Package 'paintr'

August 6, 2026

Title Create Graphics of 'R' Data Structures

Version 0.0.1

Description Draws teaching diagrams of the contents of common 'R' data structures (vectors, matrices, data frames, lists, and arrays) so their shape and values can be read at a glance. Two backends render the same picture, with base 'R' graphics through the paint_*() functions and 'ggplot2' through the gpaint_*() functions. The label under each cell is the expression you would type to reach that cell, so every diagram doubles as a lesson in how to subset and index the object it draws.

URL https://github.com/coatless-rpkg/paintr, https://r-pkg.thecoatlessprofessor.com/paintr/

BugReports https://github.com/coatless-rpkg/paintr/issues

Depends R (>= 4.2.0)

License GPL (>= 2)

Encoding UTF-8

Imports graphics, grDevices, grid

Suggests downlit, evaluate, ggplot2, knitr, quarto, svglite, testthat (>= 3.0.0), withr

Config/testthat/edition 3

VignetteBuilder quarto

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author James Joseph Balamuta [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-2826-8458>)

Maintainer James Joseph Balamuta <james.balamuta@gmail.com>

Repository CRAN

Date/Publication 2026-08-06 13:10:14 UTC

Contents

highlight_data 2
paint_array 5
paint_data_frame 10
paint_list 15
paint_matrix 20
paint_size 24
paint_vector 26

Index **30**

`highlight_data` *Highlight data*

Description

Build a logical mask that marks the cells to highlight in a subsequent `paint_*()` / `gpaint_*()` call.

Usage

```
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'numeric'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'integer'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'character'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'logical'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'complex'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'factor'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'Date'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'POSIXct'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'vector'
```

```
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'matrix'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'array'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'table'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'list'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## S3 method for class 'data.frame'
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

## Default S3 method:
highlight_data(x, rows = NULL, columns = NULL, locations = NULL, ...)

highlight_rows(x, rows = NULL)

highlight_columns(x, columns = NULL)

highlight_locations(x, locations = NULL)
```

Arguments

x	A vector, factor, matrix, or data frame.
rows	A vector of valid row locations, given either as integer indices, as row names, or as a logical mask.
columns	A vector of valid column locations, given either as integer indices, as column names, or as a logical mask.
locations	An m by 2 matrix with points listed in row, column format for a 2D object or a vector of integer indices in a 1D format.
...	Additional values (not used)

Value

A logical matrix or vector with the required rows and/or columns or points set to `TRUE`. All other values are given as `FALSE`. The result always has the same shape as `x`: a logical vector of `length(x)` for 1D structures, and a logical matrix of `dim(x)` for matrices and data frames.

Supported structures

Methods exist for `numeric`, `integer`, `character`, `logical`, `complex`, `factor`, `Date`, `POSIXct`, `matrix`, `array`, `table`, `data.frame`, and a bare `list`. Anything else is an error.

A **list**'s mask is positions by elements, because that is how `paint_list()` draws it: `columns` select elements (by name, `names(x)`, as well as by number) and `rows` select positions within them. It is as deep as the deepest element. An **empty** list is refused, because `paint_list()` cannot draw one either.

A list carrying a class – `as.POSIXlt(Sys.time())`, an `lm`, a `t.test()` result – is **not** a list for these purposes and is refused, exactly as `paint_list()` refuses it. `is.list()` is TRUE for all of them, which is why the question is never asked that way; see the note on `is_paint_list()`.

An **array** or a **table** of rank two or more is masked with its own full shape: the mask of `Titanic` is a 4 x 2 x 2 x 2 logical array, because that is what `paint_array()` draws. `rows` and `columns` select on the first two axes – the two a *block* is made of – and mark that row (or column) of **every** slice; `locations` takes a **full coordinate per point**, one column per dimension, and reaches a single cell. Rank one is refused: nothing draws it.

The governing invariant: **if a painter can draw a structure,** `highlight_data()` **must be able to mask it** – and a *wrong* mask is worse than an error, so a structure no painter accepts is refused outright rather than reshaped into a mask nothing could consume. A 2D `table` is drawn by `paint_matrix()`, so it is masked here; `Titanic` is drawn by `paint_array()`, so it is masked here. An n-D array used to *stop* for exactly the same reason it is now *masked* – the invariant never changed, the set of things a painter can draw did, and the two halves moved together.

There is deliberately no reliance on a `vector` method being *dispatched*: `inherits(letters, "vector")` is `FALSE`, so `highlight_data.vector()` is never selected by `UseMethod()` for an atomic vector. The atomic methods are therefore fanned out explicitly, and each one forwards to `highlight_data.vector()` by a direct call. `array` and `table` are the same trap one type over: an atomic array dispatches on `c("array", "integer", "numeric")`, so without an `array` method it would land on `highlight_data.integer()` and be silently flattened, and a `table` dispatches on `"table"` alone, so without a `table` method it would land on `highlight_data.default()` and be refused despite being paintable.

See Also

The painters that consume a mask, such as `paint_matrix()` and `paint_data_frame()`.

Examples

```
## 2D Highlighting for Matrices ----
# Example data
x <- matrix(1:12, nrow = 4)

# Highlight points using a row, column pairing
locations <- rbind(
  c(1, 3),
  c(2, 2),
  c(4, 1)
)
highlight_locations(x, locations)

# Highlight entries only in the 1st and 3rd rows.
highlight_rows(x, rows = c(1, 3))

# Highlight entries only in the first two rows:
```

```
highlight_rows(x, rows = 1:2)

# Highlight entries in the last column
highlight_columns(x, columns = ncol(x))

# Highlight entries in the first column
highlight_columns(x, columns = 1)

# Highlight entries in the first column or first row.
highlight_data(x, rows = 1, columns = 1)

## 1D Highlighting for Vectors ----
vec <- c(3, NA, -1, 2, NaN, Inf, 42)
highlight_data(vec, locations = c(2, 4, 6))

# Character and logical vectors work the same way.
highlight_locations(letters[1:5], c(2, 4))
highlight_data(c(TRUE, FALSE, TRUE))

## 2D Highlighting for Data Frames ----
# Columns may be named instead of numbered.
highlight_columns(iris[1:5, ], "Sepal.Width")
highlight_rows(iris[1:5, ], rows = c(1, 3))
highlight_locations(iris[1:5, ], rbind(c(1, 1), c(2, 5)))
```

`paint_array` *Visualize Data Inside of an Array*

Description

Generate a graph showing the contents of an array of any rank, laid out the way `print()` lays one out: a block per slice, each under the `, , Male, Child` subscript title that names it.

