Package 'legendry'

November 1, 2024

Title Extended Legends and Axes for 'ggplot2'

Version 0.1.0

Description A 'ggplot2' extension that focusses on expanding the plotter's arsenal of guides. Guides in 'ggplot2' include axes and legends. 'legendry' offers new axes and annotation options, as well as new legends and colour displays.

License MIT + file LICENSE

URL https://teunbrand.github.io/legendry/, https://github.com/teunbrand/legendry

BugReports https://github.com/teunbrand/legendry/issues

Depends ggplot2 (>= 3.5.0), R (>= 4.1.0)

Imports cli, grid (>= 4.1.0), gtable, lifecycle, rlang (>= 1.1.0), scales (>= 1.1.1), vctrs (>= 0.6.0)

Suggests knitr, ragg, rmarkdown, svglite, systemfonts, testthat (>= 3.0.0), vdiffr, withr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Teun van den Brand [aut, cre, cph] (<https://orcid.org/0000-0002-9335-7468>)

Maintainer Teun van den Brand <tahvdbrand@gmail.com>

Repository CRAN

Date/Publication 2024-11-01 14:30:06 UTC

Contents

`bracket_options` *Bracket options*

Description

These functions construct various sorts of brackets. They construct a matrix that can be supplied as the `bracket` argument in `primitive_bracket()`.

Usage

```
bracket_line()

bracket_square()

bracket_chevron()

bracket_round(angle = 180, n = 100)

bracket_sigmoid(curvature = 10, n = 100)

bracket_atan(curvature = 5, n = 100)

bracket_curvy(angle = 225, n = 100)
```

Arguments

angle	A numeric(1): the angle in degrees for which a circle piece is drawn. For bracket_curvy(), an angle between 180 and 270.
n	An integer(1) number of points to use for the bracket.
curvature	A numeric(1) that controls the curliness of the bracket. More precisely, it is used to construct the sequence seq(-curvature, curvature, length.out = n) over which the logistic or arctangent functions is evaluated.

Details

When designing custom bracket shapes, the expectation is both columns are are a number between 0 and 1. The first column follows the direction of the guide whereas the second column is orthogonal to that direction.

Value

A <matrix[n, 2]> with coordinates for points on the brackets.

Functions

- bracket_line(): A simple line as bracket. Has n = 2 points.
- bracket_square(): A square bracket. Has n = 4 points.
- bracket_chevron(): A chevron (V-shape) that makes a bracket. Has n = 3 points.
- bracket_round(): One circular arc that makes a bracket.
- bracket_sigmoid(): Two sigmoid curves stacked on top of one another to form a bracket.
- bracket_atan(): Two arctangent curves stacked on top of one another to form a bracket.
- bracket_curvy(): Four circular arcs that make a bracket.

Examples

```
plot(bracket_sigmoid(), type = 'l')
```

cap_options *Cap options*

Description

These functions construct various sorts of caps. They construct a matrix that can be supplied as the shape argument in gizmo_barcap().

Usage

```
cap_triangle()

cap_round(n = 100)

cap_arch(n = 100)

cap_ogee(n = 100)

cap_none()
```

Arguments

n	An <integer[n]> number of points to use for the cap.

Details

When designing custom cap shapes, the expectation is that the first point starts at the (0, 0) coordinate and the last point ends at the (0, 1) coordinate. The first column follows the orthogonal direction of the bar whereas the second column follows the direction of the bar.

Value

A <matrix[n, 2]> with coordinates for points on the brackets.

Functions

- cap_triangle(): An equilateral triangle with n = 3 points.
- cap_round(): A semicircle.
- cap_arch(): Two circular arcs forming an equilateral Gothic arch.
- cap_ogee(): Four circular arcs forming an 'ogee' arch.
- cap_none(): No cap.

Examples

```
plot(cap_arch(), type = 'l')
```

compose_crux *Compose guides in a cross*

Description

[Experimental]

This guide composition has a central guide optionally surrounded by other guides on all four sides.

Usage

```
compose_crux(
  key = NULL,
  centre = "none",
  left = "none",
  right = "none",
  top = "none",
  bottom = "none",
  args = list(),
  complete = FALSE,
  theme = NULL,
  theme_defaults = list(),
  reverse = FALSE,
  order = 0,
  title = waiver(),
  position = waiver(),
  available_aes = NULL
)
```

Arguments

key	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
centre, left, right, top, bottom	Guides to use in composition per position. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character> naming such a function, without the guide_ or primitive_ prefix.
args	A <list> of arguments to pass to guides that are given either as a function or as a string.
complete	A <logical[1]> whether to treat the composition as a complete guide. If TRUE, a title and margin are added to the result. If FALSE (default), no titles and margins are added.

theme	A <theme> object to style the guide individually of differently from the plot's theme settings. The theme arguments in the guide overrides, and is combined with, the plot's theme.
theme_defaults	A <list> of theme elements to override undeclared theme arguments.
reverse	A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.
order	A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.
title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
position	Where this guide should be drawn: one of "top", "bottom", "left", or "right".
available_aes	A <character> vector listing the aesthetics for which this guide can be build.

Value

A <ComposeCrux> guide object.

See Also

Other composition: compose_ontop(), compose_sandwich(), compose_stack(), guide-composition

Examples

```
# Roughly recreating a colour bar with extra text on top and bottom
crux <- compose_crux(
  centre = gizmo_barcap(), left = "axis_base",
  right = "axis_base",
  top = primitive_title("A lot"),
  bottom = primitive_title("A little")
)

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = cty)) +
  guides(colour = crux)
```

---

compose_ontop *Compose guides on top of one another*

Description

[Experimental]

This guide can place place other guides on top of one another.

Usage

```
compose_ontop(
  ...,
  args = list(),
  key = NULL,
  title = waiver(),
  angle = waiver(),
  theme = NULL,
  order = 0,
  position = waiver(),
  available_aes = NULL
)
```

Arguments

`...`	Guides to stack in composition. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character[1]> naming such a function, without the guide_ or primitive_ prefix.
`args`	A <list> of arguments to pass to guides that are given either as a function or as a string.
`key`	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
`title`	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
`angle`	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
`theme`	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
`order`	A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.
`position`	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
`available_aes`	A <character> giving aesthetics that must match the the guides.

Value

A <ComposeOntop> composite guide object.

See Also

Other composition: compose_crux(), compose_sandwich(), compose_stack(), guide-composition

Examples

```
# Using the ontop composition to get two types of ticks with different
# lengths
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  guides(x = compose_ontop(
    guide_axis_base(
      key_manual(c(2, 4, 6)),
      theme = theme(
        axis.ticks = element_line(colour = "limegreen"),
        axis.ticks.length = unit(11, "pt")
      )
    ),
    guide_axis_base(
      key_manual(c(3, 5, 7)),
      theme = theme(
        axis.ticks = element_line(colour = "tomato"),
        axis.ticks.length = unit(5.5, "pt")
      )
    )
  ))
```

---

compose_sandwich *Compose guides as a sandwich*

Description

[Experimental]

This guide composition has a middle guide flanked by two parallel guides.

Usage

```
compose_sandwich(
  key = key_auto(),
  middle = gizmo_barcap(),
  text = "none",
  opposite = "none",
  args = list(),
  complete = TRUE,
  theme = NULL,
```

```
  theme_defaults = list(),
  reverse = FALSE,
  order = 0,
  title = waiver(),
  position = waiver(),
  available_aes = NULL
)
```

Arguments

key	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
middle	Guide to use as the middle guide. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character> naming such a function, without the guide_ or primitive_ prefix.
text, opposite	Guides to use at the legend.text.position location and on the opposite side of the middle guide respectively. Guide specification is the same as in the middle argument.
args	A <list> of arguments to pass to guides that are given either as a function or as a string.
complete	A <logical[1]> whether to treat the composition as a complete guide. If TRUE, a title and margin are added to the result. If FALSE (default), no titles and margins are added.
theme	A <theme> object to style the guide individually of differently from the plot's theme settings. The theme arguments in the guide overrides, and is combined with, the plot's theme.
theme_defaults	A <list> of theme elements to override undeclared theme arguments.
reverse	A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.
order	A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.
title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
position	Where this guide should be drawn: one of "top", "bottom", "left", or "right".
available_aes	A <character> vector listing the aesthetics for which this guide can be build.

Details

The sandwich composition is effectively the same as a crux composition lacking two opposing arms.

Value

A <ComposeSandwich> guide object.

See Also

Other composition: compose_crux(), compose_ontop(), compose_stack(), guide-composition

Examples

```
# A standard plot with a sandwich guide
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = cty)) +
  guides(colour = compose_sandwich(
    middle = "colourbar",
    text = "axis_base",
    opposite = primitive_bracket(key = key_range_manual(
      start = c(10, 20), end = c(25, 30), name = c("A", "B")
    ))
  ))
```

---

compose_stack *Compose guides as stack*

Description

[Experimental]

This guide can stack other guides.

