Package 'pillar'

May 16, 2021

Title Coloured Formatting for Columns

Version 1.6.1

Description Provides 'pillar' and 'colonnade' generics designed for formatting columns of data using the full range of colours provided by modern terminals.

License MIT + file LICENSE

URL https://pillar.r-lib.org/, https://github.com/r-lib/pillar

BugReports https://github.com/r-lib/pillar/issues

Imports cli, crayon (>= 1.3.4), ellipsis (>= 0.3.2), fansi, lifecycle, rlang (>= 0.3.0), utf8 (>= 1.1.0), utils, vctrs (>= 0.3.8)

Suggests bit64, debugme, DiagrammeR, dplyr, formattable, ggplot2, knitr, lubridate, nycflights13, palmerpenguins, rmarkdown, scales, stringi, survival, testthat (>= 3.0.2), tibble, units, withr

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.1.9001

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first format_multi_fuzz, format_multi_fuzz_2, format_multi, ctl_colonnade, ctl_colonnade_1, ctl_colonnade_2

NeedsCompilation no

Author Kirill Müller [aut, cre], Hadley Wickham [aut], RStudio [cph]

Maintainer Kirill Müller <krlmlr+r@mailbox.org>

Repository CRAN

Date/Publication 2021-05-16 14:10:09 UTC

R topics documented:

`pillar-package` *pillar: Coloured Formatting for Columns*

Description

[Stable]

Formats tabular data in columns or rows using the full range of colours provided by modern terminals. Provides various generics for making every aspect of the display customizable.

Details

See `pillar()` for formatting a single column, and `print.tbl()` for formatting data-frame-like objects.

Package options

- `pillar.bold`: Use bold font, e.g. for column headers? This currently defaults to `FALSE`, because many terminal fonts have poor support for bold fonts.
- `pillar.subtle`: Use subtle style, e.g. for row numbers and data types? Default: `TRUE`.
- `pillar.subtle_num`: Use subtle style for insignificant digits? Default: `FALSE`, is also affected by the `pillar.subtle` option.
- `pillar.neg`: Highlight negative numbers? Default: `TRUE`.
- `pillar.sigfig`: The number of significant digits that will be printed and highlighted, default: `3`. Set the `pillar.subtle` option to `FALSE` to turn off highlighting of significant digits.
- `pillar.min_title_chars`: The minimum number of characters for the column title, default: `15`. Column titles may be truncated up to that width to save horizontal space. Set to `Inf` to turn off truncation of column titles.
- `pillar.min_chars`: The minimum number of characters wide to display character columns, default: `0`. Character columns may be truncated up to that width to save horizontal space. Set to `Inf` to turn off truncation of character columns.
- `pillar.max_dec_width`: The maximum allowed width for decimal notation, default 13.

Author(s)

Maintainer: Kirill Müller <krlmlr+r@mailbox.org>

Authors:

- Hadley Wickham

Other contributors:

- RStudio [copyright holder]

See Also

Useful links:

- https://pillar.r-lib.org/
- https://github.com/r-lib/pillar
- Report bugs at https://github.com/r-lib/pillar/issues

Examples

```
pillar(1:3)
pillar(c(1, 2, 3))
pillar(factor(letters[1:3]), title = "letters")
colonnade(iris[1:3, ])
```

align *Alignment helper*

Description

Facilitates easy alignment of strings within a character vector. Designed to help implementers of formatters for custom data types.

Usage

```
align(x, width = NULL, align = c("left", "right"), space = " ")
```

Arguments

x	A character vector
width	The width that each string is padded to. If NULL, the maximum display width of the character vector is used (see get_max_extent()).
align	How should strings be aligned? If align = left then padding appears on the right, and vice versa.
space	What character should be used for the padding?

Examples

```
align(c("abc", "de"), align = "left")
align(c("abc", "de"), align = "right")
```

char *Format a character vector in a tibble*

Description

[Experimental]

Constructs a character vector that can be formatted with predefined minimum width or without width restrictions, and where the abbreviation style can be configured.

The formatting is applied when the vector is printed or formatted, and also in a tibble column.

set_char_opts() adds formatting options to an arbitrary character vector, useful for composing with other types.

