Package 'gghdx'

July 31, 2026

Title HDX Theme, Scales, and Other Conveniences for 'ggplot2'

Version 0.2.0

Description A Humanitarian Data Exchange (HDX) theme, color palettes, and scales for 'ggplot2' to allow users to easily follow the HDX visual design guide, including convenience functions for for loading and using the Roboto and Merriweather fonts.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, ggplot2, ggthemes, lifecycle, magrittr, purrr, rlang, showtext, sysfonts, tibble

Depends R (>= 3.5)

LazyData true

Suggests covr, curl, here, knitr, rmarkdown, scales, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://github.com/OCHA-DAP/gghdx

BugReports https://github.com/OCHA-DAP/gghdx/issues

Config/testthat/edition 3

NeedsCompilation no

Author Giulia Martini [aut, cre], Seth Caldwell [aut, cph]

Maintainer Giulia Martini <giulia.martini@un.org>

Repository CRAN

Date/Publication 2026-07-30 23:20:15 UTC

Contents

`df_covid` *Example COVID-19 dataset*

Description

COVID-19 dataset derived from global WHO data. Used to provide simple graph matching the example graphs on the HDX visual guide.

Usage

```
df_covid
```

Format

A data frame with 27 rows and 3 variables:

date Date

cases_monthly Confirmed COVID-19 cases in the past 30 days

flag Flag for that date

Source

https://data.humdata.org/dataviz-guide/visual-identity/#/visual-identity/colours

geom_text_hdx *Text*

Description

Text geoms are useful for labeling plots. They can be used by themselves as scatterplots or in combination with other geoms, for example, for labeling points or for annotating the height of bars. `geom_text_hdx()` adds only text to the plot. `geom_label_hdx()` draws a rectangle behind the text, making it easier to read. The only difference with the base `geom_text()` is that the default font family is Roboto, the HDX body font as of the 2025 redesign, or Source Sans 3 when `design = "legacy"`. `geom_label_hdx()` also incorporates a default dark gray background, white text, and no borders.

Usage

```
geom_text_hdx(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  design = c("2025", "legacy")
)

geom_label_hdx(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  fill = hdx_hex("gray-dark"),
  color = "white",
  fontface = "bold",
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = unit(0.25, "lines"),
  label.r = unit(0.15, "lines"),
  label.size = 0,
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE,
  design = c("2025", "legacy")
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. Cannot be jointy specified with nudge_x or nudge_y. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). • A string nameing the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the

available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

parse — If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.

nudge_x, nudge_y — Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with position.

check_overlap — If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label().

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

design — Either "2025" (default), which sets family = "Roboto", or "legacy", which sets family = "Source Sans 3" to match the pre-2025 theme.

fill — Fill color for label box. Defaults to dark gray.

color — Font color. Defaults to white.

fontface — Font emphasis. Defaults to bold.

label.padding — Amount of padding around label. Defaults to 0.25 lines.

label.r — Radius of rounded corners. Defaults to 0.15 lines.

label.size — Size of label border, in mm.

Details

Note that when you resize a plot, text labels stay the same size, even though the size of the plot area changes. This happens because the "width" and "height" of a text element are 0. Obviously, text labels do have height and width, but they are physical units, not data units. For the same reason, stacking and dodging text will not work by default, and axis limits are not automatically expanded to include all text.

`geom_text()` and `geom_label()` add labels for each row in the data, even if coordinates x, y are set to single values in the call to `geom_label()` or `geom_text()`. To add labels at specified points use `annotate()` with `annotate(geom = "text", ...)` or `annotate(geom = "label", ...)`.

To automatically position non-overlapping text labels see the ggrepel package.

Value

A ggplot2 layer that can be added to a `ggplot2::ggplot()` plot.

Examples

```
library(ggplot2)

p <- ggplot(
  data = mtcars,
  mapping = aes(
    x = mpg,
    y = mpg,
    label = rownames(mtcars)
  )
)

# downloading and rendering with the HDX fonts is slow, so it's wrapped in
# \donttest{} rather than run on every check

try(load_hdx_fonts())
try(p + geom_text_hdx())
try(p + geom_label_hdx())