Usage

```
compose_stack(
  ...,
  args = list(),
  key = NULL,
  title = waiver(),
  side.titles = waiver(),
  angle = waiver(),
  theme = NULL,
  order = 0,
  drop = NULL,
  position = waiver(),
  available_aes = NULL
)
```

Arguments

...	Guides to stack in composition. Each guide can be specified as one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character[1]> naming such a function, without the guide_ or primitive_ prefix.
args	A <list> of arguments to pass to guides that are given either as a function or as a string.
key	A standard key specification. The key is shared among all guides that have NULL keys themselves. See more information in the linked topic.
title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
side.titles	A <character> giving labels for titles displayed on the side of the stack. Set to NULL to display no side titles. If waiver(), an attempt is made to extract the titles from the guides and use these as side titles.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
order	A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.
drop	An <integer> giving the indices of guides that should be dropped when a facet requests no labels to be drawn at axes in between panels. The default, NULL, will drop every guide except the first.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
available_aes	A <character> giving aesthetics that must match the the guides.

Value

A <ComposeStack> guide object.

See Also

Other composition: compose_crux(), compose_ontop(), compose_sandwich(), guide-composition

Examples

```
ggplot() +
  geom_function(fun = dnorm, xlim = c(-3, 3)) +
  guides(x = compose_stack(
    "axis", "axis",
    side.titles = c("first", "second")
  )) +
  # Add margin to make room for side titles
  theme(plot.margin = margin(5.5, 5.5, 5.5, 11))
```

`gizmo_barcap` *Guide gizmo: capped colour bar*

Description

This guide displays a colour bar with optional caps at either ends of the bar.

Usage

```
gizmo_barcap(
  key = "sequence",
  shape = "triangle",
  size = NULL,
  show = NA,
  alpha = NA,
  oob = "keep",
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

key	A sequence key specification. Defaults to `key_sequence(n = 15)`. Changing the argument to `key_sequence()` is fine, but changing the key type is not advised.
shape	A cap specification by providing one of the following: • A cap `<function>`, such as `cap_triangle()`. • A `<character[1]>` naming a cap function without the 'cap_'-prefix, e.g. `"round"`. • A two column `<matrix[n, 2]>` giving coordinates for a cap, like those created by cap functions such as `cap_arch()`.
size	A `<unit>` setting the size of the cap. When `NULL` (default), cap size will be proportional to the `shape` coordinates and the `legend.key.size` theme setting.

show — A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

direction — A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Value

A <GizmoBarcap> object.

See Also

Other gizmos: gizmo_density(), gizmo_grob(), gizmo_histogram(), gizmo_stepcap()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point()

# Just a bar
p + scale_colour_viridis_c(guide = gizmo_barcap())

# Caps show up when there is data outside the limits
p + scale_colour_viridis_c(
  limits = c(10, 30),
  guide  = gizmo_barcap()
)

# The scale's out-of-bounds handler determines cap colour
p + scale_colour_viridis_c(
  limits = c(10, 30), oob = scales::oob_squish,
  guide = gizmo_barcap()
)
```

```
# Customising display of the guide
p +
  scale_colour_viridis_c(
    oob = scales::oob_squish,
    guide = gizmo_barcap(
      shape = "arch", show = c(FALSE, TRUE),
      size = unit(2, "cm"),
      theme = theme(legend.key.height = unit(4, "cm"))
    )
  ) +
  theme(
    legend.frame = element_rect(colour = "black"),
    legend.key.width = unit(0.5, "cm")
  )
```

`gizmo_density` *Guide gizmo: kernel density estimate*

Description

This guide displays a kernel density estimate (KDE) of the aesthetic. If the aesthetic is `colour` or `fill`, the shape will reflect this.

Usage

```
gizmo_density(
  key = "sequence",
  density = NULL,
  density.args = list(),
  density.fun = stats::density,
  just = 0.5,
  oob = "keep",
  alpha = NA,
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

`key`	A sequence key or binned key specification.
`density`	One of the following: • `NULL` for using kernel density estimation on the data values (default). • a `<numeric>` vector to feed to the `density.fun` function. • A named `<list>` with `x` and `y` numeric elements of equal length, such as one returned by using the `density()` function. Please note that `<list>` input is expected in scale-transformed space, not original data space.

density.args	A <list> with additional arguments to the density.fun argument. Only applies when density is not provided as a <list>. already.
density.fun	A <function> to use for kernel density estimation when the density argument is not provided as a list already.
just	A <numeric[1]> between 0 and 1. Use 0 for bottom- or left-aligned densities, use 1 for top- or right-aligned densities and 0.5 for violin shapes.
oob	An out-of-bounds handling function that affects the cap colour. Can be one of the following: • A <function> like oob_squish. • A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".
alpha	A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Details

Non-finite values such as NA and NaN are ignored while infinite values such as -Inf and Inf are squished to the limits.

Value

A <GizmoDensity> object.

See Also

Other gizmos: gizmo_barcap(), gizmo_grob(), gizmo_histogram(), gizmo_stepcap()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  scale_colour_viridis_c()

# Density from plot data
p + guides(colour = gizmo_density())

# Using bins instead of gradient
p + guides(colour = gizmo_density("bins"))

# Providing custom values to compute density of
```

```
p + guides(colour = gizmo_density(density = runif(1000, min = 5, max = 35)))

# Providing a precomputed density
p + guides(colour = gizmo_density(density = density(mpg$cty, adjust = 0.5)))

# Alternatively, parameters may be passed through density.args
p + guides(colour = gizmo_density(density.args = list(adjust = 0.5)))
```

---

gizmo_grob *Guide gizmo: custom grob*

Description

This guide displays a user-provided grob.

Usage

```
gizmo_grob(
  grob,
  width = grobWidth(grob),
  height = grobHeight(grob),
  hjust = 0.5,
  vjust = 0.5,
  position = waiver()
)
```

Arguments

grob	A <grob> to display.
width, height	A [<unit[1]>][grid::unit] setting the allocated width and height of the the grob respectively.
hjust, vjust	A <numeric[1]> between 0 and 1 setting the horizontal and vertical justification of the grob when used as a guide for the x and y aesthetics.
position	Where this guide should be drawn: one of "top", "bottom", "left", or "right".

Value

A <GizmoGrob> object.

See Also

Other gizmos: gizmo_barcap(), gizmo_density(), gizmo_histogram(), gizmo_stepcap()

Examples

```
circle <- grid::circleGrob()

# A standard plot with grob gizmos
ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  guides(
    x.sec = gizmo_grob(
      circle, hjust = 0.75,
      width = unit(2, "cm"), height = unit(2, "cm")
    ),
    colour = gizmo_grob(
      circle, width = unit(1, "cm"), height = unit(1, "cm")
    )
  )
```

`gizmo_histogram` *Guide gizmo: histogram*

Description

This guide displays a histogram of the aesthetic. If the aesthetic is `colour` or `fill`, the shape will reflect this.

Usage

```
gizmo_histogram(
  key = "sequence",
  hist = NULL,
  hist.args = list(),
  hist.fun = graphics::hist,
  just = 1,
  oob = oob_keep,
  alpha = NA,
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

`key`	A sequence key or binned key specification.
`hist`	One of the following: • `NULL` for computing histograms on the data values (default). • an atomic `<vector>` to feed to the `hist.fun` function.

- A named <list> with breaks and counts numeric items, where the breaks item is exactly one element longer than the counts item. A typical way to construct such list is using the hist() function. Please note that <list> input is expected in scale-transformed space, not original data space.

hist.args — A <list> with additional arguments to the hist.fun argument. Only applies when hist is not provided as a <list> already.

hist.fun — A <function> to use for computing histograms when the hist argument is not provided as a list already.

just — A <numeric[1]> between 0 and 1. Use 0 for bottom- or left-aligned histograms, use 1 for top- or right-aligned histograms and 0.5 for centred histograms.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

direction — A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Details

Non-finite values such as NA and NaN are ignored while infinite values such as -Inf and Inf are squished to the limits.

Value

A <GizmoHistogram> object.

See Also

Other gizmos: gizmo_barcap(), gizmo_density(), gizmo_grob(), gizmo_stepcap()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  scale_colour_viridis_c()

# Histogram from plot data
p + guides(colour = gizmo_histogram())
```

```
# Using bins instead of gradient
p + guides(colour = gizmo_histogram("bins"))

# Providing custom values to compute histogram
p + guides(colour = gizmo_histogram(hist = runif(1000, min = 5, max = 35)))

# Providing precomputed histogram
p + guides(colour = gizmo_histogram(hist = hist(mpg$cty, breaks = 10)))

# Alternatively, parameters may be passed through hist.args
p + guides(colour = gizmo_histogram(hist.arg = list(breaks = 10)))
```

`gizmo_stepcap` *Guide gizmo: capped colour steps*

Description

This guide displays a binned variant of the colour bar with optional caps at either ends of the bar.