Usage

```
char(
  x,
  ...,
  min_chars = NULL,
  shorten = c("back", "front", "mid", "abbreviate")
)

set_char_opts(
  x,
  ...,
  min_chars = NULL,
  shorten = c("back", "front", "mid", "abbreviate")
)
```

Arguments

x	A character vector.
...	These dots are for future extensions and must be empty.
min_chars	The minimum width to allocate to this column, defaults to 15. The "pillar.min_chars" option is not consulted.
shorten	How to abbreviate the data if necessary: • "back" (default): add an ellipsis at the end • "front": add an ellipsis at the front • "mid": add an ellipsis in the middle • "abbreviate": use abbreviate()

See Also

Other vector classes: num()

Examples

```
# Display as a vector:
char(letters[1:3])

# Space constraints:
rand_strings <- stringi::stri_rand_strings(10, seq(40, 22, by = -2))

# Plain character vectors get truncated if space is limited:
data_with_id <- function(id) {
  tibble::tibble(
    id,
    some_number_1 = 1, some_number_2 = 2, some_number_3 = 3,
    some_number_4 = 4, some_number_5 = 5, some_number_6 = 6,
    some_number_7 = 7, some_number_8 = 8, some_number_9 = 9
  )
}
```

```
data_with_id(rand_strings)

# Use char() to avoid or control truncation
data_with_id(char(rand_strings, min_chars = 24))
data_with_id(char(rand_strings, min_chars = Inf))
data_with_id(char(rand_strings, min_chars = 24, shorten = "mid"))

# Lorem Ipsum, one sentence per row.
lipsum <- unlist(strsplit(stringi::stri_rand_lipsum(1), "(?<=[.]) +", perl = TRUE))
tibble::tibble(
  back = char(lipsum, shorten = "back"),
  front = char(lipsum, shorten = "front"),
  mid   = char(lipsum, shorten = "mid")
)
tibble::tibble(abbr = char(lipsum, shorten = "abbreviate"))
```

`ctl_new_pillar` *Customize your tibble subclass*

Description

Gain full control over the appearance of the pillars of your tibble subclass in its body. These methods are intended for implementers of subclasses of the `"tbl"` class. Users will rarely need them.

Usage

```
ctl_new_pillar(controller, x, width, ..., title = NULL)

ctl_new_compound_pillar(controller, x, width, ..., title = NULL)
```

Arguments

`controller`	The object of class `"tbl"` currently printed.
`x`	A vector, can also be a data frame, array or matrix in `ctl_new_compound_pillar()`
`width`	The available width, can be a vector for multiple tiers
`...`	These dots are for future extensions and must be empty.
`title`	The title, derived from the name of the column in the data

Details

`ctl_new_pillar()` is called to construct pillars for regular (one-dimensional) vectors. The default implementation returns an object constructed with `pillar()`. Extend this method to tweak pillar components returned from the default implementation. Override this method to completely change the appearance of the pillars.

`ctl_new_compound_pillar()` is called for compound pillars: columns that are data frames, matrices or arrays. The default implementation returns a compound pillar with suitable formatting for the titles and types of the sub-pillar. Users will only rarely need to override this method if ever.

All components must be of the same height. This restriction may be levied in the future.

Implementations should return NULL if none of the data fits the available width.

Examples

```
# Create pillar objects
ctl_new_pillar(
  palmerpenguins::penguins,
  palmerpenguins::penguins$species[1:3], width = 60
)
ctl_new_pillar(
  palmerpenguins::penguins,
  palmerpenguins::penguins$bill_length_mm[1:3],
  width = 60
)

# Packed data frame
ctl_new_compound_pillar(
  tibble::tibble(),
  palmerpenguins::penguins,
  width = 60
)

# Packed matrix
ctl_new_compound_pillar(tibble::tibble(), matrix(1:6, ncol = 2), width = 60)

# Packed array
ctl_new_compound_pillar(tibble::tibble(), Titanic, width = 60)


# Customize output
lines <- function(char = "-") {
  stopifnot(nchar(char) == 1)
  structure(char, class = "lines")
}

format.lines <- function(x, width, ...) {
  paste(rep(x, width), collapse = "")
}

ctl_new_pillar.line_tbl <- function(controller, x, width, ..., title = NULL) {
  out <- NextMethod()
  new_pillar(list(
    title = out$title,
    type = out$type,
    lines = new_pillar_component(list(lines("=")), width = 1),
    data = out$data
  ))
}

vctrs::new_data_frame(
```

```
  list(a = 1:3, b = letters[1:3]),
  class = c("line_tbl", "tbl")
)
```

