Package ‘ggicons’

September 15, 2026

Title Icon Geometries for 'ggplot2'

Version 0.1.0

Description Provides icon geometries for 'ggplot2', using vector icon sets from the 'icons' package. Icons can be drawn as points in place of ordinary markers, styled with the usual colour, size, alpha and angle aesthetics, and mapped from discrete values or passed through directly. Icons also appear in legend keys, as fixed-position annotations, and as axis, strip and legend labels. Pictograms extend this to isotype-style unit charts, waffle/percentage charts and rating widgets, encoding a value as a grid of repeated icons.

License MIT + file LICENSE

URL https://pkg.mitchelloharawild.com/ggicons/, https://github.com/mitchelloharawild/ggicons

BugReports https://github.com/mitchelloharawild/ggicons/issues

Imports cli, ggplot2 (>= 4.0.0), grid, grImport2, icons (>= 1.0.0), rlang (>= 1.1.0), rsvg, scales, vctrs

Suggests spelling, testthat (>= 3.0.0)

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 8.0.0

Language en-GB

NeedsCompilation no

Author Mitchell O'Hara-Wild [aut, cre] (ORCID: <https://orcid.org/0000-0001-6729-7695>)

Maintainer Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Repository CRAN

Date/Publication 2026-09-15 13:10:02 UTC

Contents

`annotation_icon` *Annotate a plot with icons at fixed positions*

Description

`annotation_icon()` draws one or more icons at fixed `(x, y)` positions, the `icon` analogue of `ggplot2::annotation_custom()` / `ggplot2::annotation_raster()`: a single call that adds icons directly, without a data frame or `aes()` mapping. Unlike those, though, `annotation_icon()` *does* train the panel's position scales, since a fixed-position icon is more usefully treated like a point annotation (e.g. `ggplot2::annotate()`) than like an arbitrary grob pinned to the panel's corners, so placing one outside the data's range still expands the axes to fit it.

Usage

```
annotation_icon(
  icon,
  x,
  y,
  colour = "black",
  fill = NA,
  size = 6,
  alpha = NA,
  angle = 0,
  size.unit = "mm"
)
```

Arguments

`icon`	An icon vector from the icons package. Character ids are not resolved; pass an actual icon vector.
`x, y`	The position(s) to draw the icon(s) at, in data coordinates.
`colour, fill`	Fill colour for the icon; `fill` wins when both are set.

size	The icon's width/height, in size.unit.
alpha	Transparency, from 0 (fully transparent) to 1 (opaque).
angle	Rotation, in degrees.
size.unit	The unit in which size is interpreted, one of "mm" (default), "pt", "cm", "in", or another grid::unit()-compatible absolute unit.

Details

icon, x, y and the style arguments are all recycled to a common length, so a single call can place several icons at once (e.g. annotation_icon(icon = c(rocket, star), x = c(1, 2), y = c(1, 1))).

Value

A ggplot2 layer.

Aesthetics

annotation_icon() sets the same per-icon properties geom_icon() understands as aesthetics, but as plain arguments instead:

- icon (required): an icon vector from the icons package, an icon object (e.g. from icons::read_icon() or icons::fontawesome$solid$rocket) or an icons::icon_find() result.
- x, y: the position(s) to draw at, in data coordinates.
- colour/fill: both map to the icon's single fill colour (icons don't have a separate border); fill wins when both are set.
- alpha
- size: the icon's width/height, in size.unit (default millimetres).
- angle: rotation, in degrees.

stroke/linewidth don't apply, since icons are filled shapes, not framed ones.