Usage

```
paint_array(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = NULL,
  max_cols = NULL,
  max_slices = 4L,
  slices_per_row = NULL,
```

```
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_dimnames = "all",
  max_name_chars = 8L,
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)

gpaint_array(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = NULL,
  max_cols = NULL,
  max_slices = 4L,
  slices_per_row = NULL,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_dimnames = "all",
  max_name_chars = 8L,
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)
```

Arguments

`data`	An `array`. A `matrix` and a `table` are arrays and are drawn as such. Rank 1 is not drawn.
`show_indices`	Display indices based on location. A character vector, so several kinds of index can be asked for at once: `"none"` (the default), `"cell"` (`[1, 2, 3]`, inside the cell), `"row"` (`[1, , 3]`, to the left of each block), `"column"` (`[, 2, 3]`, above each block), or `"all"`. Every one of them is a subscript expression that RUNS – see the section above.
`highlight_area`	Logical array the same shape as `data`, marking the cells to fill. Build it with `highlight_data()`, which masks an array of any rank. A length-one logical is recycled. Default: `NULL`, which highlights nothing.

highlight_color
: Color to use to fill the background of a cell.

graph_title
: Title to appear in the upper left hand corner of the graph.

graph_subtitle
: Subtitle to appear immediately under the graph title. NULL (the default) describes the data: its full shape (4 x 2 x 2 x 2) and its class. NA or "" draws no subtitle.

sigfig
: Significant digits drawn in black. Digits past the sigfig-th are drawn in grey; nothing is discarded. Must be in 1:15.

subtle_digits
: Which digits are drawn grey. "insignificant" (the default) greys everything past the sigfig-th significant digit; "rounded" greys only digits the rendering actually lost; "none" draws everything black.

max_chars
: Strings longer than this are truncated with an ellipsis.

max_rows, max_cols
: Elide the middle of each *block* when the array has more rows or columns than this. NULL (the default) takes the cap that suits the RANK of the thing being drawn: 10 by 8 for an array of rank 3 or more – tighter than paint_matrix()'s 20 by 15, because the picture is several blocks wide and they share the device between them – and 20 by 15 for a rank-2 array, **because a rank-2 array is a matrix and is drawn as one**. Passing NULL rather than 10 here is the whole of what makes paint_array(m) and paint_matrix(m) the same picture for a matrix of any size; a hard 10 would elide a 12x10 matrix that paint_matrix() draws whole. A number overrides both cases. As always, the decision is made on the dimensions alone, with no device consulted.

max_slices
: Elide the slice axes when the array has more slices than this along either of them, drawing a "..." block in place of the hidden ones. Default: 4.

  The picture is max_slices matrices wide, so this is the knob that costs the most: at 4, Titanic and HairEyeColor draw whole and UCBAdmissions draws three of its six departments and says # 3 more slices. Raise it (or set show_all) to see them all, at a smaller size.

slices_per_row
: Wrap the slice blocks into a grid this many to a row, instead of the array's own layout. NULL (the default) is today's picture exactly: a 3-D array in one row, a 4-D-or-higher array in its natural grid. A positive whole number takes the slices in their natural order – the order the block titles enumerate – and wraps them onto as many rows as it takes, like ggplot2::facet_wrap(). Every block keeps its FULL, honest slice title (, , 5 stays , , 5), so wrap position is reading order, never a claim about the data: it cannot make a 3-D array read as a 4-D one. It composes with max_slices – a wrapped array still elides and draws its "..." block – and a value at least the slice count simply draws one row. A matrix has one slice, so it is a no-op there.

show_all
: Draw every cell, however small the text becomes. Warns when the text falls below the legibility floor.

fontsize
: Font size in points. NULL (the default) fits the text to the device.

family
: Font family. "mono" by default.

palette
: Colour palette for the drawing. One of "mint" (the default), "slate", "warm", or "classic" (the original look). Defaults to the "paintr.palette" option when unset. A named list of colours is also accepted.

`show_dimnames` Which of the array's `dimnames()` to draw. A character vector, because `dimnames()` is a *list* with one slot per axis: `"none"`, `"row"`, `"column"`, `"slice"`, or `"all"` (the default).

`"slice"` is what puts the names in the block titles: with it, a block of `Titanic` is titled `, , Child, No`; without it, `, , 1, 1`. An index lane the caller asks for **wins** the axis it names, exactly as it does for a matrix. Each name is truncated at `max_name_chars`, exactly as a column name is.

The title is the block's **subscript**, and not `print()`'s full subscript line. When an array's `dimnames()` are themselves *named* – `Titanic`'s are (`Class`, `Sex`, `Age`, `Survived`) – `print()` writes `, , Age = Child, Survived = No` where this writes `, , Child, No`. The axis names are omitted because the title is fitted against the width of the block it spans, and the long form costs `Titanic` 48% of its font (15.7pt to 8.2pt at 7x5in); rank-3 tables such as `HairEyeColor` pay nothing for it, so the cost lands entirely on the largest arrays. `, , Child, No` is still the accessor: `Titanic[, , 1, 1]` returns the block it sits over.

`max_name_chars` Longest a *column* name may be drawn before it is truncated. Default: `8`. A matrix is one formatting unit, so every column of it is as wide as the widest: a long column name widens every cell in the picture. Row names sit in a gutter of their own, cost the values nothing, and are truncated at `max_chars` instead.

`highlight_rows`, `highlight_columns`, `highlight_locations`

Shorthand for `highlight_area`, built for you with `highlight_data()`: `highlight_rows` and `highlight_columns` mark that row or column of EVERY slice (the first two axes a block is made of), and `highlight_locations` takes a full coordinate per point – one column per dimension – to reach a single cell. So `highlight_rows = 1` is exactly `highlight_area = highlight_data(data, rows = 1)`. Give several at once to fill their union. Supplying `highlight_area` together with any of these is an error. Default: `NULL`.

Details

`paint_array()` draws on the current base graphics device. `gpaint_array()` returns a `ggplot` object.

A matrix is an array, and this draws it. `is.array(matrix(1:4, 2))` is TRUE, and a rank-two array is simply the case of this picture with one block and no title to put over it – so `paint_array(m)` and `paint_matrix()` draw the same picture of the same matrix, down to the cell, because they run the same code.