`dim_desc` *Format dimensions*

Description

Multi-dimensional objects are formatted as a x b x ..., for vectors the length is returned.

Usage

```
dim_desc(x)
```

Arguments

`x`	The object to format the dimensions for

Examples

```
dim_desc(1:10)
dim_desc(Titanic)
```

`format_glimpse` *Format a vector for horizontal printing*

Description

[Experimental]

This generic provides the logic for printing vectors in `glimpse()`.

The output strives to be as unambiguous as possible, without compromising on readability. In a list, to distinguish between vectors and nested lists, the latter are surrounded by [] brackets. Empty lists are shown as []. Vectors inside lists, of length not equal to one, are surrounded by <> angle brackets. Empty vectors are shown as <>.

Usage

```
format_glimpse(x, ...)
```

Arguments

`x`	A vector.
`...`	Arguments passed to methods.

Value

A character vector of the same length as x.

Examples

```
format_glimpse(1:3)

# Lists use [], vectors inside lists use <>
format_glimpse(list(1:3))
format_glimpse(list(1, 2:3))
format_glimpse(list(list(1), list(2:3)))
format_glimpse(list(as.list(1), as.list(2:3)))
format_glimpse(list(character()))
format_glimpse(list(NULL))

# Character strings are always quoted
writeLines(format_glimpse(letters[1:3]))
writeLines(format_glimpse(c("A", "B, C")))

# Factors are quoted only when needed
writeLines(format_glimpse(factor(letters[1:3])))
writeLines(format_glimpse(factor(c("A", "B, C"))))
```

---

format_tbl *Formatting of tbl objects*

Description

These functions and methods are responsible for printing objects of the "tbl" class, which includes tibbles and dbplyr lazy tables. See tibble::formatting for user level documentation, and vignette("customization") for details.

While it is possible to implement a custom format() or print() method for your tibble-like objects, it should never be necessary if your class inherits from "tbl". In this case, the default methods offer many customization options at every level of detail. This means you only need to override or extend implementations for the parts that need change.

The output uses color and highlighting according to the "cli.num_colors" option. Set it to 1 to suppress colored and highlighted output.

Usage

```
## S3 method for class 'tbl'
print(x, width = NULL, ..., n = NULL, n_extra = NULL)

## S3 method for class 'tbl'
format(x, width = NULL, ..., n = NULL, n_extra = NULL)
```

Arguments

x	Object to format or print.
width	Width of text output to generate. This defaults to NULL, which means use getOption("tibble.width") or (if also NULL) getOption("width"); the latter displays only the columns that fit on one screen. You can also set options(tibble.width = Inf) to override this default and always print all columns.
...	Passed on to tbl_format_setup().
n	Number of rows to show. If NULL, the default, will print all rows if less than option tibble.print_max. Otherwise, will print tibble.print_min rows.
n_extra	Number of extra columns to print abbreviated information for, if the width is too small for the entire tibble. If NULL, the default, the value of the tibble.max_extra_cols option is used.

See Also

- tbl_format_setup() for preparing an object for formatting

Examples

```
print(vctrs::new_data_frame(list(a = 1), class = "tbl"))
```

---

format_type_sum *Format a type summary*

Description

Called on values returned from type_sum() for defining the description in the capital.

Usage

```
format_type_sum(x, width, ...)

## Default S3 method:
format_type_sum(x, width, ...)

## S3 method for class 'AsIs'
format_type_sum(x, width, ...)
```

Arguments

x	A return value from type_sum()
width	The desired total width. If the returned string still is wider, it will be trimmed. Can be NULL.
...	Arguments passed to methods.