See Also

geom_icon()

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))
ggplot(data.frame(x = 1:3, y = c(1, 3, 2)), aes(x, y)) +
  geom_point() +
  annotation_icon(
    icon = shapes$star,
    x = 2, y = 3, size = 12, colour = "steelblue"
  )
```

draw_key_icon *Legend key glyph: an icon*

Description

A ggplot2::draw_key() function that draws the icon itself as the legend key glyph, instead of a generic point/rect. This is geom_icon()'s default key_glyph; set key_glyph = draw_key_icon on another geom to borrow it.

Usage

```
draw_key_icon(data, params, size)
```

Arguments

data	A single row data frame containing the scaled aesthetics to display in this key
params	A list of additional parameters supplied to the geom.
size	Width and height of key in mm.

Value

A grob.

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))

# borrowed onto geom_point() so its legend key shows the mapped icon
df <- data.frame(x = 1:3, y = c(1, 3, 2), cat = c("a", "b", "c"))
ggplot(df, aes(x, y, colour = cat, icon = cat)) +
  geom_point(size = 10, key_glyph = draw_key_icon) +
  scale_icon_manual(values = c(shapes$square, shapes$circle, shapes$triangle))
```

element_icon *Theme element: draw labels as icons*

Description

A theme element parallel to ggplot2::element_text(), for use in axis.text, strip.text, legend.text, and similar theme settings. Any label matching a name in icons is drawn as that icon; every other label falls back to plain element_text() drawing.

Usage

```
element_icon(
  icons = NULL,
  ...,
  size = NULL,
  colour = NULL,
  inherit.blank = FALSE
)
```

Arguments

icons	A named `list()` of icon objects (see Details), or `NULL` (the default) to disable icon substitution.
...	Other arguments passed on to `ggplot2::element_text()`.
size	Point size for text, and the point width/height for icons.
colour	Colour for text, and the fill colour for icons.
inherit.blank	See `ggplot2::element_text()`.

Details

`icons` is a named `list()` of individual, length-1 icons, one entry per label to replace:

```
element_icon(icons = list(
  low = icons::fontawesome$solid$`battery-quarter`,
  high = icons::fontawesome$solid$`battery-full`
))
```

Value

An `element_icon` object: a subclass of `element_text` (itself a ggplot2 theme element), for use in `ggplot2::theme()`.

See Also

`geom_icon()`

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))
cyl_icons <- list(
  `4` = shapes$circle,
  `6` = shapes$square,
  `8` = shapes$triangle
)

ggplot(mtcars, aes(factor(cyl), mpg)) +
```

```
  geom_boxplot() +
  theme(axis.text.x = element_icon(icons = cyl_icons, size = 14))


# a real icon pack (here, Font Awesome) works the same way
battery <- list(
  `4` = icons::fontawesome$solid$`battery-quarter`,
  `6` = icons::fontawesome$solid$`battery-half`,
  `8` = icons::fontawesome$solid$`battery-full`
)

ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot() +
  theme(axis.text.x = element_icon(icons = battery, size = 14))
```

---

geom_icon *Icons*

Description

geom_icon() is a ggplot2::geom_point()-shaped geom that draws an icon per row at (x, y), instead of a point. Icons come from the icons package: single-fill SVG vector shapes, parsed once per unique icon and cached, then recoloured/rescaled/rotated per row at draw time.

Usage

```
geom_icon(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  size.unit = "mm",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

size.unit — The unit in which the size aesthetic is interpreted, one of "mm" (default), "pt", "cm", "in", or another grid::unit()-compatible absolute unit.

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

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

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

Value

A ggplot2 layer.

Aesthetics

geom_icon() understands the following aesthetics (icon is required):

- icon: the icon to draw, as an icon vector from the icons package: an icon object (e.g. from icons::read_icon() or icons::fontawesome$solid$rocket) or an icons::icon_find() result. Character ids are not resolved; map an actual icon vector.
- x, y
- colour/fill: both map to the icon's single fill colour (icons don't have a separate border); fill wins when both are set.
- alpha
- size: the icon's width/height, in size.unit (default millimetres).
- angle: rotation, in degrees.

stroke/linewidth don't apply: icons are filled shapes, not framed ones.

geom_icon() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- icon
- alpha → NA
- angle → 0
- colour → "black"
- fill → NA
- group → inferred

- size → 6

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

`draw_key_icon()`, `scale_icon_identity()`, `scale_icon_manual()`

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))
df <- data.frame(
  x = 1:3, y = c(1, 3, 2),
  icon = c(shapes$square, shapes$circle, shapes$triangle)
)
ggplot(df, aes(x, y, icon = icon)) +
  geom_icon(size = 10, colour = "steelblue")
```

---

geom_pictogram *Pictograms*

Description

`geom_pictogram()` draws a value as a grid of repeated icons, some fraction of them filled in and the rest left faded. The same geom produces a single-row rating widget, a waffle/percentage chart, or a growing isotype-style unit chart, depending on the grid arguments (`n`/`nrow`/`ncol`/`symbol_value`) passed to `stat_pictogram()`.

Usage

