility functions: `get_brand_color()`, `get_brand_color_tint()`, `get_brand_font()`

Examples

```
get_brand_color_mix(
  position = 500,
  color_1 = "primary",
  color_2 = "secondary",
  alpha = 0.5
)
#> [1] "#8A2C50" # Expected

get_brand_color_mix(
  position = c(250, 500, 750),
  color_1 = "primary",
  color_2 = "secondary",
  alpha = 0.25
)
#> [1] "#591E23" "#B23D46" "#D89EA2" # Expected
```

---

get_brand_color_tint *Get tints of brand colors*

Description

[Maturing]

`get_brand_color_tint()` generates a range of tints (color variations) for a specific brand color, from black (position 0) through the brand color (position 500) to white (position 1000).

Usage

```
get_brand_color_tint(position = 500, color = "primary")
```

Arguments

`position`	(Optional) A `numeric` vector indicating the position of the brand color in the range of tints. The range of positions is from `0` to `1000` (Default: `500`).
`color`	(Optional) A `character` string indicating the name of a color present in the `color` section of the `_brand.yml` file (Default: `"primary"`).

Details

Path to `_brand.yml`:

`brandr` will always look for a `_brand.yml` file in the root directory of your project. If the file is not found, an error message will be displayed. You can also set the path to the file manually using the `options()` function:

```
options(BRANDR_BRAND_YML = "PATH_TO_BRAND.YML")
```

Value

A character vector with hexadecimal color codes.

See Also

Other utility functions: get_brand_color(), get_brand_color_mix(), get_brand_font()

Examples

```
seq(0, 1000, 250)
#> [1]    0  250  500  750 1000 # Expected

get_brand_color_tint(seq(0, 1000, 250), color = "primary")
#> [1] "#000000" "#6D271E" "#DA4E3C" "#ECA69D" "#FFFFFF" # Expected
```

---

get_brand_font *Get brand fonts/typefaces*

Description

[Maturing]

get_brand_font() retrieves the names of fonts/typefaces in the _brand.yml file.

Usage

```
get_brand_font(font)
```

Arguments

font	A character vector indicating the name of fonts/typefaces categories present in the typography section of the _brand.yml file.

Details

Path to _brand.yml:

brandr will always look for a _brand.yml file in the root directory of your project. If the file is not found, an error message will be displayed. You can also set the path to the file manually using the options() function:

```
options(BRANDR_BRAND_YML = "PATH_TO_BRAND.YML")
```

Value

A character vector with fonts/typeface names.

See Also

Other utility functions: get_brand_color(), get_brand_color_mix(), get_brand_color_tint()

Examples

```
get_brand_font("base")
#> [1] "Open Sans" # Expected

get_brand_font("headings")
#> [1] "Rubik" # Expected

get_brand_font("monospace")
#> [1] "IBM Plex Mono" # Expected

get_brand_font("monospace-block")
#> [1] "IBM Plex Mono" # Expected

get_brand_font(c("base", "headings"))
#> [1] "Open Sans" "Rubik" # Expected
```

---

interpolate_colors *Interpolate colors*

Description

[Maturing]

`interpolate_colors()` interpolate colors for sequential, diverging, and qualitative color scales.

Usage

```
interpolate_colors(
  n,
  colors = getOption("BRANDR_COLOR_SEQUENTIAL"),
  type = "seq",
  alpha = NULL,
  direction = 1,
  ...
)
```

Arguments

`n`	This parameter accepts two types of inputs: • If the value is an integer number and `type` is `"seq"` or `"div"`, the function will return a discrete color spectrum with `n` colors • If the value is an integer number and `type` is `"qual"`, the function will return `n` colors from the `colors` parameter, repeating them if necessary • If the value is a `numeric` vector between `0` and `1`, the function will return the color positions at `n` considering a continuous color spectrum ranging from `0` to `1`
`colors`	(Optional) A `character` vector of colors to use in the scale. If `NULL`, `brandr` will choose the colors based on the `type` argument.

type	(Optional) A character string indicating the type of color scale: "seq"/"sequential", "div"/"diverging", or "qual"/"qualitative" (Default: seq).
alpha	(Optional) A number between 0 and 1, indicating the transparency of the colors (Default: NULL).
direction	(Optional) A number (1 or -1) indicating the direction of the colors. If 1, the order remains the same. If -1, the order is reversed. (Default: 1).
...	Additional arguments passed to colorRampPalette() when creating the color ramp. Only valid when type is "seq" or "div".