Value

`paint_array()` invisibly returns the resolved cell table. See `paint_matrix()` for its components. `gpaint_array()` returns a `ggplot` object.

The label under the cell is the expression you type

This is the promise the package is built on, and an array is where it is easiest to break. On a 3-D array, `a[1, ]` and `a[2, 3]` are not "shorthand" – they are **errors**:

```
a <- array(1:24, c(2, 3, 4))
```

```
a[1, ]      # Error in a[1, ] : incorrect number of dimensions
a[2, 3]     # Error in a[2, 3] : incorrect number of dimensions
a[2, 3, 4]  # 24
```

So every index this picture draws carries the array's full subscript arity, with the slice filled in from the block the label sits in: the row gutter reads `[1, , 3]`, the column lane `[, 2, 3]`, and the in-cell index `[1, 2, 3]`. Copy any label off the picture, type it, and it returns the thing it was drawn beside. A 4-D array reads `[1, 2, 3, 2]`. A matrix reads `[1, 2]`, exactly as `paint_matrix()` does.

`show_indices = "all"` is the call this section exists for.

The whole array is one formatting unit

A `1e15` in the third slice flips the *first* slice into scientific notation, and the same number is drawn the same way in every block. An array is one homogeneous object – `a[1, 1, 1]` and `a[2, 3, 4]` measure the same thing in the same units – and R's own `print()` formats it to one common width across every slice. This is the opposite of a data frame, whose columns are separate *variables* and are therefore formatted separately.

The slices are not panels

They are blocks in a single cell table, and that is not an implementation detail. The font size is fitted to one table, so every block in the picture is drawn at the same size. Laying the slices out as real graphics panels – `par(mfrow =)`, `layout()`, or one grid viewport each – fits a *separate* font to each one, and three slices of one array come out at 24, 12.4 and 23.1 points: the same number, drawn at half the size, two inches to the left. Equal panels, unequal fonts.

See Also

Other painters: `paint_data_frame()`, `paint_list()`, `paint_matrix()`, `paint_size()`, `paint_vector()`

Examples

```
# Base graphics

# A 3-D array lays its slabs out in one line.
paint_array(array(1:24, c(2, 3, 4)))

# Wrap the same slabs three to a row instead. Every block keeps its true slice
# title, so this is reading order, not a fourth dimension.
paint_array(array(1:36, c(2, 3, 6)), slices_per_row = 3, show_all = TRUE)

# A 4-D contingency table is a real grid of blocks: across is the third
# dimension, down is the fourth.
paint_array(Titanic)

# The accessor lesson. Every label is an expression that runs: the value under
# `[2, 3, 2]` is exactly what `a[2, 3, 2]` returns.
paint_array(array(1:12, c(2, 3, 2)), show_indices = "all")

# A matrix IS an array, and this draws it -- the same picture paint_matrix() does.
paint_array(matrix(1:6, nrow = 2))
```

```
# Highlight a row of every slice.
paint_array(Titanic, highlight_area = highlight_data(Titanic, rows = 1))

# Six departments, three drawn: the gap is always drawn, on the slice axis too.
paint_array(UCBAdmissions)


# ggplot2 graphics ----

gpaint_array(array(1:24, c(2, 3, 4)))

# Wrap six slabs into a 2x3 grid, in reading order.
gpaint_array(array(1:36, c(2, 3, 6)), slices_per_row = 3, show_all = TRUE)

gpaint_array(HairEyeColor)
```

`paint_data_frame` *Visualize Data Inside of a Data Frame*

Description

Generate a graph showing the contents of a data frame.

Usage

```
paint_data_frame(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 10L,
  max_cols = 10L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_types = TRUE,
  show_names = TRUE,
  show_rownames = NULL,
  name_align = c("center", "left", "right"),
```

```
  type_align = c("center", "left", "right"),
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)

gpaint_data_frame(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 10L,
  max_cols = 10L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_types = TRUE,
  show_names = TRUE,
  show_rownames = NULL,
  name_align = c("center", "left", "right"),
  type_align = c("center", "left", "right"),
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)

paint_df(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 10L,
  max_cols = 10L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
```

```
  show_types = TRUE,
  show_names = TRUE,
  show_rownames = NULL,
  name_align = c("center", "left", "right"),
  type_align = c("center", "left", "right"),
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)

gpaint_df(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 10L,
  max_cols = 10L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_types = TRUE,
  show_names = TRUE,
  show_rownames = NULL,
  name_align = c("center", "left", "right"),
  type_align = c("center", "left", "right"),
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)
```

Arguments

`data`	An object that has the class of `data.frame`.
`show_indices`	Display indices based on location. A character vector, so several kinds of index can be asked for at once. Values are: `"none"`: no indices, `"cell"`: matrix cell indices `[i, j]`, `"row"`: row indices `[i, ]` to the left of the matrix, `"column"`: column indices `[, j]` above the matrix, and `"all"`: row, column, and cell indices together. Default: `"none"`. Each value switches on its own lane, so `c("row", "column")` draws the row *and* the column indices but no cell indices, and `"all"` is the same as `c("cell", "row", "column")`. Combining `"none"` with anything else is contradictory, and the other values win: `c("none", "row")` draws row indices. An unknown value

is an error, not a silent no-op.

`highlight_area` Logical matrix the same shape as `data`, marking the cells to fill. A length-one logical is recycled. Default: `NULL`, which highlights nothing.

`highlight_color`
Color to use to fill the background of a cell.

`graph_title` Title to appear in the upper left hand corner of the graph.

`graph_subtitle` Subtitle to appear immediately under the graph title. `NULL` (the default) describes the data: its dimensions and its class. `NA` or `""` draws no subtitle; any other string is drawn as given. The default reports the dimensions of the data itself, not of the drawing, so an elided matrix still reports all of its rows.

`sigfig` Significant digits drawn in black. Digits past the `sigfig`-th are drawn in grey; nothing is discarded. Must be in `1:15`.

`subtle_digits` Which digits are drawn grey. `"insignificant"` (the default) greys everything past the `sigfig`-th significant digit; `"rounded"` greys only digits the rendering actually lost; `"none"` draws everything black.

`max_chars` Strings longer than this are truncated with an ellipsis.

`max_rows`, `max_cols`
Elide the middle of the data frame when it has more rows or columns than this. Default: `10`, because a data frame's columns are wide.

`show_all` Draw every cell, however small the text becomes. Warns when the text falls below the legibility floor.

`fontsize` Font size in points. `NULL` (the default) fits the text to the device.