```
geom_pictogram(
  mapping = NULL,
  data = NULL,
  stat = "pictogram",
  position = "identity",
  ...,
  n = NULL,
  nrow = NULL,
  ncol = NULL,
  symbol_value = NULL,
  n_target = 20,
  flow = "row",
  size.unit = "mm",
  spacing = 0.2,
```

```
  hjust = NULL,
  vjust = NULL,
  empty.colour = "grey85",
  empty.alpha = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`n` Total slots in the grid (fixed-grid mode). `NULL` (the default) grows the grid to fit `value` instead.

`nrow`, `ncol` Grid shape. Set at most one; the other is derived to wrap the slots. Both `NULL` wraps to a single row/column; see Orientation for which.

`symbol_value` The data value one slot represents. `NULL` (the default) picks one automatically: `1` in fixed-grid mode (`value` is already a slot count, e.g. a percentage out of `n = 100`), or, in growing mode, the smallest "nice" round number (1, 2 or 5 times a power of ten) that keeps the layer's largest value to around `n_target` icons; see that argument. Set explicitly to turn auto-picking off.

`n_target` In growing mode, roughly how many icons deep the growing dimension should get for the layer's largest value, when `symbol_value` is picked automatically. Turn this up for a finer-grained chart (more, smaller icons), down for a coarser one (fewer, bigger icons). Ignored if `symbol_value` is set explicitly, or in fixed-grid mode.

`flow` Fill order: `"row"` (default) fills left-to-right then wraps to the next row; `"col"` fills top-to-bottom then wraps to the next column.

`size.unit` The unit `size` is interpreted in, as in `geom_icon()`. Left at its default (`NA`), `size` instead auto-fits: each pictogram grows or shrinks to fill its own share of the panel, the way `geom_bar()` derives bar width. Set `size` to a number to opt out to a fixed physical size.

`spacing` Gap between adjacent icons, as a fraction of `size`.

`hjust`, `vjust` Where the grid sits relative to `(x, y)`, as a fraction of its width/height: `0.5` centres it, `0`/`1` anchor an edge. Usually left at the default (`NULL`), which follows `stat_pictogram()`'s orientation: centred on an axis you mapped, anchored to the panel edge on one you left unmapped. So leaving `y` out of `aes()` draws a column chart growing up from the bottom, and leaving `x` out draws a bar growing from the left.

`empty.colour`, `empty.alpha`
: Colour and alpha for empty slots. `empty.alpha` defaults to the `alpha` aesthetic.

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

`show.legend`
: logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

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

Value

A ggplot2 layer.

Aesthetics

`geom_pictogram()` understands the following aesthetics (`value` and `icon` are required):

- `x`, `y`: the grid's anchor position. Leaving one out draws a bar/column chart instead of a fixed-position grid; see `stat_pictogram()`'s Orientation section.
- `value`: the magnitude a pictogram represents. `stat_pictogram()` turns it into a filled slot count using `symbol_value`, the amount one icon stands for. Also drives an automatic `guide_pictogram()` legend spelling that ratio out; turn it off with `guides(value = "none")`.
- `icon`: the icon drawn in every slot, filled or empty (as in `geom_icon()`).
- `colour`/`fill`: the filled-slot colour; empty slots use `empty.colour` instead.
- `alpha`, `angle`, `size`: as in `geom_icon()`.