Details

Two methods are implemented by default for this generic: the default method, and the method for the "AsIs" class. Return I("type") from your type_sum() implementation to format the type without angle brackets. For even more control over the formatting, implement your own method.

Examples

```
# Default method: show the type with angle brackets
format_type_sum(1, NULL)
pillar(1)

# AsIs method: show the type without angle brackets
type_sum.accel <- function(x) {
  I("kg m/s^2")
}
accel <- structure(9.81, class = "accel")
pillar(accel)
```

---

get_extent *Calculate display width*

Description

get_extent() calculates the display width for each string in a character vector.

get_max_extent() calculates the maximum display width of all strings in a character vector, zero for empty vectors.

Usage

```
get_extent(x)

get_max_extent(x)
```

Arguments

x	A character vector.

Examples

```
get_extent(c("abc", "de"))
get_extent("\u904b\u6c23")
get_max_extent(c("abc", "de"))
```

`glimpse` *Get a glimpse of your data*

Description

`glimpse()` is like a transposed version of `print()`: columns run down the page, and data runs across. This makes it possible to see every column in a data frame. It's a little like `str()` applied to a data frame but it tries to show you as much data as possible. (And it always shows the underlying data, even when applied to a remote data source.)

See `format_glimpse()` for details on the formatting.

Usage

```
glimpse(x, width = NULL, ...)
```

Arguments

`x`	An object to glimpse at.
`width`	Width of output: defaults to the setting of the option `tibble.width` (if finite) or the width of the console.
`...`	Unused, for extensibility.

Value

x original x is (invisibly) returned, allowing `glimpse()` to be used within a data pipe line.

S3 methods

`glimpse` is an S3 generic with a customised method for `tbls` and `data.frames`, and a default method that calls `str()`.

Examples

```
glimpse(mtcars)


glimpse(nycflights13::flights)
```

new_ornament *Helper to define the contents of a pillar*

Description

This function is useful if your data renders differently depending on the available width. In this case, implement the `pillar_shaft()` method for your class to return a subclass of "pillar_shaft" and have the `format()` method for this subclass call `new_ornament()`. See the implementation of `pillar_shaft.numeric()` and `format.pillar_shaft_decimal()` for an example.

Usage

```
new_ornament(x, width = NULL, align = NULL)
```

Arguments

`x`	A character vector with formatting, see crayon
`width`	An optional width of the resulting pillar, computed from `x` if missing
`align`	Alignment, one of `"left"` or `"right"`

Examples

```
new_ornament(c("abc", "de"), align = "right")
```

new_pillar *Construct a custom pillar object*

Description

[Experimental]

`new_pillar()` is the low-level constructor for pillar objects. It supports arbitrary components. See `pillar()` for the high-level constructor with default components.

Usage

```
new_pillar(components, ..., width = NULL, class = NULL)
```

Arguments

`components`	A named list of components constructed with `pillar_component()`.
`...`	These dots are for future extensions and must be empty.
`width`	Default width, optional.
`class`	Name of subclass.

Details

Arbitrary components are supported. If your tibble subclass needs more or different components in its pillars, override or extend `ctl_new_pillar()` and perhaps `ctl_new_compound_pillar()`.

Examples

```
lines <- function(char = "-") {
  stopifnot(nchar(char) == 1)
  structure(char, class = "lines")
}

format.lines <- function(x, width, ...) {
  paste(rep(x, width), collapse = "")
}

new_pillar(list(
  title = pillar_component(new_ornament(c("abc", "de"), align = "right")),
  lines = new_pillar_component(list(lines("=")), width = 1)
))
```

---

new_pillar_component *Components of a pillar*

Description

`new_pillar_component()` constructs an object of class `"pillar_component"`.

`pillar_component()` is a convenience helper that wraps the input in a list and extracts width and minimum width.

Usage

```
new_pillar_component(x, ..., width, min_width = NULL)

pillar_component(x)
```

Arguments

`x`	A bare list (for `new_pillar_component()`), or an object with attributes `"width"` and `"min_width"` attributes (for `pillar_component()`).
`...`	These dots are for future extensions and must be empty.
`width, min_width`	Width and minimum width for the new component. If `min_width` is `NULL`, it is assumed to match `width`.