`family` Font family. `"mono"` by default.

`palette` Colour palette for the drawing. One of `"mint"` (the default), `"slate"`, `"warm"`, or `"classic"` (the original look). Defaults to the `"paintr.palette"` option when unset. A named list of colours is also accepted.

`show_types` Draw the type-tag row (`<dbl>`, `<chr>`, ...) under the column names. Default: `TRUE`.

`show_names` Draw the column-name row. Default: `TRUE`.

`show_rownames` Draw the row names, in a gutter to the left of the frame. `NULL` (the default) draws them when the frame has names of its own: `mtcars` does ("Mazda RX4"), and `iris` does not – a gutter of `1, 2, 3` is the row's position, not data about it. `TRUE` draws whatever `rownames()` returns; `FALSE` draws nothing.

The gutter is what teaches the classic confusion: the car's name is **not** a column of `mtcars`, which is why `mtcars$name` is `NULL`. `show_indices = "row"` wins the same lane, and draws `[1, ]` instead.

`name_align` How the column names sit over their column: `"center"` (the default), `"left"` or `"right"`.

`type_align` How the type tags sit over their column: `"center"` (the default), `"left"` or `"right"`. Independent of `name_align`.

```
Both control the label lanes *only*. The values keep
their own alignment whatever the labels are told to do:
a numeric column stays anchored on its decimal point, a
 character column stays left, a logical column stays
      right.
```

highlight_rows, highlight_columns, highlight_locations
: Shorthand for highlight_area: instead of building a mask, name the rows, columns, or cell locations to fill and the mask is built for you with highlight_data(). So highlight_rows = 1 is exactly highlight_area = highlight_rows(data, 1), and the object need not be named twice. Give several at once to fill their union. Supplying highlight_area together with any of these is an error. Default: NULL.

Details

paint_data_frame() draws on the current base graphics device. gpaint_data_frame() returns a ggplot object. paint_df() and gpaint_df() are aliases.

Each column is its own formatting unit, because a data frame's columns are independent variables: a 1e15 in one column will not flip another column into scientific notation. (A matrix is the opposite – one unit for the whole thing, so that the same value looks identical in every cell.)

Value

paint_data_frame() invisibly returns the resolved cell table. See paint_matrix() for its components. gpaint_data_frame() returns a ggplot object.

The ggplot object is a shell

gpaint_data_frame() returns a real ggplot object – + theme(), ggsave(), print() and knitr chunks all work – but its panel is drawn *entirely* by a custom grid grob, held in a single annotation_custom() over a meaningless 0..1 coordinate system. There is no aes(), no geom and no scale carrying any meaning, so:

- ggplot_build() sees an empty layer.
- + scale_fill_*(), + scale_x_*() and friends have no effect on the drawing. Use highlight_area and highlight_color to fill cells.
- + geom_point() would draw onto the 0..1 coordinate system, not onto the cells.

This is not a shortcut around ggplot2. The cell text is fitted to the device at *draw* time, which no geom can do, because a layer is built long before the device size is known; and every number is drawn as two spans in two colors, which geom_text() cannot do at all. A custom grob is the only mechanism that can do either.

See Also

Other painters: paint_array(), paint_list(), paint_matrix(), paint_size(), paint_vector()

Examples

```
# Base graphics

paint_data_frame(head(iris, 5))

# The type row can be turned off.
paint_data_frame(head(mtcars, 4), show_types = FALSE)
```

```
# The two label lanes align independently. The values do not move.
paint_data_frame(head(iris, 5), name_align = "left", type_align = "right")

# A frame with row names of its own draws them in a gutter: the car's name is
# not a column of mtcars, which is why `mtcars$name` is NULL. iris has no such
# names, and draws no gutter -- a lane of 1, 2, 3 is a position, not data.
paint_data_frame(head(mtcars, 4))

# Long frames elide their middle and say so.
paint_data_frame(iris)

# Highlight a column by name.
paint_data_frame(
  head(iris, 5),
  highlight_area = highlight_columns(head(iris, 5), "Sepal.Width")
)

# ggplot2 graphics ----

gpaint_data_frame(head(iris, 5))

gpaint_df(head(mtcars, 4))
```

---

paint_list *Visualize Data Inside of a List*

Description

Generate a graph showing the contents of a list.

Usage

```
paint_list(
  data,
  summarise = FALSE,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 10L,
  max_cols = 8L,
  show_all = FALSE,
  fontsize = NULL,
```

```
  family = "mono",
  palette = NULL,
  show_types = TRUE,
  show_names = TRUE,
  name_align = c("center", "left", "right"),
  type_align = c("center", "left", "right"),
  gap = 0,
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)

gpaint_list(
  data,
  summarise = FALSE,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 10L,
  max_cols = 8L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_types = TRUE,
  show_names = TRUE,
  name_align = c("center", "left", "right"),
  type_align = c("center", "left", "right"),
  gap = 0,
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)
```

Arguments

data	A bare `list`. A list carrying a class – `as.POSIXlt(Sys.time())`, an `lm`, a `t.test()` result – is refused: `is.list()` is TRUE for all of them, but a datetime drawn as eleven ragged columns of `sec`, `min`, `hour`, ... is a wrong picture, not a picture of a list.
summarise	Draw every element as ONE cell saying what it is (`<int [3]>`, `<chr [1]>`) instead of one cell per value. Default: `FALSE`. It is the structure of the list at a glance, and it is what a 40-element list wants.

show_indices
: Draw `[[j]][i]` under each value (`"cell"` or `"all"`), or nothing (`"none"`, the default).

  `l[[2]][3]` is the expression students get wrong most often – `l[2]` is a list, `l[[2]]` is the vector, `l[[2]][3]` is the value – and this lane prints it under the number it returns. A summarised element gets `[[j]]`, because that is the accessor that returns the element itself.