`geom_pictogram()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- **`value`**
- **`icon`**
- `alpha` → `NA`
- `angle` → `0`
- `colour` → `"black"`
- `fill` → `NA`
- `group` → inferred
- `size` → `NA`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

`stat_pictogram()`, `geom_icon()`, `guide_pictogram()`

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))

# 10x10 waffle: 37%
ggplot(NULL, aes(x = "Proportion", value = 37)) +
  geom_pictogram(icon = shapes$square, n = 100, nrow = 10)

# single-row rating: 5 slots, 3 filled, legend switched off
ggplot(NULL, aes(y = "Rating", value = 3)) +
  geom_pictogram(icon = shapes$star, n = 5, colour = "goldenrod") +
  guides(value = "none")


# growing isotype chart: two categories, each sized to its own value
pets <- data.frame(animal = c("Cat", "Dog"), count = c(23, 15))
ggplot(pets, aes(animal, y = "Pet", value = count, icon = animal, colour = animal)) +
  geom_pictogram(nrow = 5) +
 scale_icon_manual(values = c(icons::fontawesome$solid$cat, icons::fontawesome$solid$dog))


# column chart: no y aesthetic, so icons fill like a column geometry
commutes <- data.frame(mode = c("Bicycle", "Car"), count = c(890, 1230))
ggplot(commutes, aes(mode, value = count, icon = mode, colour = mode)) +
  geom_pictogram(ncol = 5) +
 scale_icon_manual(values = c(icons::fontawesome$solid$bicycle, icons::fontawesome$solid$car)) +
  labs(value = "commuters")
```

`guide_pictogram` *Pictogram value guide*

Description

`guide_pictogram()` is the default guide for `scale_value_continuous()` (and so, automatically, for `geom_pictogram()`'s `value` aesthetic): a one-entry legend spelling out the ratio every pictogram already encodes but never states: one icon, labelled with the `symbol_value` it stands for (e.g. an icon glyph next to "= 1000"), the way an infographic isotype chart captions "each icon represents 1000 units" or a map's scale bar does for distance.

Usage

```
guide_pictogram(
  title = ggplot2::waiver(),
  symbol_value = NULL,
  label = NULL,
  icon = NULL,
```

```
  theme = NULL,
  position = NULL,
  direction = NULL,
  override.aes = list(),
  order = 0,
  ...
)
```

Arguments

title	The legend title. Defaults to the value scale's name (typically set via labs(value = ...), e.g. the units value counts).
symbol_value	Override the ratio shown, instead of reading it from the matched layer. Rarely needed.
label	Override the key's label text entirely, instead of the default "= {symbol_value}".
icon	Which icon to draw in the key. Only needed when the matched layer's icon aesthetic is *mapped* rather than fixed (as in the isotype-chart example below, where each category gets its own icon): there's no single icon to show automatically then, so the key falls back to a plain colour swatch instead of guessing a category's icon; set this to pick one deliberately.
theme	A theme object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide partially overrides, and is combined with, the plot's theme. Arguments that apply to a single legend are respected, most of which have the legend-prefix. Arguments that apply to combined legends (the legend box) are ignored, including legend.position, legend.justification.*, legend.location and legend.box.*.
position	A character string indicating where the legend should be placed relative to the plot panels. One of "top", "right", "bottom", "left", or "inside".
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical".
override.aes	A list specifying aesthetic parameters of legend key. See details and examples.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
...	ignored.

Details

Unlike a standard legend, this never shows one key per break: symbol_value is a single ratio, not something that varies over value's range, so there is only ever one row to show, and it comes from the matched geom_pictogram() layer's own symbol_value rather than from the scale's trained breaks.

Value

A ggplot2 guide.

See Also

scale_value_continuous(), geom_pictogram()

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))

# legend reads "[square] = 1": each square is one percentage point
ggplot(NULL, aes(x = "Proportion", value = 37)) +
  geom_pictogram(icon = shapes$square, n = 100, nrow = 10) +
  labs(value = "%")


# a unit chart where each icon is worth 5: the legend spells that out
pets <- data.frame(animal = c("Cat", "Dog"), count = c(23, 15))
ggplot(pets, aes(animal, value = count, icon = animal, colour = animal)) +
  geom_pictogram(symbol_value = 5) +
 scale_icon_manual(values = c(icons::fontawesome$solid$cat, icons::fontawesome$solid$dog)) +
  labs(value = "pets")
```

---

scale_icon_identity *Use icon values as-is*

Description

scale_icon_identity() is for the common case where the icon aesthetic is already mapped to icon values (an icon vector from the icons package), so the data is passed through unchanged rather than looked up in a palette. No legend is drawn by default, as with ggplot2::scale_shape_identity().