Usage

```
gizmo_stepcap(
  key = "bins",
  shape = "triangle",
  size = NULL,
  show = NA,
  alpha = NA,
  oob = "keep",
  theme = NULL,
  position = waiver(),
  direction = NULL
)
```

Arguments

key	A bins key specificiation. Defaults to `key_bins(even.steps = FALSE, show.limits = NULL)`. Changing the arguments to `key_bins()` is fine, but changing the key type is not advised.
shape	A cap specification by providing one of the following: • A cap `<function>`, such as `cap_triangle()`. • A `<character[1]>` naming a cap function without the 'cap_'-prefix, e.g. `"round"`. • A two column `<matrix[n, 2]>` giving coordinates for a cap, like those created by cap functions such as `cap_arch()`.
size	A `<unit>` setting the size of the cap. When `NULL` (default), cap size will be proportional to the `shape` coordinates and the `legend.key.size` theme setting.

show	A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.
alpha	A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.
oob	An out-of-bounds handling function that affects the cap colour. Can be one of the following: • A <function> like oob_squish. • A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".

Value

A GizmoStepcap object.

See Also

Other gizmos: gizmo_barcap(), gizmo_density(), gizmo_grob(), gizmo_histogram()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point()

# Just some recangles
p + scale_colour_viridis_c(guide = gizmo_stepcap())

# Caps show up when there is data outside the limits
p + scale_colour_viridis_c(
  limits = c(10, 30),
  guide = gizmo_stepcap()
)

# The scale's out-of-bounds handler determines cap colour
p + scale_colour_viridis_c(
  limits = c(10, 30), oob = scales::oob_squish,
  guide = gizmo_stepcap()
)
```

```
# Customising the display of the guide
p +
  scale_colour_viridis_c(
    oob = scales::oob_squish,
    guide = gizmo_stepcap(
      shape = "round", show = c(FALSE, TRUE),
      size = unit(1, "cm"),
      theme = theme(legend.key.height = unit(4, "cm"))
    )
  ) +
  theme(
    legend.frame = element_rect(colour = "black"),
    legend.key.width = unit(0.5, "cm")
  )
```

`guide-composition` *Guide composition*

Description

[Experimental]

Guide composition is a meta-guide orchestrating an ensemble of other guides. On their own, a 'composing' guide is not very useful as a visual reflection of a scale.

Usage

```
new_compose(
  guides,
  args = list(),
  ...,
  available_aes = c("any", "x", "y", "r", "theta"),
  call = caller_env(),
  super = Compose
)
```

Arguments

`guides`	A <list> of guides wherein each element is one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character[1]> naming such a function, without the `guide_` or `primitive_` prefix.
`args`	A <list> of arguments to pass to guides that are given either as a function or as a string.
`...`	Additional parameters to pass on to `new_guide()`.
`available_aes`	A <character> giving aesthetics that must match the the guides.
`call`	A call to display in messages.
`super`	A <Compose> class object giving a meta-guide for composition.

Value

A <Compose> (sub-)class guide that composes other guides.

See Also

Other composition: compose_crux(), compose_ontop(), compose_sandwich(), compose_stack()

Examples

```
# The `new_compose()` function is not intended to be used directly
my_composition <- new_compose(list("axis", "axis"), super = ComposeStack)

# Is the same as
my_composition <- compose_stack("axis", "axis")
```

---

guide-gizmos *Guide gizmos*

Description

Guide gizmos are a speciality guide components that are very specific to one or a few aesthetics to display.

Typically they can be composed with other guides or guide primitives to form a complete guide.

---

guide-primitives *Guide primitives*

Description

Guide primitives are the building blocks of more complex guides. On their own, they are not very useful as a visual reflection of a scale.

Their purpose is to be combined with one another to form a more complex, complete guides that *do* reflect a scale in some way.

Details

The guide primitives are simple, but flexible in that they are not tailored for one particular aesthetic. That way they can be reused and combined at will.

guide_axis_base *Custom axis guide*

Description

This axis guide is a visual representation of position scales and can represent the `x`, `y`, `theta` and `r` aesthetics. It differs from `guide_axis()` in that it can accept custom keys and is can act as an axis for `coord_radial()` like `guide_axis_theta()`.

Usage

```
guide_axis_base(
  key = NULL,
  title = waiver(),
  theme = NULL,
  n.dodge = 1,
  check.overlap = FALSE,
  angle = waiver(),
  cap = "none",
  bidi = FALSE,
  order = 0,
  position = waiver()
)
```

Arguments

`key`	A standard key specification. Defaults to `key_auto()`. See more information in the linked topic and the 'Details' section.
`title`	A `<character[1]>` or `<expression[1]>` indicating the title of the guide. If `NULL`, the title is not shown. The default, `waiver()`, takes the name of the scale object or the name specified in `labs()` as the title.
`theme`	A `<theme>` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides and is combined with the plot's theme.
`n.dodge`	An positive `<integer[1]>` setting the number of layers text labels can occupy to avoid overlapping labels.
`check.overlap`	A `<logical[1]>` indicating whether to check for and omit overlapping text. If `TRUE`, first, last and middle labels are recursively prioritised in that order. If `FALSE`, all labels are drawn.
`angle`	A specification for the text angle. Compared to setting the `angle` argument in `element_text()`, this argument uses some heuristics to automatically pick the `hjust` and `vjust` that you probably want. Can be one of the following: • `NULL` to take angles and justification settings directly from the theme. • `waiver()` to allow reasonable defaults in special cases. • A `<numeric[1]>` between -360 and 360 for the text angle in degrees.

cap — A method to cap the axes. One of the following:

- A <character[1]> with one of the following:
  - "none" to perform no capping.
  - "both" to cap the line at both ends at the most extreme breaks.
  - "upper" to cap the line at the upper extreme break.
  - "lower" to cap the line at the lower extreme break.
- A <logical>[1], where TRUE is equivalent to "both" and FALSE is equivalent to "none" in the options above.
- A sorted <numeric>[2n] with an even number of members. The lines will be drawn between every odd-even pair.
- A <function> that takes the scale's breaks as the first argument, the scale's limits as the second argument and returns a <numeric>[2n] as described above.

bidi — A <logical[1]>: whether ticks should be drawn bidirectionally (TRUE) or in a single direction (FALSE, default).

order — A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Under the hood, this guide is a stack composition of a line, ticks and labels primitives.

To set minor ticks, use key = "minor", or use the type argument in key_manual() or key_map().

To use this as a logarithmic axis, set key = "log".

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_nested(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group()

Examples

```
# A standard plot with custom keys
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  scale_x_continuous(
    guide = guide_axis_base(key = key_minor())
  ) +
  scale_y_continuous(
    guide = guide_axis_base(key = key_manual(c(20, 25, 30, 40)))
```

```
  )
p

# Is translated to theta axis without fuss
p + coord_radial()

# To use as logarithmic axis:
ggplot(msleep, aes(bodywt, brainwt)) +
  geom_point(na.rm = TRUE) +
  scale_x_continuous(
    transform = "log10",
    guide = guide_axis_base("log")
  )
```

`guide_axis_nested` *Nested axis guide*

Description

This axis guide gives extra range annotations to position scales. It can be used to infer nesting structure from labels or annotate ranges.

Usage

```
guide_axis_nested(
  key = "range_auto",
  regular_key = "auto",
  type = "bracket",
  title = waiver(),
  theme = NULL,
  angle = waiver(),
  cap = "none",
  bidi = FALSE,
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = NULL,
  levels_text = NULL,
  ...,
  order = 0,
  position = waiver()
)
```

Arguments

`key`	A range key specification. If not `key = "range_auto"`, additional labels will be inserted to represent point values.
`regular_key`	A standard key specification for the appearance of regular tick marks.

type — Appearance of ranges, either "box" to put text in boxes or "bracket" (default) to text brackets.

title — A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

angle — A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following:

- NULL to take angles and justification settings directly from the theme.
- waiver() to allow reasonable defaults in special cases.
- A <numeric[1]> between -360 and 360 for the text angle in degrees.

cap — A method to cap the axes. One of the following:

- A <character[1]> with one of the following:
  - "none" to perform no capping.
  - "both" to cap the line at both ends at the most extreme breaks.
  - "upper" to cap the line at the upper extreme break.
  - "lower" to cap the line at the lower extreme break.
- A <logical>[1], where TRUE is equivalent to "both" and FALSE is equivalent to "none" in the options above.
- A sorted <numeric>[2n] with an even number of members. The lines will be drawn between every odd-even pair.
- A <function> that takes the scale's breaks as the first argument, the scale's limits as the second argument and returns a <numeric>[2n] as described above.

bidi — A <logical[1]>: whether ticks should be drawn bidirectionally (TRUE) or in a single direction (FALSE, default).

oob — A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".

drop_zero — A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).

pad_discrete — A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.

levels_text — A list of <element_text> objects to customise how text appears at every level.

... — Arguments passed on to primitive_bracket(), primitive_box() or primitive_fence().

order — A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Details

Under the hood, this guide is a stack composition of a line, ticks, optionally labels and either bracket, box or fence primitives.

By default, the `key = "range_auto"` will incorporate the 0th level labels inferred from the scale's labels. These labels will look like regular labels.

To offer other keys the opportunity to display ranges alongside regular-looking labels, the `regular_key` argument can be used to setup a separate key for display in between the ticks and ranges.

Value

A `<Guide>` object.

See Also

Other standalone guides: `guide_axis_base()`, `guide_colbar()`, `guide_colring()`, `guide_colsteps()`, `guide_legend_base()`, `guide_legend_cross()`, `guide_legend_group()`

Examples