  There is deliberately **no** `[i]` gutter down the left. In a data frame, reading across a row is the whole point: `df[[1]][2]` and `df[[2]][2]` are one record. **In a list that is false**, and a lane that teaches it would be a lane that teaches a falsehood. Elision sharpens the point: because it is asked of each element separately, a drawn row of a long list can hold `a[9]` beside `c[7]` – so a gutter could not even name the positions it sat beside, let alone claim they were a record.

highlight_area
: Logical matrix marking the cells to fill: POSITIONS by ELEMENTS, as deep as the deepest element. Build it with `highlight_columns()` (which selects elements, by name as well as by number) and `highlight_rows()` (which selects positions within them). A length-one logical is recycled. Default: `NULL`, which highlights nothing.

highlight_color
: Color to use to fill the background of a cell.

graph_title
: Title to appear in the upper left hand corner of the graph.

graph_subtitle
: Subtitle to appear immediately under the graph title. `NULL` (the default) describes the data: how many elements, the range of their lengths, and the class. `NA` or `""` draws no subtitle.

sigfig
: Significant digits drawn in black. Digits past the `sigfig`-th are drawn in grey; nothing is discarded. Must be in `1:15`.

subtle_digits
: Which digits are drawn grey. `"insignificant"` (the default) greys everything past the `sigfig`-th significant digit; `"rounded"` greys only digits the rendering actually lost; `"none"` draws everything black.

max_chars
: Strings longer than this are truncated with an ellipsis.

max_rows
: Elide the middle of an element longer than this. Default: `10`. It is asked of each element SEPARATELY, so a length-2 element beside a length-40 one draws no `"..."` – it is hiding nothing.

max_cols
: Elide the middle of the list when it has more elements than this. Default: `8`, because a list's columns are as wide as a name.

show_all
: Draw every cell, however small the text becomes. Warns when the text falls below the legibility floor.

fontsize
: Font size in points. `NULL` (the default) fits the text to the device.

family
: Font family. `"mono"` by default.

palette
: Colour palette for the drawing. One of `"mint"` (the default), `"slate"`, `"warm"`, or `"classic"` (the original look). Defaults to the `"paintr.palette"` option when unset. A named list of colours is also accepted.

show_types
: Draw the type-tag row (`<dbl>`, `<chr>`, ...) under the element names. Default: `TRUE`.

`show_names` Draw the element-name row. Default: `TRUE`. A named element is labelled `$a`; an unnamed one is labelled `[[2]]`, exactly as `print()` does it. (This per-element fallback is the one place in the package where a label lane holds two kinds of thing at once – a list is the only structure whose parts are named one at a time.)

`name_align` How the element names sit over their column: `"center"` (the default), `"left"` or `"right"`.

`type_align` How the type tags sit over their column. Independent of `name_align`. Neither moves the values, which keep their own alignment.

`gap` Empty space to insert BETWEEN adjacent element columns, in units of one column width. Default: `0`, which draws the columns tight – so a rectangular list stays pixel-identical to the equivalent data frame. `gap = 0.5` inserts half a column of background between each pair of elements, `gap = 1` a full column, to emphasise that a list is a BAG OF INDEPENDENT VECTORS and not a 2-D grid. The space is empty background, never a cell; it goes only between columns, not before the first, after the last, or down the index gutter. A spaced list draws no block outline, because a single border across the gaps would imply the rectangle the gap is there to break; see the outline section.

`highlight_rows`, `highlight_columns`, `highlight_locations`
Shorthand for `highlight_area`, built for you with `highlight_data()`: `highlight_columns` selects ELEMENTS (by name or number), `highlight_rows` selects POSITIONS within them, and `highlight_locations` reaches individual cells. So `highlight_columns = "c"` is exactly `highlight_area = highlight_columns(data, "c")`. Give several at once to fill their union. Supplying `highlight_area` together with any of these is an error. Default: `NULL`.

Details

This picture does not draw NESTING. A sublist is drawn as a single grey token saying what it is – `<list [2]>` – and its contents are not drawn at all. Nesting is the hard part of lists (`l$a$b`, "why is my result a list of lists"), and this painter refuses to teach it rather than teach it badly: a geometric sub-grid inside a cell would not extend the layout engine, it would delete it (every column has ONE width, indexed by integer column, and a fractional row is silently truncated). The refusal is deliberate and it is stated here rather than discovered in a picture. The same is true of any element that is not a plain one-dimensional vector: a matrix element draws as `<int [2 x 2]>`, a data frame element as `<df [5 x 3]>`.

`paint_list()` draws on the current base graphics device. `gpaint_list()` returns a `ggplot` object.

A data frame IS a list whose elements happen to share a length. That is what this picture is for. Draw `paint_data_frame(data.frame(a = 1:3, b = 4:6))` and `paint_list(list(a = 1:3, b = 4:6))` side by side and they are the same picture, down to the heavy border around the block and every cell inside it – only the header (`$a`, not `a`) and the subtitle say which is which. Then draw `paint_list(list(a = 1:3, b = 4))` and watch the rectangle break. The shared length is the only thing a data frame adds, and these two pictures are the proof.

Each element is its own formatting unit, exactly as a data frame's column is: a `1e15` in one element will not flip another into scientific notation.

Value

`paint_list()` invisibly returns the resolved cell table. See `paint_matrix()` for its components. `gpaint_list()` returns a `ggplot` object.

The outline is the lesson

Every painter draws a heavy border around the block of values when that block is a rectangle – and **a list's block is a rectangle exactly when its elements share a length, which is exactly when the list could have been a data frame.** So the border is not decoration on this picture; it is the fact being taught. Give the elements a shared length and the rectangle closes, around the very same cells the equivalent `paint_data_frame()` closes it around; take the shared length away and the rectangle breaks.

A ragged list therefore draws no outline, and that is a decision rather than an omission. The heavy border would be the BOUNDING BOX of the drawn cells, so on a 4/1/3 list it would run down to the bottom of the deepest element and the length-1 element would sit at the top of a tall, empty, heavily-boxed column – a box drawn around cells that do not exist. The cells keep their own borders, so the block still reads as a block; it just reads as the ragged block it actually is.

`summarise = TRUE` draws every element as one cell, so the block is one row deep and rectangular whatever the elements' lengths are, and it is outlined.

The ggplot object is a shell

`gpaint_list()` returns a real `ggplot` object – `+ theme()`, `ggsave()`, `print()` and knitr chunks all work – but its panel is drawn *entirely* by a custom grid grob, held in a single `annotation_custom()` over a meaningless `0..1` coordinate system. There is no `aes()`, no geom and no scale carrying any meaning, so `ggplot_build()` sees an empty layer and `+ scale_fill_*()` has no effect on the drawing. Use `highlight_area` and `highlight_color` to fill cells.