Usage

```
scale_icon_identity(
  name = ggplot2::waiver(),
  ...,
  guide = "none",
  aesthetics = "icon"
)
```

Arguments

name	The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
...	Other arguments passed on to discrete_scale() or continuous_scale()

guide	Guide to use for this scale. Defaults to "none".
aesthetics	Character string(s) naming the aesthetic(s) this scale works with, defaulting to "icon".

Details

Requesting a legend (guide = "legend") isn't supported here: ggplot2's discrete scale training only recognises character/factor data, not icon vectors. Use scale_icon_manual() instead when a legend is needed.

Value

A ggplot2 scale.

See Also

scale_icon_manual()

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))
df <- data.frame(
  x = 1:3, y = c(1, 3, 2),
  icon = c(shapes$square, shapes$circle, shapes$triangle)
)
# applied automatically for an icon-vector aesthetic; shown explicitly here
ggplot(df, aes(x, y, icon = icon)) +
  geom_icon(size = 10) +
  scale_icon_identity()
```

---

scale_icon_manual *Map discrete values to icons manually*

Description

scale_icon_manual() maps a discrete data value (e.g. a category column) to an icon, using a manually-specified lookup, the icon analogue of ggplot2::scale_shape_manual(). There's no automatic discrete palette for icons, so values is required.

Usage

```
scale_icon_manual(..., values, breaks = ggplot2::waiver(), na.value = NA)
```

Arguments

...	Arguments passed on to discrete_scale limits One of: • NULL to use the default scale values • A character vector that defines possible values of the scale and their order • A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation. drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE includes the levels in the factor. Please note that to display every level in a legend, the layer should use show.legend = TRUE. na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE. name The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted. minor_breaks One of: • NULL for no minor breaks • waiver() for the default breaks (none for discrete, one minor break between each major break for continuous) • A numeric vector of positions • A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions. labels One of the options below. Please note that when labels is a vector, it is highly recommended to also set the breaks argument as a vector to protect against unintended mismatches. • NULL for no labels • waiver() for the default labels computed by the transformation object • A character vector giving labels (must be same length as breaks) • An expression vector (must be the same length as breaks). See ?plotmath for details. • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation. guide A function used to create a guide or its name. See guides() for more information. call The call used to construct the scale for reporting messages. super The super class to use for the constructed scale
values	An icon vector from the icons package, the same length as the number of levels to map. Unlike most scale_*_manual() counterparts, values can't be named by level, since icon vectors don't support names, so it's matched positionally against the sorted data levels; pass limits to map against a different order (breaks only controls the legend, not this matching).

breaks	One of: • NULL for no breaks • waiver() for the default breaks (the scale limits) • A character vector of breaks • A function that takes the limits as input and returns breaks as output
na.value	The aesthetic value to use for missing (NA) values

Value

A ggplot2 scale.

See Also

scale_icon_identity()

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))
df <- data.frame(
  x = 1:3, y = c(1, 3, 2),
  type = c("a", "b", "c")
)
ggplot(df, aes(x, y, icon = type)) +
  geom_icon(size = 10) +
  scale_icon_manual(values = c(shapes$square, shapes$circle, shapes$triangle))
```

---

scale_value_continuous

Continuous scale for the pictogram value aesthetic

Description

scale_value_continuous() registers value (as mapped in geom_pictogram()) as a real trained aesthetic, purely so it can carry a default guide (see guide_pictogram()). It never rescales value itself: symbol_value slot counts need the real data value, so the palette is the identity function.

Usage

```
scale_value_continuous(
  name = ggplot2::waiver(),
  breaks = ggplot2::waiver(),
  labels = ggplot2::waiver(),
  limits = NULL,
  guide = "pictogram",
  ...
)
```

Arguments

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

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

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`guide` The value legend's guide; defaults to `guide_pictogram()`. Set to `"none"` (or pass `show.legend = FALSE` to `geom_pictogram()`) to hide it.

`...` Arguments passed on to `continuous_scale`

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`oob` One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::censor()`) replaces out of bounds values with `NA`.
- `scales::squish()` for squishing out of bounds values into range.
- `scales::squish_infinite()` for squishing infinite values into range.