Details

Objects of class "pillar" are internally a named lists of their components. The default components are title (may be missing), type, and data. Each component is a "pillar_component".

This class captures contents that can be fitted in a rectangle. Each component consists of one or multiple cells that are aligned horizontally (with one space in between) when printed. Each cell has a maximum (i.e., desired) width and may have a minimum width if the contents are compressible. The component object stores the width of the cells as an attribute.

Examples

```
new_pillar_component(list(letters[1:3]), width = 1)
pillar_component(new_pillar_title("letters"))
pillar_component(new_pillar_type(letters))
pillar_component(pillar_shaft(letters[1:3]))
```

---

new_pillar_shaft *Constructor for column data*

Description

The new_pillar_shaft() constructor creates objects of the "pillar_shaft" class. This is a virtual or abstract class, you must specify the class argument. By convention, this should be a string that starts with "pillar_shaft_". See vignette("extending",package = "tibble") for usage examples.

This method accepts a vector of arbitrary length and is expected to return an S3 object with the following properties:

- It has an attribute "width"
- It can have an attribute "min_width", if missing, "width" is used
- It must implement a method format(x,width,...) that can be called with any value between min_width and width
- This method must return an object that inherits from character and has attributes "align" (with supported values "left", "right", and "center") and "width"

The function new_pillar_shaft() returns such an object, and also correctly formats NA values. In many cases, the implementation of pillar_shaft.your_class_name() will format the data as a character vector (using color for emphasis) and simply call new_pillar_shaft(). See pillar:::pillar_shaft.numeric for a code that allows changing the display depending on the available width.

new_pillar_shaft_simple() provides an implementation of the pillar_shaft class suitable for output that has a fixed formatting, which will be truncated with a continuation character (ellipsis or ~) if it doesn't fit the available width. By default, the required width is computed from the natural width of the formatted argument.

Usage

```
new_pillar_shaft(
  x,
  ...,
  width = NULL,
  min_width = width,
  class = NULL,
  subclass = NULL
)

new_pillar_shaft_simple(
  formatted,
  ...,
  width = NULL,
  align = "left",
  min_width = NULL,
  na = NULL,
  na_indent = 0L,
  shorten = c("back", "front", "mid", "abbreviate")
)
```

Arguments

x	An object
...	Additional attributes.
width	The maximum column width.
min_width	The minimum allowed column width, `width` if omitted.
class	The name of the subclass.
subclass	Deprecated, pass the `class` argument instead.
formatted	An object coercible to character.
align	Alignment of the column.
na	String to use as `NA` value, defaults to `"NA"` styled with `style_na()` with fallback if color is not available.
na_indent	Indentation of `NA` values.
shorten	How to abbreviate the data if necessary: • `"back"` (default): add an ellipsis at the end • `"front"`: add an ellipsis at the front • `"mid"`: add an ellipsis in the middle • `"abbreviate"`: use `abbreviate()`

Details

The `formatted` argument may also contain ANSI escapes to change color or other attributes of the text, see crayon.

new_pillar_title *Prepare a column title for formatting*

Description

Call `format()` on the result to render column titles.

Usage

```
new_pillar_title(x, ...)
```

Arguments

x	A character vector of column titles.
...	These dots are for future extensions and must be empty.

Examples

```
format(new_pillar_title(names(iris)))
```

new_pillar_type *Prepare a column type for formatting*

Description

Calls `type_sum()` to format the type. Call `format()` on the result to render column types.

Usage

```
new_pillar_type(x, ...)
```

Arguments

x	A vector for which the type is to be retrieved.
...	These dots are for future extensions and must be empty.

Examples

```
format(new_pillar_type(iris$Species))
```

num *Format a numeric vector in a tibble*

Description

[Experimental]

Constructs a numeric vector that can be formatted with predefined significant digits, or with a maximum or fixed number of digits after the decimal point. Scaling is supported, as well as forcing a decimal, scientific or engineering notation. If a label is given, it is shown in the header of a column.