# match the pre-2025 legacy theme instead
try(load_source_sans_3())
try(p + geom_text_hdx(design = "legacy"))
```

---

gghdx *Set HDX theme and aesthetics*

Description

gghdx() gives you the convenience of theme_hdx() without having to explicitly call it for each plot. It also allows for setting the default continuous and discrete scales to follow the HDX color scheme, including default line and point colors and area fills. gghdx_reset() returns all of these values back to the defaults.

Usage

```
gghdx(
  showtext = TRUE,
  base_size = 10,
  base_family = NULL,
  horizontal = TRUE,
  title_family = NULL,
  design = c("2025", "legacy")
)

gghdx_reset()
```

Arguments

showtext	logical If TRUE, uses the showtext package to add the Roboto and Merriweather fonts (or, when design = "legacy", Source Sans 3) and runs showtext_auto() so all future plots in this session will use them.
base_size	base font size, given in pts.
base_family	Base font family for body text, such as axis text and legend text. Defaults to "Roboto" when design = "2025", or "Source Sans 3" when design = "legacy".
horizontal	logical Horizontal axis lines?
title_family	Font family for titles, subtitles, and strip text when design = "2025". Defaults to "Merriweather", unless base_family is also supplied, in which case it defaults to base_family. Ignored when design = "legacy", which always uses a single font family.
design	Either "2025" (default), the current HDX visual design, or "legacy", the original pre-2025 theme kept for backwards compatibility.

Details

gghdx() changes global settings for this R session. This includes updating the ggplot2 default geometries using ggplot2::update_geom_defaults() and setting global options to scale color and fill for ggplot2:

- options("ggplot2.discrete.fill")
- options("ggplot2.discrete.colour")
- options("ggplot2.continuous.fill")
- options("ggplot2.continous.colour")

The default discrete scale is `scale_..._hdx()` for both `fill` and `color`. For continuous scales, the default is `scale_fill_gradient_hdx_primary()` for fill and `scale_color_gradient_hdx_primary()` for color. Passing `design = "legacy"` uses the pre-2025 equivalents instead (`scale_fill_gradient_hdx_mint()` and `scale_color_gradient_hdx_sapphire()` for continuous scales), so a report that isn't ready to move to the new look can call `gghdx(design = "legacy")` to keep rendering as it did in gghdx 0.1.4.

Once `gghdx()` is run, the easiest way to return to the default ggplot2 settings is to run `gghdx_reset()`. This will make changes by running:

- `ggplot2::reset_theme_settings()`: resets the global theme to default.
- For all of the options listed above, run `options("option") <- NULL`.
- `showtext::showtext_end()` to stop using the showtext library if it was activated.
- Runs `ggplot2::update_geom_defaults()` for all geometries in `ggplot2_geom_defaults()`.

You can also simply restart your R session to return to the defaults.

Value

No return value, run for the side effects described in Details.

See Also

`gghdx()` relies on the following functions:

- `theme_hdx()` as the default theme.
- `load_hdx_fonts()` to load the fonts and activate showtext.
- `hdx_geom_defaults()` as the default geometries to set with `ggplot2::update_geom_defaults()`.
- `scale_color_hdx_discrete()` and other family of functions to set standard fill and color scales.

Examples

```
library(ggplot2)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  ) +
  labs(
    x = "Miles per gallon",
    y = "Horsepower",
    title = "Horsepower relative to miles per gallon"
  )

# see the plot using base aesthetics
p
```

```
# downloading the HDX fonts is slow, so it's wrapped in \donttest{} rather
# than run on every check

# automatically use the gghdx theme and visuals
try(gghdx())
p

# get rid of the changes of gghdx
gghdx_reset()
p

# keep the pre-2025 look instead
try(gghdx(design = "legacy"))
p
gghdx_reset()
```

`ggplot2_geom_defaults` *Default ggplot2 geometry aesthetics*

Description

Default geometry aesthetics from the ggplot2 library. All of the aesthetics are the standard ggplot2 defaults for those changed in `gghdx()` based on `hdx_geom_defaults()`. Used in `gghdx_reset()` to return the plotting defaults back to normal.

Usage

```
ggplot2_geom_defaults()
```

Details

These aesthetics were manually pulled from ggplot2 from the geometries' `default_aes` information, such as `ggplot2::GeomPoint$default_aes`. Since the `default_aes` is changed after `gghdx()` is run, the default geometries in this function are hardcoded.

Value

A list of geometry defaults.

Examples

```
library(purrr)
library(ggplot2)