```
# A plot with nested categories on the x-axis
p <- ggplot(mpg, aes(interaction(drv, cyl), hwy)) +
  geom_boxplot()

p + guides(x = "axis_nested")

# Apply styling to brackets
p + guides(x = "axis_nested") +
  theme_guide(bracket = element_line("red", linewidth = 1))

# Don't drop nesting indicators that have 0-width
p + guides(x = guide_axis_nested(drop_zero = FALSE))

# Change additional padding for discrete categories
p + guides(x = guide_axis_nested(pad_discrete = 0))

# Change bracket type
p + guides(x = guide_axis_nested(bracket = "curvy"))

# Use boxes instead of brackets + styling of boxes
p + guides(x = guide_axis_nested(type = "box")) +
  theme_guide(box = element_rect("limegreen", "forestgreen"))

# Using fences instead of brackets + styling of fences
p + guides(x = guide_axis_nested(type = "fence", rail = "inner")) +
  theme_guide(
    fence.post = element_line("tomato"),
    fence.rail = element_line("dodgerblue")
  )

# Use as annotation of a typical axis
# `regular_key` controls display of typical axis
```

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  guides(x = guide_axis_nested(
    key = key_range_manual(start = 2:3, end = 5:6, name = c("First", "Second")),
    regular_key = key_manual(c(2, 2.5, 3, 5, 7))
  ))
```

---

guide_colbar *Custom colour bar guide*

Description

Similar to guide_colourbar(), this guide displays continuous colour or fill aesthetics. It has additional options to display caps at the end of the bar, depending on out-of-bounds values.

Usage

```
guide_colbar(
  title = waiver(),
  key = "auto",
  first_guide = "axis_base",
  second_guide = first_guide,
  shape = "triangle",
  size = NULL,
  show = NA,
  nbin = 15,
  alpha = NA,
  reverse = FALSE,
  oob = scales::oob_keep,
  theme = NULL,
  vanilla = TRUE,
  position = waiver(),
  available_aes = c("colour", "fill")
)
```

Arguments

title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
key	A sequence key specification. Defaults to key_sequence(n = 15). Changing the argument to key_sequence() is fine, but changing the key type is not advised.
first_guide, second_guide	Guides to flank the colour bar. Each guide can be specified using one of the following:

- A <Guide> class object.

- A <function> that returns a <Guide> class object.
- A <character> naming such a function, without the guide_ or primitive_ prefix.

The first_guide will be placed at the location specified by the legend.text.position theme setting. The second_guide will be placed opposite that position. When second_guide has a label suppression mechanism, no labels will be drawn for that guide.

shape — A cap specification by providing one of the following:

- A cap <function>, such as cap_triangle().
- A <character[1]> naming a cap function without the 'cap_'-prefix, e.g. "round".
- A two column <matrix[n, 2]> giving coordinates for a cap, like those created by cap functions such as cap_arch().

size — A <unit> setting the size of the cap. When NULL (default), cap size will be proportional to the shape coordinates and the legend.key.size theme setting.

show — A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.

nbin — A positive <integer[1]> determining how many colours to use for the colour gradient.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

reverse — A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

vanilla — A <logical[1]> whether to have the default style match the vanilla guide_colourbar() (TRUE) or take the theme verbatim (FALSE).

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

available_aes — A <character> vector listing the aesthetics for which this guide can be build.

Details

As colours are always rendered as gradients, it is important to use a graphics device that can render these. This can be checked by using check_device("gradients").

Value

A <Guide> object

See Also

Other standalone guides: guide_axis_base(), guide_axis_nested(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = cty))

# The colourbar shows caps when values are out-of-bounds (oob)
p + scale_colour_viridis_c(
  limits = c(10, NA),
  guide = "colbar"
)

# It also shows how oob values are handled
p + scale_colour_viridis_c(
  limits = c(10, NA), oob = scales::oob_squish,
  guide = "colbar"
)

# Adjusting the type of cap
p + scale_colour_viridis_c(
  limits = c(10, 30), oob = scales::oob_squish,
  guide = guide_colbar(shape = "round")
)

# One-sided ticks
p + scale_colour_viridis_c(
  guide = guide_colbar(second_guide = "none")
)

# Colour bar with minor breaks
p + scale_colour_viridis_c(
  minor_breaks = scales::breaks_width(1),
  guide = guide_colbar(key = "minor")
)

# Using log ticks on a colourbar
ggplot(msleep, aes(sleep_total, sleep_rem)) +
  geom_point(aes(colour = bodywt), na.rm = TRUE) +
  scale_colour_viridis_c(
    transform = "log10",
    guide = guide_colbar(key = "log")
  )
```

guide_colring *Colour rings and arcs*

Description

Similar to guide_colourbar(), this guide displays continuous colour or fill aesthetics. Instead of a bar, the gradient in shown in a ring or arc, which can be convenient for cyclical palettes such as some provided in the **scico** package.

Usage

```
guide_colring(
  title = waiver(),
  key = "auto",
  start = 0,
  end = NULL,
  outer_guide = "axis_base",
  inner_guide = "axis_base",
  nbin = 300,
  reverse = FALSE,
  show_labels = "outer",
  theme = NULL,
  vanilla = TRUE,
  position = waiver(),
  available_aes = c("colour", "fill"),
  ...
)
```

Arguments

title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
key	A standard key specification. Defaults to key_auto().
start, end	A <numeric[1]> in radians specifying the offset of the starting and end points from 12 o'clock. The NULL default for end, internally defaults to start + 2 * pi.
outer_guide, inner_guide	Guides to display on the outside and inside of the colour ring. Each guide can be specified using one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character> naming such function, without the guide_ or primitive_ prefix.
nbin	A positive <integer[1]> determining how many colours to display.

reverse	A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.
show_labels	A <character[1]> indicating for which guide labels should be shown. Can be one of "outer" (default), "inner", "both" or "none". Note that labels can only be omitted if the related guide has a label suppression mechanism.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
vanilla	A <logical[1]> whether to have the default style match the vanilla guide_colourbar() (TRUE) or take the theme verbatim (FALSE).
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
available_aes	A <character> vector listing the aesthetics for which this guide can be build.
...	Arguments forwarded to the outer_guide and inner_guide if provided as functions or strings.

Value

A <Guide> object.

See Also

Other standalone guides: guide_axis_base(), guide_axis_nested(), guide_colbar(), guide_colsteps(), guide_legend_base(), guide_legend_cross(), guide_legend_group()

Examples

```
# Rings works best with a cyclical palette
my_pal <- c("black", "tomato", "white", "dodgerblue", "black")

p <- ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  scale_colour_gradientn(colours = my_pal)

# Standard colour ring
p + guides(colour = "colring")

# As an arc
p + guides(colour = guide_colring(
  start = 1.25 * pi, end = 2.75 * pi
))

# Removing the inner tick marks
p + guides(colour = guide_colring(inner_guide = "none"))

# Include labels on the inner axis
p + guides(colour = guide_colring(show_labels = "both"))

# Passing an argument to inner/outer guides
p + guides(colour = guide_colring(angle = 0))
```

guide_colsteps *Custom colour steps guide*

Description

Similar to guide_coloursteps(), this guide displays continuous colour or fill aesthetics. It has additional options to display caps at the end of the bar, depending on out-of-bounds values.

Usage

```
guide_colsteps(
  title = waiver(),
  key = "bins",
  first_guide = "axis_base",
  second_guide = "axis_base",
  shape = "triangle",
  size = NULL,
  show = NA,
  alpha = NA,
  reverse = FALSE,
  oob = scales::oob_keep,
  theme = NULL,
  position = waiver(),
  vanilla = TRUE,
  available_aes = c("colour", "fill")
)
```

Arguments

title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
key	A bins key specificiation. Defaults to key_bins(even.steps = FALSE, show.limits = NULL). Changing the arguments to key_bins() is fine, but changing the key type is not advised.
first_guide, second_guide	Guides to flank the colour steps. Each guide can be specified using one of the following: • A <Guide> class object. • A <function> that returns a <Guide> class object. • A <character> naming such a function, without the guide_ or primitive_ prefix. The first_guide will be placed at the location specified by the legend.text.position theme setting. The second_guide will be placed opposite that position. When second_guide has a label suppression mechanism, no labels will be drawn for that guide.

shape — A cap specification by providing one of the following:

- A cap <function>, such as cap_triangle().
- A <character[1]> naming a cap function without the 'cap_'-prefix, e.g. "round".
- A two column <matrix[n, 2]> giving coordinates for a cap, like those created by cap functions such as cap_arch().

size — A <unit> setting the size of the cap. When NULL (default), cap size will be proportional to the shape coordinates and the legend.key.size theme setting.

show — A <logical> to control how caps are displayed at the ends of the bar. When TRUE, caps are always displayed. When FALSE, caps are never displayed. When NA (default), caps are displayed when the data range exceed the limits. When given as <logical[2]>, show[1] controls the display at the lower end and show[2] at the upper end.

alpha — A <numeric[1]> between 0 and 1 setting the colour transparency of the bar. Use NA to preserve the alpha encoded in the colour itself.

reverse — A <logical[1]> whether to reverse continuous guides. If TRUE, guides like colour bars are flipped. If FALSE (default), the original order is maintained.

oob — An out-of-bounds handling function that affects the cap colour. Can be one of the following:

- A <function> like oob_squish.
- A <character[1]> naming such a function without the 'oob'-prefix, such as "keep".

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

vanilla — A <logical[1]> whether to have the default style match the vanilla guide_colourbar() (TRUE) or take the theme verbatim (FALSE).

available_aes — A <character> vector listing the aesthetics for which this guide can be build.

Details

As steps are rendered as clipped rectangles, it is important to use a graphics device that can render clipped paths. This can be checked by using check_device("clippingPaths").

Value

A <Guide> object

See Also

Other standalone guides: guide_axis_base(), guide_axis_nested(), guide_colbar(), guide_colring(), guide_legend_base(), guide_legend_cross(), guide_legend_group()

Examples