The formatting is applied when the vector is printed or formatted, and also in a tibble column. The formatting annotation and the class survives most arithmetic transformations, the most notable exceptions are `var()` and `sd()`.

`set_num_opts()` adds formatting options to an arbitrary numeric vector, useful for composing with other types.

Usage

```
num(
  x,
  ...,
  sigfig = NULL,
  digits = NULL,
  label = NULL,
  scale = NULL,
  notation = c("fit", "dec", "sci", "eng", "si"),
  fixed_exponent = NULL
)

set_num_opts(
  x,
  ...,
  sigfig = NULL,
  digits = NULL,
  label = NULL,
  scale = NULL,
  notation = c("fit", "dec", "sci", "eng", "si"),
  fixed_exponent = NULL
)
```

Arguments

`x`	A numeric vector.
`...`	These dots are for future extensions and must be empty.
`sigfig`	Define the number of significant digits to show. Must be one or greater. The `"pillar.sigfig"` option is not consulted. Can't be combined with `digits`.

digits — Number of digits after the decimal points to show. Positive numbers specify the exact number of digits to show. Negative numbers specify (after negation) the maximum number of digits to show. With `digits = 2`, the numbers 1.2 and 1.234 are printed as 1.20 and 1.23, with `digits = -2` as 1.2 and 1.23, respectively. Can't be combined with `sigfig`.

label — A label to show instead of the type description.

scale — Multiplier to apply to the data before showing. Useful for displaying e.g. percentages. Must be combined with `label`.

notation — One of `"fit"`, `"dec"`, `"sci"`, `"eng"`, or `"si"`.

- `"fit"`: Use decimal notation if it fits and if it consumes 13 digits or less, otherwise use scientific notation. (The default for numeric pillars.)
- `"dec"`: Use decimal notation, regardless of width.
- `"sci"`: Use scientific notation.
- `"eng"`: Use engineering notation, i.e. scientific notation using exponents that are a multiple of three.
- `"si"`: Use SI notation, prefixes between `1e-24` and `1e24` are supported.

fixed_exponent — Use the same fixed_exponent for all numbers in scientific, engineering or SI notation. `-Inf` uses the smallest, `+Inf` the largest fixed_exponent present in the data. The default is to use varying fixed_exponents.

See Also

Other vector classes: `char()`

Examples

```
# Display as a vector
num(9:11 * 100 + 0.5)


# Significant figures
tibble::tibble(
  x3 = num(9:11 * 100 + 0.5, sigfig = 3),
  x4 = num(9:11 * 100 + 0.5, sigfig = 4),
  x5 = num(9:11 * 100 + 0.5, sigfig = 5),
)

# Maximum digits after the decimal points
tibble::tibble(
  x0 = num(9:11 * 100 + 0.5, digits =  0),
  x1 = num(9:11 * 100 + 0.5, digits = -1),
  x2 = num(9:11 * 100 + 0.5, digits = -2),
)

# Use fixed digits and a currency label
tibble::tibble(
  usd = num(9:11 * 100 + 0.5, digits = 2, label = "USD"),
  gbp = num(9:11 * 100 + 0.5, digits = 2, label = "£"),
  chf = num(9:11 * 100 + 0.5, digits = 2, label = "SFr")
```

```
)

# Scale
tibble::tibble(
  small  = num(9:11 / 1000 + 0.00005, label = "%", scale = 100),
  medium = num(9:11 /  100 + 0.0005 , label = "%", scale = 100),
  large  = num(9:11 /   10 + 0.005  , label = "%", scale = 100)
)

# Notation
tibble::tibble(
  sci = num(10^(-13:6), notation = "sci"),
  eng = num(10^(-13:6), notation = "eng"),
  si  = num(10^(-13:6), notation = "si"),
  dec = num(10^(-13:6), notation = "dec")
)

# Fixed exponent
tibble::tibble(
  scimin = num(10^(-7:6) * 123, notation = "sci", fixed_exponent = -Inf),
  engmin = num(10^(-7:6) * 123, notation = "eng", fixed_exponent = -Inf),
  simin  = num(10^(-7:6) * 123, notation = "si",  fixed_exponent = -Inf)
)

tibble::tibble(
  scismall = num(10^(-7:6) * 123, notation = "sci", fixed_exponent = -3),
  scilarge = num(10^(-7:6) * 123, notation = "sci", fixed_exponent = 3),
  scimax   = num(10^(-7:6) * 123, notation = "sci", fixed_exponent = Inf)
)
```

`pillar` *Object for formatting a vector suitable for tabular display*

Description

`pillar()` creates an object that formats a vector. The output uses one row for a title (if given), one row for the type, and `vec_size(x)` rows for the data.