# updating geom defaults (like default color of a point or fill for bar)
purrr::walk(
```

```
  hdx_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .),
)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  )

# see the points are automatically in HDX primary blue
p

# need to reset back to the default geometries
purrr::walk(
  ggplot2_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .)
)

# now the points are back to default black
p
```

`hdx_colors` *Hex values for HDX colors*

Description

`hdx_colors()` conveniently returns a vector of hex values for specified color ramps. Full values can be found in hdx_color_list. If you know the name of the color you want, such as "sapphire-hdx", you can use `hdx_hex(c("sapphire-hdx"))` to directly access the hex code.

Usage

```
hdx_colors(colors = c("sapphire", "mint", "tomato", "gray"))

hdx_colours(colors = c("sapphire", "mint", "tomato", "gray"))

hdx_hex(color_names)

hdx_color_names()

hdx_colour_names()
```

Arguments

colors	Specified color ramps to return. Some set of "sapphire", "mint", "tomato", "gray", "primary", "brand", "neutral", "success", "warning", and "error". By default returns the original "sapphire", "mint", "tomato", and "gray" colors.
color_names	Vector of color names. Valid values are all available using hdx_colors

Details

All valid color names are in the named vector returned by hdx_colors() or accessible in the convenient hdx_color_names().

As of the 2025 HDX website redesign, the "primary", "brand", "neutral", "success", "warning", and "error" scales are also available, following the step naming of the HDX design tokens (e.g. "primary-5" for the primary scale's base blue, "neutral-2" for a hairline gridline gray). These are not returned by default to avoid breaking existing uses of hdx_colors(); request them explicitly, e.g. hdx_colors("primary"). The original "sapphire", "mint", and "tomato" scales are kept for backwards compatibility but are superseded by "primary"/"brand"/"error" going forward.

Value

- hdx_colors() returns a named vector of hex values.
- hdx_color_names() returns a character vector of color names.

See Also

Other color hdx: hdx_color_list, hdx_pal_discrete()

Examples

```
# get hex values
hdx_colors()
hdx_colors("sapphire")

# 2025 redesign colors
hdx_colors("primary")

# get color names
hdx_color_names()
```

---

hdx_color_list *HDX color ramps*

Description

List of color ramps from the HDX visual identity. These are mint, sapphire, tomato, and grays.

Usage

```
hdx_color_list
```

Format

A list with 4 data frames:

Source

https://data.humdata.org/dataviz-guide/visual-identity/#/visual-identity/colours/

See Also

Other color hdx: `hdx_colors()`, `hdx_pal_discrete()`

---

hdx_display_pal *Display HDX palette*

Description

Displays the HDX color palettes. By default, shows all values for all palettes. You can change the number of values for each palette or only show a subset of the available palettes (from `hdx_pal_...()`).

Usage

```
hdx_display_pal(
  n = NULL,
 palette = c("discrete", "primary", "brand", "error", "gray", "mint", "sapphire",
    "tomato")
)
```

Arguments

`n`	Number of colors for each palette to show.
`palette`	Character vector of palettes to show.

Value

Plot of HDX color palettes.

Examples

```
hdx_display_pal()
hdx_display_pal(n = 3)
```

hdx_geom_defaults *Default HDX geometry aesthetics*

Description

Default geometry aesthetics fitting the HDX design guide. Used in `gghdx()` to set default fill, color, size, and point geometry defaults, which is not possible using just `theme_hdx()`.

Usage

```
hdx_geom_defaults(design = c("2025", "legacy"))
```

Arguments

`design`	Either `"2025"` (default), the current HDX visual design, or `"legacy"`, the original pre-2025 defaults kept for backwards compatibility.

Details

Derived from the ggthemr methods.

As of the 2025 HDX website redesign, the defaults use the new primary blue and institutional datagrid blue. Pass `design = "legacy"` to instead get the original pre-2025 defaults (HDX sapphire and mint) for reports that aren't ready to move to the new look.

Value

A list of geometry defaults.

See Also

- `gghdx()` for automatically setting default geometries, along with other styling.
- `ggplot2_geom_defaults()` for the ggplot2 default aesthetics.

Examples

```
library(purrr)
library(ggplot2)

# updating geom defaults (like default color of a point or fill for bar)
purrr::walk(
  hdx_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .),
)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
```

```
      y = hp
    )
  )