```
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = cty))

# The colour steps show caps when values are out-of-bounds
p + scale_colour_viridis_b(
  limits = c(10, NA),
  guide = "colsteps"
)

# It also shows how oob values are handled
p + scale_colour_viridis_b(
  limits = c(10, 30), oob = scales::oob_censor,
  guide = "colsteps"
)

# Adjusting the type of cap
p + scale_colour_viridis_b(
  limits = c(10, 30),
  guide = guide_colsteps(shape = "round")
)

# The default is to use the breaks as-is
p + scale_colour_viridis_b(
  limits = c(10, 30), breaks = c(10, 20, 25),
  guide = "colsteps"
)

# But the display can be set to use evenly spaced steps
p + scale_colour_viridis_b(
  limits = c(10, 30), breaks = c(10, 20, 25),
  guide = guide_colsteps(key = key_bins(even.steps = TRUE))
)

# Using tick marks by swapping side guides
p + scale_colour_viridis_b(
  guide = guide_colsteps(
    first_guide  = "axis_base",
    second_guide = "axis_base"
  )
)
```

---

guide_legend_base *Custom legend guide*

Description

This legend closely mirrors `ggplot2::guide_legend()`, but has two adjustments. First, `guide_legend_base()` supports a `design` argument for a more flexible layout. Secondly, the `legend.spacing.y` theme element is observed verbatim instead of overruled.

Usage

```
guide_legend_base(
  key = NULL,
  title = waiver(),
  theme = NULL,
  design = NULL,
  nrow = NULL,
  ncol = NULL,
  reverse = FALSE,
  override.aes = list(),
  position = NULL,
  direction = NULL,
  order = 0
)
```

Arguments

key	A standard key specification. Defaults to key_auto(). See more information in the linked topic.
title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
design	Specification of the legend layout. One of the following: • NULL (default) to use the layout algorithm of guide_legend(). • A <character[1]> string representing a cell layout wherein # defines an empty cell. See examples. • A <matrix[n, m]> representing a cell layout wherein NA defines an empty cell. See examples. Non-string atomic vectors will be treated with as.matrix().
nrow, ncol	A positive <integer[1]> setting the desired dimensions of the legend layout. When NULL (default), the dimensions will be derived from the design argument or fit to match the number of keys.
reverse	A <logical[1]> whether the order of keys should be inverted.
override.aes	A named <list> specifying aesthetic parameters of the key glyphs. See details and examples in guide_legend().
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".
order	A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.

Value

A <GuideLegend> object.

See Also

Other standalone guides: guide_axis_base(), guide_axis_nested(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_cross(), guide_legend_group()

Other legend guides: guide_legend_cross(), guide_legend_group()

Examples

```
# A dummy plot
p <- ggplot(data.frame(x = 1:3, type = c("tic", "tac", "toe"))) +
  aes(x, x, shape = type) +
  geom_point(na.rm = TRUE) +
  scale_shape_manual(values = c(1, 4, NA))

# A design string, each character giving a cell value.
# Newlines separate rows, white space is ignored.
design <- "
  123
  213
  321
"

# Alternatively, the same can be specified using a matrix directly
# design <- matrix(c(1, 2, 3, 2, 1, 3, 3, 2, 1), 3, 3, byrow = TRUE)

p + guides(shape = guide_legend_base(design = design))

# Empty cells can be created using `#`
design <- "
  #2#
  1#3
"

# Alternatively:
# design <- matrix(c(NA, 1, 2, NA, NA, 3), nrow = 2)

p + guides(shape = guide_legend_base(design = design))
```

---

guide_legend_cross *Cross legend guide*

Description

This is a legend type similar to guide_legend() that displays crosses, or: interactions, between two variables.

Usage

```
guide_legend_cross(
  key = NULL,
  title = waiver(),
  swap = FALSE,
  col_text = element_text(angle = 90, vjust = 0.5),
  override.aes = list(),
  reverse = FALSE,
  theme = NULL,
  position = NULL,
  direction = NULL,
  order = 0
)
```

Arguments

key	One of the following key specifications: • A group split specification when using the legend to display a compound variable like `paste(var1, var2)`. • A standard key specification, like `key_auto()`, when crossing two separate variables across two scales.
title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, `waiver()`, takes the name of the scale object or the name specified in `labs()` as the title.
swap	A <logical[1]> which when TRUE exchanges the column and row variables in the displayed legend.
col_text	An <element_text> object giving adjustments to text for the column labels. Can be NULL to display column labels in equal fashion to the row labels.
override.aes	A named <list> specifying aesthetic parameters of the key glyphs. See details and examples in `guide_legend()`.
reverse	A <logical[2]> whether the order of the keys should be inverted, where the first value controls the row order and second value the column order. Input as <logical[1]> will be recycled.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".
direction	A <character[1]> indicating the direction of the guide. Can be on of "horizontal" or "vertical".
order	A positive <integer[1]> that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If 0 (default), the order is determined by a hashing indicative settings of a guide.

Value

A <GuideLegend> object.

See Also

Other standalone guides: guide_axis_base(), guide_axis_nested(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_group()

Other legend guides: guide_legend_base(), guide_legend_group()

Examples

```
# Standard use for single aesthetic. The default is to split labels to
# disentangle aesthetics that are already crossed (by e.g. `paste()`)
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = paste(year, drv))) +
  guides(colour = "legend_cross")

# If legends should be merged between identical aesthetics, both need the
# same legend type.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = paste(year, drv), shape = paste(year, drv))) +
  guides(colour = "legend_cross", shape = "legend_cross")

# Crossing two aesthetics requires a shared title and `key = "auto"`. The
# easy way to achieve this is to predefine a shared guide.
my_guide <- guide_legend_cross(key = "auto", title = "My title")

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = drv, shape = factor(year))) +
  guides(colour = my_guide, shape  = my_guide)

# You can cross more than 2 aesthetics but not more than 2 unique aesthetics.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = drv, shape = factor(year), size = factor(drv))) +
  scale_size_ordinal() +
  guides(colour = my_guide, shape = my_guide, size = my_guide)

# You can merge an aesthetic that is already crossed with an aesthetic that
# contributes to only one side of the cross.
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = paste(year, drv), shape  = drv)) +
  guides(
    colour = guide_legend_cross(title = "My Title"),
    shape  = guide_legend_cross(title = "My Title", key = "auto")
  )
```

guide_legend_group *Grouped legend*

Description

This legend resembles `ggplot2::guide_legend()`, but has the ability to keep groups in blocks with their own titles.

Usage

```
guide_legend_group(
  key = "group_split",
  title = waiver(),
  override.aes = list(),
  nrow = NULL,
  ncol = NULL,
  theme = NULL,
  position = NULL,
  direction = NULL,
  order = 0
)
```

Arguments

`key`	A group key specification. Defaults to `key_group_split()` to split labels to find groups.
`title`	A `<character[1]>` or `<expression[1]>` indicating the title of the guide. If `NULL`, the title is not shown. The default, `waiver()`, takes the name of the scale object or the name specified in `labs()` as the title.
`override.aes`	A named `<list>` specifying aesthetic parameters of the key glyphs. See details and examples in `guide_legend()`.
`nrow`, `ncol`	A positive `<integer[1]>` setting the desired dimensions of the legend layout. Either `nrow` or `ncol` can be set, but not both,
`theme`	A `<theme>` object to style the guide individually or differently from the plot's theme settings. The `theme` argument in the guide overrides and is combined with the plot's theme.
`position`	A `<character[1]>` giving the location of the guide. Can be one of `"top"`, `"bottom"`, `"left"` or `"right"`.
`direction`	A `<character[1]>` indicating the direction of the guide. Can be on of `"horizontal"` or `"vertical"`.
`order`	A positive `<integer[1]>` that specifies the order of this guide among multiple guides. This controls in which order guides are merged if there are multiple guides for the same position. If `0` (default), the order is determined by a hashing indicative settings of a guide.

Value

A <GuideLegend> object.

See Also

Other standalone guides: guide_axis_base(), guide_axis_nested(), guide_colbar(), guide_colring(), guide_colsteps(), guide_legend_base(), guide_legend_cross()

Other legend guides: guide_legend_base(), guide_legend_cross()

Examples

```
# Standard plot for selection of `msleep`
df <- msleep[c(9, 28, 11, 5, 34, 54, 64, 24, 53), ]

p <- ggplot(df) +
  aes(bodywt, awake, colour = paste(order, name)) +
  geom_point()