See Also

Other painters: `paint_array()`, `paint_data_frame()`, `paint_matrix()`, `paint_size()`, `paint_vector()`

Examples

```
# Base graphics

# A list is a bag of vectors, and they need not be the same length.
paint_list(list(a = 1:4, b = "x", c = c(TRUE, FALSE, NA)))

# A data frame IS a list whose elements happen to share a length. These two
# pictures are the same picture.
paint_list(list(a = 1:3, b = 4:6))
paint_data_frame(data.frame(a = 1:3, b = 4:6))

# An unnamed element is labelled the way print() labels it.
paint_list(list(1:3, b = letters[1:2]))

# The accessor students get wrong, printed under the value it returns.
paint_list(list(a = 1:4, b = c(2.5, 3.5)), show_indices = "cell")
```

```
# A sublist is NOT drawn. It says what it is and stops there.
paint_list(list(a = 1:3, b = list(1, 2)))

# The structure of a long list, at a glance.
paint_list(list(a = 1:40, b = letters, c = matrix(1:4, 2)), summarise = TRUE)

# Highlight an element by name.
l <- list(a = 1:4, b = "x", c = c(TRUE, FALSE, NA))
paint_list(l, highlight_area = highlight_columns(l, "c"))

# Space the columns apart to stress that a list is a bag of independent
# vectors, not a grid. A spaced list draws no block outline.
paint_list(list(a = 1:3, b = c("x", "y", "z"), c = c(TRUE, FALSE, NA)), gap = 0.5)

# ggplot2 graphics ----

gpaint_list(list(a = 1:4, b = "x", c = c(TRUE, FALSE, NA)))

gpaint_list(list(a = 1:40, b = letters), summarise = TRUE)
```

`paint_matrix` *Visualize Data Inside of a Matrix*

Description

Generate a graph showing the contents of a matrix.

Usage

```
paint_matrix(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 20L,
  max_cols = 15L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_dimnames = "all",
  max_name_chars = 8L,
```

```
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)

gpaint_matrix(
  data,
  show_indices = "none",
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 20L,
  max_cols = 15L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_dimnames = "all",
  max_name_chars = 8L,
  highlight_rows = NULL,
  highlight_columns = NULL,
  highlight_locations = NULL
)
```

Arguments

`data`	An object that has the class of `matrix`.
`show_indices`	Display indices based on location. A character vector, so several kinds of index can be asked for at once. Values are: `"none"`: no indices, `"cell"`: matrix cell indices `[i, j]`, `"row"`: row indices `[i, ]` to the left of the matrix, `"column"`: column indices `[, j]` above the matrix, and `"all"`: row, column, and cell indices together. Default: `"none"`. Each value switches on its own lane, so `c("row", "column")` draws the row *and* the column indices but no cell indices, and `"all"` is the same as `c("cell", "row", "column")`. Combining `"none"` with anything else is contradictory, and the other values win: `c("none", "row")` draws row indices. An unknown value is an error, not a silent no-op.
`highlight_area`	Logical matrix the same shape as `data`, marking the cells to fill. A length-one logical is recycled. Default: `NULL`, which highlights nothing.
`highlight_color`	Color to use to fill the background of a cell.
`graph_title`	Title to appear in the upper left hand corner of the graph.
`graph_subtitle`	Subtitle to appear immediately under the graph title. `NULL` (the default) describes the data: its dimensions and its class. `NA` or `""` draws no subtitle; any other string

is drawn as given. The default reports the dimensions of the data itself, not of the drawing, so an elided matrix still reports all of its rows.

`sigfig` Significant digits drawn in black. Digits past the `sigfig`-th are drawn in grey; nothing is discarded. Must be in `1:15`.

`subtle_digits` Which digits are drawn grey. `"insignificant"` (the default) greys everything past the `sigfig`-th significant digit; `"rounded"` greys only digits the rendering actually lost; `"none"` draws everything black.

`max_chars` Strings longer than this are truncated with an ellipsis.

`max_rows`, `max_cols`
Elide the middle of the matrix when it has more rows or columns than this. The decision is made on the dimensions alone, with no device consulted, so the same object elides the same way on every device.

`show_all` Draw every cell, however small the text becomes. Warns when the text falls below the legibility floor.

`fontsize` Font size in points. `NULL` (the default) fits the text to the device.

`family` Font family. `"mono"` by default.

`palette` Colour palette for the drawing. One of `"mint"` (the default), `"slate"`, `"warm"`, or `"classic"` (the original look). Defaults to the `"paintr.palette"` option when unset. A named list of colours is also accepted.

`show_dimnames` Which of the matrix's `dimnames()` to draw. A character vector, because `dimnames()` is a *list* with one slot per axis: `"none"`, `"row"`, `"column"`, or `"all"` (the default). `c("row", "column")` is the same as `"all"`, and an axis with no names draws no lane.

An index lane the caller asks for **wins** the axis it names: with `show_indices = "row"` the row lane draws `[1, ]`, not the row names. To see both a name and an accessor, ask for `show_indices = "cell"` – the cell index is drawn *inside* the cell, so it composes with the names rather than competing for their lane.

`max_name_chars` Longest a *column* name may be drawn before it is truncated. Default: `8`. A matrix is one formatting unit, so every column of it is as wide as the widest: a long column name widens every cell in the picture. Row names sit in a gutter of their own, cost the values nothing, and are truncated at `max_chars` instead.

`highlight_rows`, `highlight_columns`, `highlight_locations`
Shorthand for `highlight_area`: instead of building a mask, name the rows, columns, or cell locations to fill and the mask is built for you with `highlight_data()`. So `highlight_rows = 1` is exactly `highlight_area = highlight_rows(data, 1)`, and the object need not be named twice. Give several at once to fill their union. Supplying `highlight_area` together with any of these is an error. Default: `NULL`.

Details

`paint_matrix()` draws on the current base graphics device. `gpaint_matrix()` returns a `ggplot` object.

Value

`paint_matrix()` invisibly returns the resolved cell table: a list with the components `cells`, `fontsize`, `floored`, `u`, `x0`, `y0`, `usr`, `pin`, `graph_title`, `graph_subtitle`, and – only when the drawing elides – `note`. `graph_title` and `graph_subtitle` are the chrome as it was actually drawn, so `graph_subtitle` is the resolved default rather than the `NULL` that was passed in. `note` holds the "# N more rows/columns" string and is present only when the drawing elides; it is absent otherwise. `gpaint_matrix()` returns a `ggplot` object.