# see the points are automatically in HDX primary blue
p

# need to reset back to the default geometries
purrr::walk(
  ggplot2_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .)
)

# now the points are back to default black
p
```

`hdx_pal_discrete` *HDX color palette (discrete)*

Description

As of the 2025 HDX website redesign, `hdx_pal_discrete()` utilizes the primary, brand, and error hues for up to a 12 element discrete scale. These supersede the original sapphire, mint, and tomato hues, kept available individually via `hdx_pal_sapphire()`, `hdx_pal_mint()`, and `hdx_pal_tomato()` for backwards compatibility. Pass `design = "legacy"` to instead get the original sapphire/mint/tomato 12 element scale for reports that aren't ready to move to the new look.

Usage

```
hdx_pal_discrete(design = c("2025", "legacy"))

hdx_pal_sapphire()

hdx_pal_primary()

hdx_pal_brand()

hdx_pal_error()

hdx_pal_tomato()

hdx_pal_mint()

hdx_pal_gray()
```

Arguments

design	Either "2025" (default), the current HDX visual design, or "legacy", the original pre-2025 12 element scale kept for backwards compatibility. Only used by hdx_pal_discrete().

Details

hdx_pal_primary(), hdx_pal_brand(), and hdx_pal_error() allow for a 4 element discrete scale using only the specified color. These are color ramps with a range from dark, normal (HDX standard), light, and ultra light. hdx_pal_mint(), hdx_pal_tomato(), and hdx_pal_sapphire() provide the same 4 element ramps for the original palette.

Value

A palette function.

See Also

Other color hdx: hdx_color_list, hdx_colors()

Examples

```
hist(mtcars$mpg, col = hdx_pal_discrete()(5))
```

---

label_number_hdx *Format and labels numbers in HDX key figures style*

Description

Use format_number_hdx() to directly format numeric vectors, and use label_number_hdx() in the same way as the scales:: family of label functions. The return value of label_number_hdx() is a function, based on the additional_prefix. So you should pass it in to scales_...() labels parameter in the same way as scales_...()

Usage

```
label_number_hdx(additional_prefix = "")

format_number_hdx(x, additional_prefix = "")
```

Arguments

additional_prefix	Additional prefix to add to string, that will come between sign_prefix and the number. For example, "$" could produce a return value of -$1.1K.
x	Numeric vector to format

Details

Numeric vectors are formatted in the HDX style for key figures, which abbreviates numbers 1,000 and above to X.YK, 10,000 and above to XYK, 100,000 and above to XYZK, and the same for 1,000,000 and above, replacing the K with an M, and the same for B. Details of the data viz style can be found in the visualization guidelines

Just for continuity, values are labeled with T for trillion, and that is the maximum formatting available, anything above the trillions will continue to be truncated to report in the trillions.

Deals with negative values in case those ever need to be formatted in similar manners. Also ensures that rounding is performed so numbers look correct. Not to be used for percents, which should just use `scales::label_percent()`.

Value

`label_number_hdx()`: "labelling" function, in the same way as `scales::label_...()` functions work, i.e. a function that takes `x` and returns a labelled character vector of `length(x)`.

`format_number_hdx()`: Formatted character vector of number strings

Examples

```
# label_number_hdx()

library(ggplot2)

# discrete scaling
p <- ggplot(txhousing) +
  geom_point(
    aes(
      x = median,
      y = volume
    )
  )

p

p +
  scale_x_continuous(
    labels = label_number_hdx("$")
  ) +
  scale_y_continuous(
    labels = label_number_hdx()
  )

# number_hdx()

x <- c(1234, 7654321)
format_number_hdx(x)
format_number_hdx(x, "$")
```

load_hdx_fonts *Load and use the HDX 2025 redesign fonts*

Description

Simple wrapper for `sysfonts::font_add_google()` and `showtext::showtext_auto()` to load the two fonts used in the HDX website's 2025 redesign, Merriweather for display and heading text, and Roboto for body text, and specify all plots to automatically use showtext. Use to load the default font families for `theme_hdx()` and `gghdx()`.

Usage