# By default, groups are inferred from the name
p + guides(colour = "legend_group")

# You can also use a look-up table for groups
# The lookup table can be more expansive than just the data:
# We're using the full 'msleep' data here instead of the subset
lut <- key_group_lut(msleep$name, msleep$order)

p + aes(colour = name) +
  guides(colour = guide_legend_group(key = lut))

# `nrow` and `ncol` apply within groups
p + guides(colour = guide_legend_group(nrow = 1))

# Groups are arranged according to `direction`
p + guides(colour = guide_legend_group(ncol = 1, direction = "horizontal")) +
  theme(legend.title.position = "top")

# Customising the group titles
p + guides(colour = "legend_group") +
  theme(
    legendry.legend.subtitle.position = "left",
    legendry.legend.subtitle = element_text(
      hjust = 1, vjust = 1, size = rel(0.9),
      margin = margin(t = 5.5, r = 5.5)
    )
  )

# Changing the spacing between groups
p + guides(colour = "legend_group") +
  theme(legendry.group.spacing = unit(0, "cm"))
```

key_group *Group keys*

Description

These functions are helper functions for working with grouped data as keys in guides. They all share the goal of creating a guide key, but have different methods.

- `key_group_split()` is a function factory whose functions make an attempt to infer groups from the scale's labels.
- `key_group_lut()` is a function factory whose functions use a look up table to sort out group membership.

Usage

```
key_group_split(sep = "[^[:alnum:]]+", reverse = FALSE)

key_group_lut(members, group, ungrouped = "Other")
```

Arguments

`sep`	A `<character[1]>` giving a regular expression to use for splitting labels provided by the scale using the `strsplit()` function. By defaults, labels are splitted on any non-alphanumeric character.
`reverse`	A `<logical[1]>` which if `FALSE` (default) treats the first part of the split string as groups and later parts as members. If `TRUE`, treats the last part as groups.
`members`	A vector including the scale's `breaks` values.
`group`	A vector parallel to `members` giving the group of each member.
`ungrouped`	A `<character[1]>` giving a group label to assign to the scale's `breaks` that match no values in the `members` argument.

Value

A function to use as the `key` argument in a guide.

See Also

Other keys: `key_range`, `key_specialty`, `key_standard`

Examples

```
# Example scale
values <- c("group A:value 1", "group A:value 2", "group B:value 1")
template <- scale_colour_discrete(limits = values)

# Treat the 'group X' part as groups
key <- key_group_split(sep = ":")
```

```
key(template)

# Treat the 'value X' part as groups
key <- key_group_split(sep = ":", reverse = TRUE)
key(template)

# Example scale
template <- scale_colour_discrete(limits = msleep$name[c(1, 7, 9, 23, 24)])

# A lookup table can have more entries than needed
key <- key_group_lut(msleep$name, msleep$order)
key(template)

# Or less entries than needed
key <- key_group_lut(
  msleep$name[23:24], msleep$order[23:24],
  ungrouped = "Other animals"
)
key(template)
```

`key_range` *Range keys*

Description

These functions are helper functions for working with ranged data as keys in guides. They all share the goal creating of a guide key, but have different methods:

- `key_range_auto()` is a function factory whose functions make an attempt to infer ranges from the scale's labels.
- `key_range_manual()` uses user-provided vectors to set ranges.
- `key_range_map()` makes mappings from a `<data.frame>` to set ranges.

Usage

```
key_range_auto(sep = "[^[:alnum:]]+", reverse = FALSE, ...)

key_range_manual(start, end, name = NULL, level = NULL, ...)

key_range_map(data, ..., .call = caller_env())
```

Arguments

`sep`	A `<character[1]>` giving a regular expression to use for splitting labels provided by the scale using `strsplit()`. Defaults to splitting on any non-alphanumeric character.
`reverse`	A `<logical[1]>` which if `FALSE` (default) treats the first labels as the inner labels and the last labels as the outer labels. If `TRUE`, thee first labels are treated as the outer labels and the last labels are treated as the inner labels.

...	<data-masking> A set of mappings similar to those provided to aes(), which will be evaluated in the data argument. For key_range_map(), these *must* contain start and end mappings. Can contain additional parameters for text styling, namely colour, family, face, size, hjust, vjust, angle and lineheight.
start, end	A vector that can be interpreted by the scale, giving the start and end positions of each range respectively.
name	A <character> or list of expressions
level	An <integer> giving the depth of each range to avoid overlaps between different ranges. When level is smaller than 1, no brackets are drawn.
data	A <data.frame> or similar object coerced by fortify() to a <data.frame>, in which the mapping argument is evaluated.
.call	A call to display in messages.

Details

The level variable is optional and when missing, the guides use an algorithm similar to IRanges::disjointBins() to avoid overlaps.

The key_range_auto() does *not* work with expression labels.

Value

For key_range_auto() a function. For key_range_manual() and key_range_map() a <data.frame> with the <key_range> class.

See Also

Other keys: key_group, key_specialty, key_standard

Examples

```
# Example scale
template <- scale_x_discrete(limits = c("A 1", "B 1", "C&1", "D&2", "E&2"))

# By default, splits on all non-alphanumeric characters
auto <- key_range_auto()
auto(template)

# Only split on a specific character
auto <- key_range_auto(sep = "&")
auto(template)

# Treating the letters as outer labels and numbers as inner labels
auto <- key_range_auto(reverse = TRUE)
auto(template)

# Providing custom values
key_range_manual(
  start = c(1, 5,  10),
  end   = c(4, 15, 11),
```

```
  level = c(0, 2, 1),
  name  = c("A", "B", "C")
)

# Values from a <data.frame>
key_range_map(presidential, start = start, end = end, name = name)
```

---

key_specialty *Speciality keys*

Description

These functions are helper functions for working with keys in guides. The functions described here are not widely applicable and may only apply to a small subset of guides. As such, it is fine to adjust the arguments of a speciality key, but swapping types is ill-advised.

- `key_sequence()` is a function factory whose functions create a regularly spaced sequence between the limits of a scale. It is used in colour bar guides.
- `key_bins()` is a function factory whose function create a binned key given the breaks in the scale. It is used in colour steps guides.

Usage

```
key_sequence(n = 15)

key_bins(even.steps = FALSE, show.limits = NULL)
```

Arguments

`n`	A positive `<integer[1]>` giving the number of colours to use for a gradient.
`even.steps`	A `<logical[1]>` indicating whether the size of bins should be displayed as equal (`TRUE`) or proportional to their length in data space (`FALSE`).
`show.limits`	A `<logical[1]>` stating whether the limits of the scale should be shown with labels and ticks (`TRUE`) or remain hidden (`FALSE`). Note that breaks coinciding with limits are shown regardless of this setting. The default, `NULL`, consults the scale's `show.limits` setting or defaults to `FALSE`.

Value

For `key_sequence()` a function.

See Also

Other keys: `key_group`, `key_range`, `key_standard`

Examples

```
# An example scale
template <- scale_fill_viridis_c(limits = c(0, 10), breaks = c(2, 4, 6, 8))

# Retrieving colourbar and colourstep keys
key_sequence()(template)
key_bins()(template)
```

---

key_standard *Standard keys*

Description

These functions are helper functions for working with tick marks as keys in guides. They all share the goal of creating a guide key, but have different outcomes:

- `key_auto()` is a function factory whose functions extract a typical key from major breaks in a scale.
- `key_manual()` uses user-provided vectors to make a key.
- `key_map()` makes mappings from a `<data.frame>` to make a key.
- `key_minor()` is a function factory whose functions also extract minor break positions for minor tick marks.
- `key_log()` is a function factory whose functions place ticks at intervals in log10 space.
- `key_none()` makes an empty key with no entries.

Usage

```
key_auto(...)

key_manual(
  aesthetic,
  value = aesthetic,
  label = as.character(value),
  type = NULL,
  ...
)

key_map(data, ..., .call = caller_env())

key_minor(...)

key_log(
  prescale_base = NULL,
  negative_small = 0.1,
  expanded = TRUE,
  labeller = NULL,
```

```
  ...
)

key_none()
```

Arguments

...	<data-masking> A set of mappings similar to those provided to aes(), which will be evaluated in the data argument. These must contain aesthetic mapping.
aesthetic, value	A vector of values for the guide to represent equivalent to the breaks argument in scales. The aesthetic will be mapped, whereas value will not. For most intents and purposes, aesthetic and value should be identical.
label	A <character> or list of expressions to use as labels.
type	A <character> vector representing the one of the break types: "major", "minor" or "mini". If NULL (default), all breaks are treated as major breaks.
data	A <data.frame> or similar object coerced by fortify() to a <data.frame>, in which the mapping argument is evaluated.
.call	A call to display in messages.
prescale_base	A <numeric[1]> giving the base of logarithm to transform data manually. The default, NULL, will use the scale transformation to calculate positions. It is only advisable to set the prescale_base argument when the data have already been log-transformed. When using a log-transform in the scale or in coord_trans(), the default NULL is recommended.
negative_small	A <numeric[1]> setting the smallest absolute value that is marked with a tick in case the scale limits include 0 or negative numbers.
expanded	A <logical[1]> determining whether the ticks should cover the entire range after scale expansion (TRUE, default), or be restricted to the scale limits (FALSE).
labeller	A <function> that receives major breaks and returns formatted labels. For key_log(), NULL will default to scales::label_log() for strictly positive numbers and a custom labeller when negative numbers are included.

Value

For key_auto(), key_minor() and key_log() a function. For key_manual() and key_map() a <data.frame> with the <key_standard> class.

See Also

Other keys: key_group, key_range, key_specialty

Examples