The ggplot object is a shell

`gpaint_matrix()` returns a real `ggplot` object – `+ theme()`, `ggsave()`, `print()` and knitr chunks all work – but its panel is drawn *entirely* by a custom grid grob, held in a single `annotation_custom()` over a meaningless `0..1` coordinate system. There is no `aes()`, no geom and no scale carrying any meaning, so:

- `ggplot_build()` sees an empty layer.
- `+ scale_fill_*()`, `+ scale_x_*()` and friends have no effect on the drawing. Use `highlight_area` and `highlight_color` to fill cells.
- `+ geom_point()` would draw onto the `0..1` coordinate system, not onto the cells.

This is not a shortcut around ggplot2. The cell text is fitted to the device at *draw* time, which no geom can do, because a layer is built long before the device size is known; and every number is drawn as two spans in two colors, which `geom_text()` cannot do at all. A custom grob is the only mechanism that can do either.

See Also

Other painters: `paint_array()`, `paint_data_frame()`, `paint_list()`, `paint_size()`, `paint_vector()`

Examples

```
# Base graphics

# Visualize a 3x3
mat_3x3 = matrix(c(10, 200, -30, 40, 500, 30, 90, -55, 10), ncol = 3)
paint_matrix(mat_3x3)

# Show the cell indices
paint_matrix(mat_3x3, show_indices = "cell")

# Character data renders as the strings it contains.
paint_matrix(matrix(letters[1:6], nrow = 2))

# A dimnamed matrix labels its rows and columns with its names, like print().
mat_named = matrix(
  c(21, 6, 22.8, 4), nrow = 2, byrow = TRUE,
  dimnames = list(c("Mazda", "Datsun"), c("mpg", "cyl"))
)
paint_matrix(mat_named)
```

```
# The name above the column, the accessor under the value: both at once.
paint_matrix(mat_named, show_indices = "cell")

# An index lane wins the axis it names.
paint_matrix(mat_named, show_indices = "row")

# Or draw no names at all.
paint_matrix(mat_named, show_dimnames = "none")

# Highlight a row
mat_4x4 = matrix(seq_len(16), nrow = 4)
paint_matrix(
  mat_4x4, show_indices = "row",
  highlight_area = highlight_rows(mat_4x4, rows = 1)
)

# Highlight values above 5
mat_2x4 = matrix(round(rnorm(16, 5, 2), 2), ncol = 4)
paint_matrix(mat_2x4, highlight_area = mat_2x4 > 2)

# ggplot2 graphics ----

# Visualize a 3x3
mat_3x3 = matrix(c(10, 200, -30, 40, 500, 30, 90, -55, 10), ncol = 3)
gpaint_matrix(mat_3x3)

# View the matrix without any highlighting
gpaint_matrix(mat_3x3, highlight_area = FALSE)

# Highlight a row
mat_2x2 = matrix(c(1, 2, 3, 4), nrow = 2)
mat_2x2_mask = matrix(c(TRUE, TRUE, FALSE, FALSE), nrow = 2)
gpaint_matrix(mat_2x2, highlight_area = mat_2x2_mask)

# Highlight values above 5
mat_3x5 = matrix(round(rnorm(15, 5, 2), 2), ncol = 5)
gpaint_matrix(mat_3x5, highlight_area = mat_3x5 > 2)
```

`paint_size` *The device size a data structure needs*

Description

How big the graphics device has to be for `paint_matrix()` and friends to draw `data` at the legibility floor. Opens no device and reads no device, so it works in a fresh session with nothing plotted – which is the whole point, since the situation it exists for is "the device I have is too small".

Usage

```
paint_size(
  data,
  ...,
  min_pt = 5,
  family = "mono",
  units = c("in", "cm", "px"),
  dpi = 96
)
```

Arguments

data	A vector, matrix, or data frame.
...	Any shape-affecting painter argument – for example show_indices, summarise, show_all, layout, max_rows, max_cols, max_slices, sigfig, max_chars, show_names, show_types, show_dimnames, max_name_chars, name_align, type_align, or a list's gap. These are passed to the cell builder, so the estimate matches what the painter would draw.
min_pt	The legibility floor, in points. The returned size is the one that puts the fitted text exactly here.
family	Font family. Only its metrics matter, and measure_mono() is used for all of them, so this argument currently changes nothing; it is present so the signature matches the painters'.
units	"in", "cm", or "px".
dpi	Pixels per inch, used only when units = "px".

Details

The answer is a lower bound: at exactly this size the text lands on min_pt, so round up in practice.

Value

A named numeric vector, c(width = , height =). Inches and centimetres are rounded up to a tenth; pixels are rounded up to a whole pixel.

See Also

Other painters: paint_array(), paint_data_frame(), paint_list(), paint_matrix(), paint_vector()

Examples

```
# How large a device does a 20x20 matrix need, drawn in full?
paint_size(matrix(1:400, nrow = 20), show_all = TRUE)

# The same question in pixels, for a png() at 96 dpi.
paint_size(iris, show_all = TRUE, units = "px")

# The answer is in inches by default, so it can be pasted straight into a
```

```
# device call or a knitr chunk header (fig.width, fig.height).
s <- paint_size(iris, show_all = TRUE)
s
# png("iris.png", width = s[["width"]], height = s[["height"]],
#     units = "in", res = 96)
```

---

paint_vector *Visualize Data Inside of a Vector*

Description

Generate a graph showing the contents of a vector.

Usage

```
paint_vector(
  data,
  layout = c("vertical", "horizontal"),
  show_indices = c("none", "inside", "outside"),
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 20L,
  max_cols = 15L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_names = TRUE,
  max_name_chars = 8L,
  highlight_locations = NULL
)

gpaint_vector(
  data,
  layout = c("vertical", "horizontal"),
  show_indices = c("none", "inside", "outside"),
  highlight_area = NULL,
  highlight_color = "lemonchiffon",
  graph_title = paste0("Data Object: ", deparse(substitute(data))),
  graph_subtitle = NULL,
  sigfig = 3L,
```

```
  subtle_digits = c("insignificant", "rounded", "none"),
  max_chars = 12L,
  max_rows = 20L,
  max_cols = 15L,
  show_all = FALSE,
  fontsize = NULL,
  family = "mono",
  palette = NULL,
  show_names = TRUE,
  max_name_chars = 8L,
  highlight_locations = NULL
)
```