Value

A character vector with hexadecimal color codes.

See Also

Other color functions: color_brand_sequential()

Examples

```
interpolate_colors(3, colors = c("red", "blue"), type = "seq")
#> [1] "#FF0000" "#7F007F" "#0000FF" # Expected

interpolate_colors(3, colors = c("red", "blue"), direction = -1)
#> [1] "#0000FF" "#7F007F" "#FF0000" # Expected

interpolate_colors(3, colors = c("red", "blue"), alpha = 0.5)
#> [1] "#FF000080" "#7F007F80" "#0000FF80" # Expected

# `type = "seq"` and `type = "div"` produce the same result
interpolate_colors(3, colors = c("red", "white", "blue"), type = "div")
#> [1] "#FF0000" "#FFFFFF" "#0000FF" # Expected

interpolate_colors(3, colors = c("red", "blue"), type = "qual")
#> [1] "#FF0000" "#0000FF" "#FF0000" # Expected
```

---

scale_brand *Brand color scales for* ggplot2

Description

[Maturing]
scale_*_brand_*() functions provide color scales for ggplot2 based on brand colors defined in the _brand.yml file. These functions create discrete, continuous, or binned scales with sequential, diverging, or qualitative color palettes that match your brand identity.

Usage

```
scale_brand(
  aesthetics = "color",
  scale_type = "c",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_color_brand_d(
  aesthetics = "color",
  scale_type = "d",
  color_type = "qual",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_color_brand_c(
  aesthetics = "color",
  scale_type = "c",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_color_brand_b(
  aesthetics = "color",
  scale_type = "b",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_colour_brand_d(
  aesthetics = "color",
  scale_type = "d",
```

```
  color_type = "qual",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_colour_brand_c(
  aesthetics = "color",
  scale_type = "c",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_colour_brand_b(
  aesthetics = "color",
  scale_type = "b",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_fill_brand_d(
  aesthetics = "fill",
  scale_type = "d",
  color_type = "qual",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)

scale_fill_brand_c(
  aesthetics = "fill",
  scale_type = "c",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
```

```
  reverse = FALSE,
  ...
)

scale_fill_brand_b(
  aesthetics = "fill",
  scale_type = "b",
  color_type = "seq",
  alpha = NULL,
  direction = 1,
  na.value = NA,
  reverse = FALSE,
  ...
)
```

Arguments

aesthetics	(Optional) A character string indicating the name of the aesthetic of the scale (e.g., "color", "fill") (Default: "color").
scale_type	(Optional) A character string indicating the type of scale: "d"/"discrete", "c"/"continuous", or "b"/"binned" (Default: "c").
color_type	(Optional) A character string indicating the type of color scale: "seq"/"sequential", "div"/"diverging", or "qual"/"qualitative" (Default: "seq").
alpha	(Optional) A number between 0 and 1, indicating the transparency of the colors (Default: NULL).
direction	(Optional) A number (1 or -1) indicating the direction of the colors. If 1, the order remains the same. If -1, the order is reversed. (Default: 1).
na.value	(Optional) A character string indicating the color to use for missing values. It must contain a hexadecimal color code or one of the values output by colors() (Default: NA).
reverse	(Optional) A logical flag indicating whether the legend or color bar should be reversed (Default: FALSE).
...	Additional arguments passed to the ggplot2 scale function: discrete_scale(), continuous_scale(), or binned_scale().