```
load_hdx_fonts(
  display_family = NULL,
  display_regular = NULL,
  body_family = NULL,
  body_regular = NULL
)
```

Arguments

`display_family`	Character string for the Merriweather family name to register, used whether the font is downloaded from Google or loaded locally. If `NULL`, defaults to `"Merriweather"`, the standard family name. See "Details" in the `sysfonts::font_add()` documentation for further explanation.
`display_regular`	Path to the font file for the Merriweather regular font face. If `NULL`, defaults to `"Merriweather-Regular.ttf"`, the standard file name downloaded from Merriweather. Used only when no internet connection is available to directly load from Google.
`body_family`	Character string for the Roboto family name to register, used whether the font is downloaded from Google or loaded locally. If `NULL`, defaults to `"Roboto"`, the standard family name.
`body_regular`	Path to the font file for the Roboto regular font face. If `NULL`, defaults to `"Roboto-Regular.ttf"`, the standard file name downloaded from Roboto. Used only when no internet connection is available to directly load from Google.

Details

By default, both fonts are loaded from Google using `sysfonts::font_add_google()`. If an internet connection is unavailable, then attempts to use locally installed versions of the fonts using `sysfonts::font_add(family, regular)`. If you have the fonts installed but still receive an error from this function, check the `*_family` and `*_regular` arguments match your installed fonts.

Value

Nothing, run for side effect of loading the fonts and activating showtext.

See Also

gghdx() for automatically running load_hdx_fonts(), along with other styling. load_source_sans_3() to instead load the original HDX font, kept for backwards compatibility.

Examples

```
library(ggplot2)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  ) +
  labs(
    x = "Miles per gallon",
    y = "Horsepower",
    title = "Horsepower relative to miles per gallon"
  )

# font not loaded so an error will be generated
try(p + theme_hdx())

try(load_hdx_fonts())

try(p + theme_hdx())
```

---

load_source_sans_3 *Load and use Source Sans 3*

Description

Simple wrapper for sysfonts::font_add_google() and showtext::showtext_auto() to load the Source Sans 3 font and specify all plots to automatically use showtext. Use to load the default font family for geom_text_hdx() and geom_label_hdx().

Usage

```
load_source_sans_3(family = NULL, regular = NULL)
```

Arguments

family	Character string for the Source Sans 3 family. If NULL, defaults to "Source Sans 3", the standard family name. See "Details" in the sysfonts::font_add() documentation for further explanation. Used only when no internet connection is available to directly load from Google.

`regular` Path to the font file for the regular font face. If `NULL`, defaults to `"SourceSans3-Regular.ttf"`, the standard file name downloaded from Source Sans 3. Used only when no internet connection is available to directly load from Google.

Details

By default, the font is loaded from Google using `sysfonts::font_add_google()`. If an internet connection is unavailable, then attempts to use a locally installed version of the font using `sysfonts::font_add(family, regular)`. If you have the font installed but still receive an error from this function, check the `family` and `regular` arguments match your installed font.

Value

Nothing, run for side effect of loading the font and activating showtext.

See Also

`gghdx()` for automatically running `load_source_sans_3()`, along with other styling.

Examples

```
library(ggplot2)
p <- ggplot(
  data = mtcars,
  mapping = aes(
    x = mpg,
    y = mpg,
    label = rownames(mtcars)
  )
)

# font not loaded so error will be generated
try(p + geom_label_hdx())

try(load_source_sans_3())

try(p + geom_label_hdx())
```

---

scale_color_hdx_discrete

HDX color scales

Description

Color scales using the HDX palette. For discrete color scales, the `scale_color_hdx_...()` and `scale_fill_hdx_...()` family of functions are available. For gradient scales, use `scale_color_gradient_hdx()` and `scale_fill_gradient_hdx()` functions for a single color scale or `scale_..._gradient2...()` alternative.

Usage

```
scale_color_hdx_discrete(na.value = NULL, ..., design = c("2025", "legacy"))

scale_colour_hdx_discrete(na.value = NULL, ..., design = c("2025", "legacy"))

scale_color_hdx_gray(na.value = hdx_hex("tomato-hdx"), ...)

scale_colour_hdx_gray(na.value = hdx_hex("tomato-hdx"), ...)

scale_colour_hdx_grey(na.value = hdx_hex("tomato-hdx"), ...)

scale_color_hdx_grey(na.value = hdx_hex("tomato-hdx"), ...)

scale_color_hdx_mint(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_mint(na.value = hdx_hex("gray-light"), ...)

scale_color_hdx_sapphire(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_sapphire(na.value = hdx_hex("gray-light"), ...)