```
# An example scale
template <- scale_x_continuous(limits = c(0, 10))
```

```
# The auto, minor and log keys operate on scales
key_auto()(template)
key_minor()(template)

# So does the log key
template <- scale_x_continuous(transform = "log10", limits = c(0.1, 10))
key_log()(template)

# Providing custom values
key_manual(
  aesthetic = 1:5,
  label = c("one", "two", "three", "four", "five")
)

# Values from a `<data.frame>`
key_map(ToothGrowth, aesthetic = unique(supp))

# Empty key
key_none()
```

`primitive_box` *Guide primitives: boxes*

Description

This function constructs a boxes guide primitive.

Usage

```
primitive_box(
  key = "range_auto",
  angle = waiver(),
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = 0.4,
  min_size = NULL,
  levels_box = NULL,
  levels_text = NULL,
  theme = NULL,
  position = waiver()
)
```

Arguments

`key`	A range key specification. See more information in the linked topic.
`angle`	A specification for the text angle. Compared to setting the `angle` argument in `element_text()`, this argument uses some heuristics to automatically pick the `hjust` and `vjust` that you probably want. Can be one of the following:

- NULL to take angles and justification settings directly from the theme.
- waiver() to allow reasonable defaults in special cases.
- A <numeric[1]> between -360 and 360 for the text angle in degrees.

oob	A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".
drop_zero	A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).
pad_discrete	A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.
min_size	A [<grid::unit[1]>][grid::unit] setting the minimal size of a box.
levels_box	A list of <element_rect> objects to customise how boxes appear at every level.
levels_text	A list of <element_text> objects to customise how text appears at every level.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A <PrimitiveBox> primitive guide that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

Common to both types is the following:

- legendry.box an <element_rect> for the boxes to draw.

As an axis guide:

- axis.text.{x/y}.{position} an <element_text> for the text inside the boxes.

As a legend guide:

- legend.text an <element_text> for the text inside the boxes.

See Also

Other primitives: primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(interaction(drv, year), displ)) +
 geom_point()

key <- key_range_manual(c(2, 4), c(5, 6), c("A", "B"))

# Adding as secondary guides
p + guides(
  x.sec = primitive_box(),
  y.sec = primitive_box(key = key)
)
```

---

primitive_bracket *Guide primitive: brackets*

Description

This function constructs a brackets guide primitive.

Usage

```
primitive_bracket(
  key = "range_auto",
  bracket = "line",
  angle = waiver(),
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = 0.4,
  levels_brackets = NULL,
  levels_text = NULL,
  theme = NULL,
  position = waiver()
)
```

Arguments

key	A range key specification. See more information in the linked topic.
bracket	A bracket by providing one of the following: • A bracket <function>, such as bracket_square. • A <character[1]> naming a bracket function without the 'bracket_'-prefix, e.g. "square". • A two-column <matrix[n, 2]> giving line coordinates for a bracket, like those created by bracket functions, such as bracket_round().
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following:

- NULL to take angles and justification settings directly from the theme.
- waiver() to allow reasonable defaults in special cases.
- A <numeric[1]> between -360 and 360 for the text angle in degrees.

oob — A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".

drop_zero — A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).

pad_discrete — A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.

levels_brackets — A list of <element_line> objects to customise how brackets appear at every level.

levels_text — A list of <element_text> objects to customise how text appears at every level.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A <PrimitiveBracket> primitive guide that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or a legend context.

Common to both types is the following:

- legendry.bracket an <element_line> for the line used to draw the brackets.
- legendry.backet.size a <unit> setting the space afforded to a bracket.

As an axis guide:

- axis.text.{x/y}.{position} an <element_text> for the text displayed over the brackets.

As a legend guide:

- legend.text an <element_text> for the text displayed over the brackets.

See Also

Other primitives: primitive_box(), primitive_fence(), primitive_labels(), primitive_line(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(interaction(drv, year), displ)) +
 geom_point()

key <- key_range_manual(c(2, 4), c(5, 6), c("A", "B"))

# Adding as secondary guides
p + guides(
  x.sec = primitive_bracket(),
  y.sec = primitive_bracket(key = key)
)
```

`primitive_fence` *Guide primitive: fence*

Description

This function constructs a fence guide primitive. The customisation options are easier to understand if we view fence 'post' as the vertical pieces of a real world fence, and the 'rail' as the horizontal pieces.

Usage

```
primitive_fence(
  key = "range_auto",
  rail = "none",
  angle = waiver(),
  oob = "squish",
  drop_zero = TRUE,
  pad_discrete = 0.5,
  levels_text = NULL,
  levels_post = NULL,
  levels_rail = NULL,
  theme = NULL,
  position = waiver()
)
```

Arguments

`key`	A range key specification. See more information in the linked topic.
`rail`	A `<character[1]>` giving an option for how to display fence railing. Can be either `"none"` (default) to display no railings, `"inner"` to draw one rail closer to the plot panel, `"outer"` to display one rail farther from the plot panel, or `"both"` to sandwich the labels between rails.
`angle`	A specification for the text angle. Compared to setting the `angle` argument in `element_text()`, this argument uses some heuristics to automatically pick the `hjust` and `vjust` that you probably want. Can be one of the following:

- NULL to take angles and justification settings directly from the theme.
- waiver() to allow reasonable defaults in special cases.
- A <numeric[1]> between -360 and 360 for the text angle in degrees.

oob — A method for dealing with out-of-bounds (oob) ranges. Can be one of "squish", "censor" or "none".

drop_zero — A <logical[1]> whether to drop near-zero width ranges (TRUE, default) or preserve them (FALSE).

pad_discrete — A <numeric[1]> giving the amount ranges should be extended when given as a discrete variable. This is applied after the drop_zero setting.

levels_text — A list of <element_text> objects to customise how text appears at every level.

levels_post, levels_rail — A list of <element_line> objects to customise how fence posts and rails are displayed at every level.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A <PrimitiveFence> primitive guie that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or legend context.

Common to both types is the following:

- legendry.fence.post an <element_line> for the line used to draw the pieces orthogonal to the direction of the scale.
- legendry.fence.rail an <element_line> for the line used to draw the pieces parallel to the direction of the scale.

As an axis guide:

- axis.text.{x/y}.{position} an <element_text> for the text displayed.

As a legend guide:

- legend.text an <element_text> for the text displayed.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_labels(), primitive_line(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(interaction(drv, year), displ)) +
  geom_point()

key <- key_range_manual(c(2, 4), c(5, 6), c("A", "B"))

# Adding as secondary guides
p + guides(
  x.sec = primitive_fence(rail = "inner"),
  y.sec = primitive_fence(key = key, rail = "outer")
)
```

---

primitive_labels *Guide primitive: labels*

Description

This function constructs a labels guide primitive.

Usage

```
primitive_labels(
  key = NULL,
  angle = waiver(),
  n.dodge = 1,
  check.overlap = FALSE,
  theme = NULL,
  position = waiver()
)
```

Arguments

key	A standard key specification. See more information in the linked topic.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
n.dodge	An positive <integer[1]> setting the number of layers text labels can occupy to avoid overlapping labels.
check.overlap	A <logical[1]> indicating whether to check for and omit overlapping text. If TRUE, first, last and middle labels are recursively prioritised in that order. If FALSE, all labels are drawn.

theme — A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.

position — A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A <PrimitiveLabels> primitive guide that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

As an axis guide:

- axis.text.{x/y}.{position} an <element_text> for the display of the labels.

As a legend guide.:

- legend.text an <element_text> for the display of the labels.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_line(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
 geom_point()

# Adding as secondary guides
p + guides(
  x.sec = primitive_labels(),
  y.sec = primitive_labels(n.dodge = 2)
)
```

---

primitive_line *Guide primitive: line*

Description

This function constructs a line guide primitive.

Usage

```
primitive_line(key = NULL, cap = "none", theme = NULL, position = waiver())
```

Arguments

key	A standard key specification. See more information in the linked topic.
cap	A method to cap the axes. One of the following: • A <character[1]> with one of the following: – "none" to perform no capping. – "both" to cap the line at both ends at the most extreme breaks. – "upper" to cap the line at the upper extreme break. – "lower" to cap the line at the lower extreme break. • A <logical>[1], where TRUE is equivalent to "both" and FALSE is equivalent to "none" in the options above. • A sorted <numeric>[2n] with an even number of members. The lines will be drawn between every odd-even pair. • A <function> that takes the scale's breaks as the first argument, the scale's limits as the second argument and returns a <numeric>[2n] as described above.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A PrimitiveLine primitive guide that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

As an axis guide:

- axis.line.{x/y}.{position} an <element_line> for the line style.

As a legend guide:

- legend.axis.line an <element_line> for the line style.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_spacer(), primitive_ticks(), primitive_title()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  theme(axis.line = element_line())
```

```
# Adding as secondary guides
p + guides(
  x.sec = primitive_line(),
  y.sec = primitive_line(cap = "both")
)
```

`primitive_spacer` *Guide primitive: spacer*

Description

This function constructs a spacer guide primitive.