Usage

```
pillar(x, title = NULL, width = NULL, ...)
```

Arguments

`x`	A vector to format.
`title`	An optional title for the column. The title will be used "as is", no quoting will be applied.
`width`	Default width, optional.
`...`	Passed on to `pillar_shaft()`.

Details

A pillar consists of arbitrary components. The `pillar()` constructor uses `title`, `type`, and `data`.

- `title` via `new_pillar_title()`
- `type` via `new_pillar_type()`, which calls `type_sum()` internally
- `data` via `pillar_shaft()`

All components are formatted via `format()` when displaying the pillar. A `width` argument is passed to each `format()` call.

As of pillar 1.5.0, `pillar()` returns `NULL` if the width is insufficient to display the data.

Examples

```
x <- 123456789 * (10^c(-1, -3, -5, NA, -8, -10))
pillar(x)
pillar(-x)
pillar(runif(10))
pillar(rcauchy(20))

# Special values are highlighted
pillar(c(runif(5), NA, NaN, Inf, -Inf))

# Very wide ranges will be displayed in scientific format
pillar(c(1e10, 1e-10), width = 20)
pillar(c(1e10, 1e-10))

x <- c(FALSE, NA, FALSE, FALSE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE)
pillar(x)

x <- c("This is string is rather long", NA, "?", "Short")
pillar(x)
pillar(x, width = 30)
pillar(x, width = 5)

date <- as.Date("2017-05-15")
pillar(date + c(1, NA, 3:5))
pillar(as.POSIXct(date) + c(30, NA, 600, 3600, 86400))
```

---

pillar_shaft *Column data*

Description

Internal class for formatting the data for a column. `pillar_shaft()` is a coercion method that must be implemented for your data type to display it in a tibble.

This class comes with a default method for `print()` that calls `format()`. If `print()` is called without `width` argument, the natural width will be used when calling `format()`. Usually there's no need to implement this method for your subclass.

Your subclass must implement `format()`, the default implementation just raises an error. Your `format()` method can assume a valid value for the `width` argument.

Usage

```
pillar_shaft(x, ...)

## S3 method for class 'pillar_shaft'
print(x, width = NULL, ...)

## S3 method for class 'pillar_shaft'
format(x, width, ...)

## S3 method for class 'logical'
pillar_shaft(x, ...)

## S3 method for class 'numeric'
pillar_shaft(x, ..., sigfig = NULL)

## S3 method for class 'Date'
pillar_shaft(x, ...)

## S3 method for class 'POSIXt'
pillar_shaft(x, ...)

## S3 method for class 'character'
pillar_shaft(x, ..., min_width = NULL)

## S3 method for class 'pillar_vertical'
pillar_shaft(x, ..., min_width = NULL, na_indent = 0L, shorten = NULL)

## S3 method for class 'list'
pillar_shaft(x, ...)

## S3 method for class 'factor'
pillar_shaft(x, ...)

## S3 method for class 'AsIs'
pillar_shaft(x, ...)

## Default S3 method:
pillar_shaft(x, ...)
```

Arguments

x	A vector to format
...	Arguments passed to methods.
width	Width for printing and formatting.
sigfig	Deprecated, use num() or set_num_opts() on the data instead.
min_width	Deprecated, use char() or set_char_opts() on the data instead.
na_indent	Indentation of NA values.

shorten — How to abbreviate the data if necessary:

- "back" (default): add an ellipsis at the end
- "front": add an ellipsis at the front
- "mid": add an ellipsis in the middle
- "abbreviate": use abbreviate()

Details

The default method will currently format via format(), but you should not rely on this behavior.