scale_color_hdx_primary(na.value = hdx_hex("neutral-2"), ...)

scale_colour_hdx_primary(na.value = hdx_hex("neutral-2"), ...)

scale_color_hdx_brand(na.value = hdx_hex("neutral-2"), ...)

scale_colour_hdx_brand(na.value = hdx_hex("neutral-2"), ...)

scale_color_hdx_error(na.value = hdx_hex("neutral-2"), ...)

scale_colour_hdx_error(na.value = hdx_hex("neutral-2"), ...)

scale_color_hdx_tomato(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_tomato(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_discrete(na.value = NULL, ..., design = c("2025", "legacy"))

scale_fill_hdx_gray(na.value = hdx_hex("tomato-hdx"), ...)

scale_fill_hdx_grey(na.value = hdx_hex("tomato-hdx"), ...)

scale_fill_hdx_mint(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_sapphire(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_primary(na.value = hdx_hex("neutral-2"), ...)
```

```
scale_fill_hdx_brand(na.value = hdx_hex("neutral-2"), ...)

scale_fill_hdx_error(na.value = hdx_hex("neutral-2"), ...)

scale_fill_hdx_tomato(na.value = hdx_hex("gray-light"), ...)

scale_fill_gradient_hdx(na.value = "transparent", ...)

scale_fill_gradient_hdx_sapphire(na.value = "transparent", ...)

scale_fill_gradient_hdx_mint(na.value = "transparent", ...)

scale_fill_gradient_hdx_tomato(na.value = "transparent", ...)

scale_fill_gradient_hdx_primary(na.value = "transparent", ...)

scale_fill_gradient_hdx_brand(na.value = "transparent", ...)

scale_fill_gradient_hdx_error(na.value = "transparent", ...)

scale_color_gradient_hdx(na.value = "transparent", ...)

scale_colour_gradient_hdx(na.value = "transparent", ...)

scale_color_gradient_hdx_sapphire(na.value = "transparent", ...)

scale_colour_gradient_hdx_sapphire(na.value = "transparent", ...)

scale_color_gradient_hdx_mint(na.value = "transparent", ...)

scale_colour_gradient_hdx_mint(na.value = "transparent", ...)

scale_color_gradient_hdx_tomato(na.value = "transparent", ...)

scale_colour_gradient_hdx_tomato(na.value = "transparent", ...)

scale_color_gradient_hdx_primary(na.value = "transparent", ...)

scale_colour_gradient_hdx_primary(na.value = "transparent", ...)

scale_color_gradient_hdx_brand(na.value = "transparent", ...)

scale_colour_gradient_hdx_brand(na.value = "transparent", ...)

scale_color_gradient_hdx_error(na.value = "transparent", ...)

scale_colour_gradient_hdx_error(na.value = "transparent", ...)
```

```
scale_color_gradient2_hdx(na.value = "transparent", ...)

scale_colour_gradient2_hdx(na.value = "transparent", ...)

scale_fill_gradient2_hdx(na.value = "transparent", ...)
```

Arguments

na.value	Colour to use for missing values
...	Arguments passed on to discrete_scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take (e.g., scales::pal_hue()).

breaks One of:

- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

limits One of:

- NULL to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE includes the levels in the factor. Please note that to display every level in a legend, the layer should use show.legend = TRUE.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.

labels One of:

- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

guide A function used to create a guide or its name. See guides() for more information.

call The call used to construct the scale for reporting messages.

super The super class to use for the constructed scale

design — Either "2025" (default), the current HDX visual design, or "legacy", the original pre-2025 discrete scale kept for backwards compatibility. Only used by scale_color_hdx_discrete() and scale_fill_hdx_discrete(); see hdx_pal_discrete().

Value

Relevant ggplot2 scale object to add to a ggplot2::ggplot() plot, either ggplot2::ScaleDiscrete or ggplot2::ScaleContinuous.

See Also

gghdx() for setting default fill and color scaling, along with other styling.

Examples

```
library(ggplot2)

# discrete scaling
p1 <- ggplot(iris) +
  geom_point(
    aes(
      x = Sepal.Length,
      y = Petal.Width,
      color = Species
    )
  )

p1

p1 + scale_color_hdx_discrete()
p1 + scale_color_hdx_mint()