`na.value` Missing values will be replaced with this value.

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

`super` The super class to use for the constructed scale

Details

Picked up automatically for `aes(value = ...)`, so most plots never need to call it directly. Exported for overriding `guide` or setting explicit `breaks`.

Value

A ggplot2 scale.

See Also

`guide_pictogram()`, `geom_pictogram()`

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))

# applied automatically; shown explicitly here just to relabel the legend
ggplot(NULL, aes(x = "Proportion", value = 37)) +
  geom_pictogram(icon = shapes$square, n = 100, nrow = 10) +
  scale_value_continuous(name = "%")
```

---

stat_pictogram *Pictogram grid layout*

Description

`stat_pictogram()` turns one row of data (an anchor position and a `value`) into one row per icon slot in a grid, ready for `geom_pictogram()` to draw. It's the counting/layout half of a pictogram.

Usage

```
stat_pictogram(
  mapping = NULL,
  data = NULL,
  geom = "pictogram",
  position = "identity",
  ...,
  n = NULL,
  nrow = NULL,
  ncol = NULL,
  symbol_value = NULL,
  n_target = 20,
  flow = "row",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	Override the default connection between stat_pictogram() and geom_pictogram(). For more information about overriding these connections, see ggplot2::layer().
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the

position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

n — Total slots in the grid (fixed-grid mode). NULL (the default) grows the grid to fit value instead.

nrow, ncol — Grid shape. Set at most one; the other is derived to wrap the slots. Both NULL wraps to a single row/column; see Orientation for which.

symbol_value — The data value one slot represents. NULL (the default) picks one automatically: 1 in fixed-grid mode (value is already a slot count, e.g. a percentage out of n = 100), or, in growing mode, the smallest "nice" round number (1, 2 or 5 times a power of ten) that keeps the layer's largest value to around n_target icons; see that argument. Set explicitly to turn auto-picking off.

n_target — In growing mode, roughly how many icons deep the growing dimension should get for the layer's largest value, when symbol_value is picked automatically. Turn this up for a finer-grained chart (more, smaller icons), down for a coarser one (fewer, bigger icons). Ignored if symbol_value is set explicitly, or in fixed-grid mode.

flow — Fill order: "row" (default) fills left-to-right then wraps to the next row; "col" fills top-to-bottom then wraps to the next column.

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

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

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

Value

A ggplot2 layer.

Grid sizing

Two modes, chosen by whether `n` (or both `nrow` and `ncol`) is set:

- **Fixed grid** (`n`, or both `nrow`/`ncol`, given): the grid always has `n` slots; `round(value / symbol_value)` of them are filled (capped at `n`), the rest empty. This is the waffle-chart/rating-widget shape: proportions vary, the grid footprint doesn't.
- **Growing grid** (`n`, `nrow` and `ncol` all left `NULL`, or only one of `nrow`/`ncol` given): the grid has exactly `round(value / symbol_value)` slots, all filled. An isotype/unit-chart shape where more icons *is* the value, so the footprint grows with it.

At most one of `nrow`/`ncol` needs setting; the other wraps to fit. With neither set, the grid defaults to a single row, unless `x`/`y` asks for a bar/column shape instead; see Orientation below.

Orientation

`x` and `y` are the grid's anchor, but only `value` is actually required. Map both and you get a fixed-position grid, centred on `(x, y)`: a waffle chart or rating widget. Leave one unmapped and that becomes the growth axis of a bar-style stack instead, anchored to that edge of the plot panel. `aes(x = category, value = count)` draws a column chart, one growing column per category, up from the bottom of the panel; `aes(y = category, value = count)` draws a horizontal bar, growing right from the left of the panel.

See Also

`geom_pictogram()`

Examples

```
library(ggplot2)

# bundled placeholder shapes; a real pack (e.g. icons::fontawesome) works the same
shapes <- icons::icon_set(system.file("icons", package = "ggicons"))

# only needed directly when pairing with a different geom
ggplot(NULL, aes(x = "Proportion", value = 37)) +
  stat_pictogram(geom = "pictogram", icon = shapes$square, n = 100, nrow = 10)
```

Index