Usage

```
primitive_spacer(
  space = NULL,
  title = waiver(),
  theme = NULL,
  position = waiver()
)
```

Arguments

space	A [<unit[1]>][grid::unit()]
title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A <PrimitiveSpacer> primitive guide that can be used inside other guides.

Styling options

#' Below are the theme options that determine the styling of this guide. This guide does not have option dependent on its role as axis or legend.

- legendry.guide.spacing A <unit> setting the amount of spacing when the space argument is NULL.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_ticks(), primitive_title()

Examples

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  guides(
    x = guide_axis_stack("axis", primitive_spacer(unit(1, "cm")), "axis")
  )
```

---

primitive_ticks *Guide primitive: line*

Description

This function constructs a ticks guide primitive.

Usage

```
primitive_ticks(key = NULL, bidi = FALSE, theme = NULL, position = waiver())
```

Arguments

key	A standard key specification. See more information in the linked topic.
bidi	A <logical[1]>: whether ticks should be drawn bidirectionally (TRUE) or in a single direction (FALSE, default).
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A PrimitiveTicks primitive guide that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

Common to both types is the following:

As an axis guide:

- axis.ticks.{x/y}.{position} an <element_line> for major tick lines.

- `axis.minor.ticks.{x/y}.{position}` an `<element_line>` for minor tick lines.
- `legendry.axis.mini.ticks` an `<element_line>` internally inheriting from the minor ticks for the smallest ticks in e.g. log axes.
- `axis.ticks.length.{x/y}.{position}` a `<unit>` for the major ticks length.
- `axis.minor.ticks.length.{x/y}.{position}` a `<unit>` for the minor ticks length.
- `legendry.axis.mini.ticks.length` a `<unit>` internally inheriting from the minor tick length for the smallest ticks in e.g. log axes.

As a legend guide:

- `legend.ticks` an `<element_line>` for major tick lines.
- `legendry.legend.minor.ticks` an `<element_line>` for minor tick lines.
- `legendry.legend.mini.ticks` an `<element_line>` for the smallest ticks in e.g. log axes.
- `legend.ticks.length` a `<unit>` for the major ticks length.
- `legendry.legend.minor.ticks.length` a `<unit>` for the minor ticks length.
- `legendry.legend.mini.ticks.length` a `<unit>` for the smallest ticks in e.g. log axes.

See Also

Other primitives: `primitive_box()`, `primitive_bracket()`, `primitive_fence()`, `primitive_labels()`, `primitive_line()`, `primitive_spacer()`, `primitive_title()`

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# Adding as secondary guides
p + guides(x.sec = primitive_ticks(), y.sec = primitive_ticks())
```

`primitive_title` *Guide primitive: title*

Description

This function constructs a title guide primitive.

Usage

```
primitive_title(
  title = waiver(),
  angle = waiver(),
  theme = NULL,
  position = waiver()
)
```

Arguments

title	A <character[1]> or <expression[1]> indicating the title of the guide. If NULL, the title is not shown. The default, waiver(), takes the name of the scale object or the name specified in labs() as the title.
angle	A specification for the text angle. Compared to setting the angle argument in element_text(), this argument uses some heuristics to automatically pick the hjust and vjust that you probably want. Can be one of the following: • NULL to take angles and justification settings directly from the theme. • waiver() to allow reasonable defaults in special cases. • A <numeric[1]> between -360 and 360 for the text angle in degrees.
theme	A <theme> object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide overrides and is combined with the plot's theme.
position	A <character[1]> giving the location of the guide. Can be one of "top", "bottom", "left" or "right".

Value

A <PrimitiveTitle> primitive guide that can be used inside other guides.

Styling options

Below are the theme options that determine the styling of this guide, which may differ depending on whether the guide is used in an axis or in a legend context.

As an axis guide:

- axis.title.{x/y}.{position} an <element_text> for the title display.

As a legend guide:

- legend.title an <element_text> for the title display.

See Also

Other primitives: primitive_box(), primitive_bracket(), primitive_fence(), primitive_labels(), primitive_line(), primitive_spacer(), primitive_ticks()

Examples

```
# A standard plot
p <- ggplot(mpg, aes(displ, hwy)) +
 geom_point()

# Adding as secondary guides
p + guides(
  x.sec = primitive_title("Horizontal Title"),
  y.sec = primitive_title(c("along vertical", "Multiple tiles"))
)
```

theme_guide *Theme wrapper for guides*

Description

This function has shorthand names for theme elements relating to guides. It is intended to be used as the `guide_*(theme)` argument. Because of this intent, and due to legends and axes having mutually exclusive theme elements, this function sets the elements for both simultaneously.

Usage

```
theme_guide(
  text = NULL,
  line = NULL,
  title = NULL,
  subtitle = NULL,
  text.position = NULL,
  title.position = NULL,
  subtitle.position = NULL,
  ticks = NULL,
  minor.ticks = NULL,
  mini.ticks = NULL,
  ticks.length = NULL,
  minor.ticks.length = NULL,
  mini.ticks.length = NULL,
  spacing = NULL,
  group.spacing = NULL,
  key = NULL,
  key.size = NULL,
  key.width = NULL,
  key.height = NULL,
  key.spacing = NULL,
  key.spacing.x = NULL,
  key.spacing.y = NULL,
  frame = NULL,
  byrow = NULL,
  background = NULL,
  margin = NULL,
  bracket = NULL,
  bracket.size = NULL,
  box = NULL,
  fence = NULL,
  fence.post = NULL,
  fence.rail = NULL
)
```

Arguments

`text`	An <element_text> setting both `legend.text` and `axis.text` elements.
`line`	An <element_line> setting both `legend.axis.line` and `axis.line` elements.
`title`	An <element_text> setting both `legend.title` and `axis.title` elements.
`subtitle`	An <element_text> setting both `legendry.legend.subtitle` and `legendry.axis.subtitle` elements.
`text.position`, `title.position`, `subtitle.position`	One of `"top"`, `"right"`, `"bottom"` or `"right"` setting the following elements: • `text.position`: sets only `legend.text.position`. • `title.position`: sets only `legend.title.position`. • `subtitle.position` sets both `legendry.legend.subtitle.position` and `legendry.axis.subtitle.position`
`ticks`	An <element_line> setting both `axis.ticks` and `legend.ticks` elements.
`minor.ticks`	An <element_line> setting `legendry.legend.minor.ticks` and all 6 of the `axis.ticks.minor.{r/theta/x.top/x.bottom/y.left/y.right}` elements.
`mini.ticks`	An <element_line> setting both `legendry.legend.mini.ticks` and `legendry.axis.mini.ticks` elements.
`ticks.length`, `minor.ticks.length`, `mini.ticks.length`	A [<unit[1]>][grid::unit()] setting the following elements: • `ticks.length`: sets both `legend.ticks.length` and `axis.ticks.length`. • `minor.ticks.length` sets both `axis.minor.ticks.length` and `legendry.legend.minor.ticks.` • `mini.ticks.length` sets both `legendry.axis.mini.ticks.length` and `legendry.legend.mini.ticks.length`.
`spacing`, `group.spacing`	A [<unit[1]>][grid::unit()] setting both the `legendry.guide.spacing` and `legendry.group.spacing` theme elements.
`key`	An <element_rect> setting the `legend.key` element.
`key.size`, `key.width`, `key.height`	A <unit> setting the `legend.key.size`, `legend.key.width` and `legend.key.height` elements respectively.
`key.spacing`, `key.spacing.x`, `key.spacing.y`	A [<unit[1]>][grid::unit()] setting the `legend.key.spacing`, `legend.key.spacing.x` and `legend.key.spacing.y` elements respectively.
`frame`	An <element_rect> setting the `legend.frame` element.
`byrow`	A <logical[1]> setting the `legend.byrow` element.
`background`	An <element_rect> setting the `legend.background` element.
`margin`	A <margin> setting the `legend.margin` element.
`bracket`	An <element_line> setting the `legendry.bracket` element.
`bracket.size`	A [<unit[1]>][grid::unit()] setting the `legendry.bracket.size` element.
`box`	An <element_rect> setting the `legendry.box` element.
`fence`, `fence.post`, `fence.rail`	An <element_line> setting the `legendry.fence`, `legendry.fence.post` and `legendry.fence.rail` respectively.

Value

A <theme> object that can be provided to a guide.

Examples

```
red_ticks <- theme_guide(ticks = element_line(colour = "red", linewidth = 0.5))

# Both axis and colourbar gain red ticks
ggplot(mpg, aes(displ, hwy, colour = cty)) +
  geom_point() +
  guides(
    colour = guide_colourbar(theme = red_ticks),
    x = guide_axis(theme = red_ticks)
  )
```

Index