# use gradient scaling
p2 <- ggplot(iris) +
  geom_point(
    aes(
      x = Sepal.Length,
      y = Petal.Width,
      color = Petal.Length
    )
  )

p2

p2 + scale_color_gradient_hdx_mint()
p2 + scale_color_gradient_hdx_tomato()
```

scale_y_continuous_hdx *Position scales for continuous y data*

Description

`scale_y_continuous_hdx()` and the three variants with different `trans` arguments are defaults scales for the y axis that ensures the distance from data to the y-axis is reduced to 0, as is common throughout the HDX data visualization guidelines. This is done by setting `expand = c(0, 0)`.

Usage

```
scale_y_continuous_hdx(...)

scale_y_log10_hdx(...)

scale_y_reverse_hdx(...)

scale_y_sqrt_hdx(...)
```

Arguments

`...`	Other arguments pass on to `ggplot2::scale_y_continuous()`.

Details

For simple manipulation of labels and limits, you may wish to use `labs()` and `lims()` instead.

Value

`ggplot2::ScaleContinuousPosition` object to scale a `ggplot2::ggplot()` plot.

Examples

```
library(ggplot2)

p <- ggplot(df_covid) +
  geom_line(
    aes(
      x = date,
      y = cases_monthly
    )
  )

p

# start y axis at 0
p + scale_y_continuous_hdx()
p + scale_y_log10_hdx()
```

theme_hdx *ggplot color theme based on HDX visual design guide*

Description

A theme that approximates the style of the *Humanitarian Data Exchange (HDX)*.

Usage

```
theme_hdx(
  base_size = 10,
  base_family = NULL,
  horizontal = TRUE,
  title_family = NULL,
  design = c("2025", "legacy")
)
```

Arguments

base_size	base font size, given in pts.
base_family	Base font family for body text, such as axis text and legend text. Defaults to "Roboto" when design = "2025", or "Source Sans 3" when design = "legacy".
horizontal	logical Horizontal axis lines?
title_family	Font family for titles, subtitles, and strip text when design = "2025". Defaults to "Merriweather", unless base_family is also supplied, in which case it defaults to base_family. Ignored when design = "legacy", which always uses a single font family.
design	Either "2025" (default), the current HDX visual design, or "legacy", the original pre-2025 theme kept for backwards compatibility.

Details

theme_hdx() implements a chart that follows the general visual guide of the HDX platform, as defined in the dataviz-guide.

Use scale_color_hdx_discrete() with this theme.

As of the 2025 HDX website redesign, theme_hdx() defaults to the new typography: the free Google font Merriweather for titles and other display text (title_family), and the free Google font Roboto for body text (base_family). Use the **sysfonts** package to add the Google fonts easily, or use load_hdx_fonts() to load both at once. Pass design = "legacy" to instead get the pre-2025 theme (single Source Sans 3 font, original gray palette, no axis ticks) for reports that aren't ready to move to the new look.

If you supply base_family but not title_family, title_family is set to match base_family rather than defaulting to "Merriweather". This means code written for the pre-2025 single-font theme, such as theme_hdx(base_family = "Source Sans 3"), continues to work without requiring Merriweather to also be loaded.

Value

A `theme()` to stylize a `ggplot2::ggplot()` plot.

References

- Humanitarian Data Exchange
- Google Fonts, Merriweather
- Google Fonts, Roboto
- HDX Dataviz Guide

See Also

`gghdx()` for automatically applying the theme to all plots in this current R session, along with other styling.

Examples

```
library(ggplot2)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  ) +
  labs(
    x = "Miles per gallon",
    y = "Horsepower",
    title = "Horsepower relative to miles per gallon"
  )

# the default fonts are Roboto (body) and Merriweather (titles)
# an error will occur if not loaded before using theme_hdx()
try(p + theme_hdx())

# you can change the base and title families
p + theme_hdx(base_family = "sans", title_family = "sans")

# supplying only base_family carries it over to title_family too
try(load_source_sans_3())
try(p + theme_hdx(base_family = "Source Sans 3"))

# or load Roboto and Merriweather using gghdx() or load_hdx_fonts()
try(load_hdx_fonts())
try(p + theme_hdx())

# we can change the axis line direction depending on the plot
p + theme_hdx(horizontal = FALSE)
```

```
# use the pre-2025 theme instead
p + theme_hdx(design = "legacy")
```

Index