Arguments

`data`	An object that has the class of `vector`. Factors, Dates and other classed atomic vectors are accepted too.
`layout`	Orientation of the vector. Default: `"vertical"`.
`show_indices`	Display data indices either `"inside"` the cell, `"outside"` it, or `"none"`. Default: `"none"`. Exactly one value: a vector has a single index `[i]`, so its placements are mutually exclusive. (A matrix or data frame has independent row, column and cell lanes, and `paint_matrix()` does take several at once.)
`highlight_area`	Logical vector the same length as `data`, marking the cells to fill. A length-one logical is recycled. Default: `NULL`, which highlights nothing.
`highlight_color`	Color to use to fill the background of a cell.
`graph_title`	Title to appear in the upper left hand corner of the graph.
`graph_subtitle`	Subtitle to appear immediately under the graph title. `NULL` (the default) describes the data: its length and its class. `NA` or `""` draws no subtitle; any other string is drawn as given. The default reports the length of the data itself, not of the drawing, so an elided vector still reports all of its elements.
`sigfig`	Significant digits drawn in black. Digits past the `sigfig`-th are drawn in grey; nothing is discarded. Must be in `1:15`.
`subtle_digits`	Which digits are drawn grey. `"insignificant"` (the default) greys everything past the `sigfig`-th significant digit; `"rounded"` greys only digits the rendering actually lost; `"none"` draws everything black.
`max_chars`	Strings longer than this are truncated with an ellipsis.
`max_rows, max_cols`	Elide the middle of the vector when it is longer than this. A vertical vector is elided by `max_rows`, a horizontal one by `max_cols`.
`show_all`	Draw every cell, however small the text becomes. Warns when the text falls below the legibility floor.
`fontsize`	Font size in points. `NULL` (the default) fits the text to the device.
`family`	Font family. `"mono"` by default.

palette — Colour palette for the drawing. One of "mint" (the default), "slate", "warm", or "classic" (the original look). Defaults to the "paintr.palette" option when unset. A named list of colours is also accepted.

show_names — Draw the vector's names() in the label lane its layout gives it: a gutter to the left when the vector is vertical, a lane above it when it is horizontal. Default: TRUE. A logical scalar, because names() returns one vector. An unnamed vector draws no lane.

show_indices = "outside" **wins** that lane, because the caller asked for it by name. show_indices = "inside" draws the index *within* the cell, so it composes with the names instead of competing with them.

max_name_chars — Longest a name may be drawn before it is truncated. Default: 8. It bites only a "horizontal" vector, whose names sit over the cells and widen them; a vertical vector's names are in a gutter of their own and are truncated at max_chars instead.

highlight_locations — Shorthand for highlight_area: instead of building a mask, give the element positions to fill and the mask is built for you with highlight_locations(). So highlight_locations = c(2, 4) is exactly highlight_area = highlight_locations(data, c(2, 4)). A vector has a single axis addressed by position, so this is the only shorthand it takes – there are no rows or columns to name. Supplying highlight_area together with it is an error. Default: NULL.

Details

paint_vector() draws on the current base graphics device. gpaint_vector() returns a ggplot object.

Value

paint_vector() invisibly returns the resolved cell table. See paint_matrix() for its components. gpaint_vector() returns a ggplot object.

The ggplot object is a shell

gpaint_vector() returns a real ggplot object – + theme(), ggsave(), print() and knitr chunks all work – but its panel is drawn *entirely* by a custom grid grob, held in a single annotation_custom() over a meaningless 0..1 coordinate system. There is no aes(), no geom and no scale carrying any meaning, so:

- ggplot_build() sees an empty layer.
- + scale_fill_*(), + scale_x_*() and friends have no effect on the drawing. Use highlight_area and highlight_color to fill cells.
- + geom_point() would draw onto the 0..1 coordinate system, not onto the cells.

This is not a shortcut around ggplot2. The cell text is fitted to the device at *draw* time, which no geom can do, because a layer is built long before the device size is known; and every number is drawn as two spans in two colors, which geom_text() cannot do at all. A custom grob is the only mechanism that can do either.

See Also

Other painters: paint_array(), paint_data_frame(), paint_list(), paint_matrix(), paint_size()

Examples

```
# Base graphics

# Visualize a vector with 5 elements
vec_5 <- round(rnorm(5, 0, 4), 2)
paint_vector(vec_5)

# Character vectors render as the strings they contain.
paint_vector(letters[1:5], layout = "horizontal")

# A named vector draws its names, like print() does.
paint_vector(c(alpha = 1, beta = 2.5, gamma = -30))

# The name beside the cell, the accessor inside it: both at once.
paint_vector(c(alpha = 1, beta = 2.5), show_indices = "inside")

# Visualize a 6 element vector with indices underneath the data
vec_6 <- c(-3, 5, NA, Inf, 2, 1)
paint_vector(vec_6, layout = "horizontal", show_indices = "inside")

# Highlight the 2nd, 4th, and 6th cell with indices shown outside
paint_vector(
  vec_6, show_indices = "outside",
  highlight_area = highlight_locations(vec_6, c(2, 4, 6))
)

# ggplot2 graphics ----

gpaint_vector(c(-3, 5, NA, Inf, 2, 1))

gpaint_vector(letters[1:5], layout = "horizontal", show_indices = "outside")
```

Index

∗ **highlighters**
highlight_data, 2
∗ **painters**
paint_array, 5
paint_data_frame, 10
paint_list, 15
paint_matrix, 20
paint_size, 24
paint_vector, 26

gpaint_array *(*paint_array*)*, 5
gpaint_data_frame *(*paint_data_frame*)*, 10
gpaint_df *(*paint_data_frame*)*, 10
gpaint_list *(*paint_list*)*, 15
gpaint_matrix *(*paint_matrix*)*, 20
gpaint_vector *(*paint_vector*)*, 26

highlight_columns *(*highlight_data*)*, 2
highlight_columns(), *17*
highlight_data, 2
highlight_data(), *6*, *8*, *14*, *18*, *22*
highlight_locations *(*highlight_data*)*, 2
highlight_locations(), *28*
highlight_rows *(*highlight_data*)*, 2
highlight_rows(), *17*

paint_array, 5
paint_array(), *4*, *14*, *19*, *23*, *25*, *29*
paint_data_frame, 10
paint_data_frame(), *4*, *9*, *19*, *23*, *25*, *29*
paint_df *(*paint_data_frame*)*, 10
paint_list, 15
paint_list(), *4*, *9*, *14*, *23*, *25*, *29*
paint_matrix, 20
paint_matrix(), *4*, *7–9*, *14*, *19*, *24, 25*, *27–29*
paint_size, 24
paint_size(), *9*, *14*, *19*, *23*, *29*
paint_vector, 26
paint_vector(), *9*, *14*, *19*, *23*, *25*