Details

Path to _brand.yml:

brandr will always look for a _brand.yml file in the root directory of your project. If the file is not found, an error message will be displayed. You can also set the path to the file manually using the options() function:

```
options(BRANDR_BRAND_YML = "PATH_TO_BRAND.YML")
```

Brand Color Scales:

To control the colors for each brand color scale, assign the desired hexadecimal color codes in a character vector to the following options:

- `BRANDR_COLOR_SEQUENTIAL`: For sequential color scales
- `BRANDR_COLOR_DIVERGING`: For diverging color scales
- `BRANDR_COLOR_QUALITATIVE`: For qualitative color scales

You can use `get_brand_color()` to get the hexadecimal color codes from the `_brand.yml` file.

Example:

```
options(
  BRANDR_COLOR_SEQUENTIAL =
    get_brand_color(c("primary", "secondary")),
  BRANDR_COLOR_DIVERGING =
    get_brand_color(c("primary", "white", "secondary")),
  BRANDR_COLOR_QUALITATIVE =
    get_brand_color(c("primary", "secondary", "tertiary"))
)
```

Value

A ggplot2 scale object.

Examples

```
if (requireNamespace(
      c("palmerpenguins", "tidyr", "ggplot2"),
      quiet = TRUE
    )
   ) {
  library(ggplot2)
  library(palmerpenguins)
  library(tidyr)

  penguins |>
    drop_na(bill_length_mm, species) |>
    ggplot(aes(x = species, y = bill_length_mm, fill = species)) +
    geom_boxplot(outlier.color = get_brand_color("red")) +
    geom_jitter(width = 0.2, alpha = 0.1) +
    scale_fill_brand_d(alpha = 0.5) +
    labs(
      x = "Species",
      y = "Bill Length (mm)",
      fill = "Species"
    ) +
    theme_bw()
}

if (requireNamespace(
      c("palmerpenguins", "tidyr", "ggplot2"),
      quiet = TRUE
    )
   ) {
  library(ggplot2)
  library(palmerpenguins)
```

```
  library(tidyr)

  penguins |>
  drop_na(flipper_length_mm, species) |>
    ggplot(aes(x = flipper_length_mm, fill = species)) +
      geom_histogram(alpha = 0.5, bins = 30, position = "identity") +
      scale_fill_brand_d() +
      labs(
        x = "Flipper Length (mm)",
        y = "Frequency",
        fill = "Species"
      ) +
      theme_bw()
}

if (requireNamespace(
      c("palmerpenguins", "tidyr", "ggplot2"),
      quiet = TRUE
    )
   ) {
  library(ggplot2)
  library(palmerpenguins)
  library(tidyr)

  penguins |>
    drop_na(flipper_length_mm, body_mass_g, species) |>
    ggplot(
      aes(
        x = flipper_length_mm,
        y = body_mass_g,
        color = species,
        shape = species
      )
    ) +
    geom_point(size = 2) +
    geom_smooth(method = "lm", formula = y ~ x, se = FALSE) +
    scale_color_brand_d() +
    labs(
      x = "Flipper Length (mm)",
      y = "Body Mass (g)",
      color = "Species",
      shape = "Species"
    ) +
    theme_bw()
}

if (requireNamespace("ggplot2", quiet = TRUE)) {
  library(ggplot2)

  faithfuld |>
    ggplot(aes(waiting, eruptions, fill = density)) +
    geom_raster() +
    scale_fill_brand_b() +
```

```
    labs(
      x = "Waiting Time to Next Eruption (min)",
      y = "Eruption Time (min)",
      fill = "Density"
    ) +
    theme_bw()
}

if (requireNamespace("ggplot2", quiet = TRUE)) {
  library(ggplot2)
  library(hexbin)

  data.frame(x = runif(10000), y = runif(10000)) |>
    ggplot(aes(x, y)) +
    geom_hex() +
    coord_fixed() +
    scale_fill_brand_c() +
    labs(fill = "") +
    theme_bw()
}
```

Index