Examples

```
pillar_shaft(1:3)
pillar_shaft(1.5:3.5)
pillar_shaft(NA)
pillar_shaft(c(1:3, NA))
```

---

style_num *Styling helpers*

Description

Functions that allow implementers of formatters for custom data types to maintain a consistent style with the default data types.

Usage

```
style_num(x, negative, significant = rep_along(x, TRUE))

style_subtle(x)

style_subtle_num(x, negative)

style_bold(x)

style_na(x)

style_neg(x)
```

Arguments

x — The character vector to style.

negative, significant — Logical vector the same length as x that indicate if the values are negative and significant, respectively

Details

`style_subtle()` is affected by the `pillar.subtle` option.

`style_subtle_num()` is affected by the `pillar.subtle_num` option, which is `FALSE` by default.

`style_bold()` is affected by the `pillar.bold` option.

`style_neg()` is affected by the `pillar.neg` option.

See Also

pillar-package for a list of options

Examples

```
style_num(
  c("123", "456"),
  negative = c(TRUE, FALSE)
)
style_num(
  c("123", "456"),
  negative = c(TRUE, FALSE),
  significant = c(FALSE, FALSE)
)
style_subtle("text")
style_subtle_num(0.01 * 1:3, c(TRUE, FALSE, TRUE))
style_bold("Petal.Width")
style_na("NA")
style_neg("123")
```

---

tbl_format_body *Format the body of a tibble*

Description

[Experimental]

For easier customization, the formatting of a tibble is split into three components: header, body, and footer. The `tbl_format_body()` method is responsible for formatting the body of a tibble.

Override this method if you need to change the appearance of all parts of the body. If you only need to change the appearance of a single data type, override `vctrs::vec_ptype_abbr()` and `pillar_shaft()` for this data type.

Usage

```
tbl_format_body(x, setup, ...)
```

Arguments

`x`	A tibble-like object.
`setup`	A setup object returned from `tbl_format_setup()`.
`...`	These dots are for future extensions and must be empty.

Value

A character vector.

Examples

```
setup <- tbl_format_setup(palmerpenguins::penguins)
tbl_format_body(palmerpenguins::penguins, setup)

# Shortcut for debugging
tbl_format_body(setup)
```

`tbl_format_footer` *Format the footer of a tibble*

Description

[Experimental]

For easier customization, the formatting of a tibble is split into three components: header, body, and footer. The `tbl_format_footer()` method is responsible for formatting the footer of a tibble.

Override or extend this method if you need to change the appearance of the footer. The default implementation adds information about rows and columns that are not shown in the body.

Usage

```
tbl_format_footer(x, setup, ...)
```

Arguments

`x`	A tibble-like object.
`setup`	A setup object returned from `tbl_format_setup()`.
`...`	These dots are for future extensions and must be empty.

Value

A character vector.

Examples

```
setup <- tbl_format_setup(palmerpenguins::penguins)
tbl_format_footer(palmerpenguins::penguins, setup)

# Shortcut for debugging
tbl_format_footer(setup)
```

`tbl_format_header` *Format the header of a tibble*

Description

[Experimental]

For easier customization, the formatting of a tibble is split into three components: header, body, and footer. The `tbl_format_header()` method is responsible for formatting the header of a tibble.

Override this method if you need to change the appearance of the entire header. If you only need to change or extend the components shown in the header, override or extend `tbl_sum()` for your class which is called by the default method.

Usage

```
tbl_format_header(x, setup, ...)
```

Arguments

`x`	A tibble-like object.
`setup`	A setup object returned from `tbl_format_setup()`.
`...`	These dots are for future extensions and must be empty.

Value

A character vector.

Examples

```
setup <- tbl_format_setup(palmerpenguins::penguins)
tbl_format_header(palmerpenguins::penguins, setup)

# Shortcut for debugging
tbl_format_header(setup)
```

`tbl_format_setup` *Set up formatting*

Description

`tbl_format_setup()` is called by `format.tbl()`. This method collects information that is common to the header, body, and footer parts of a tibble. Examples:

- the dimensions sometimes are reported both in the header and (implicitly) in the footer of a tibble;
- the columns shown in the body decide which columns are shown in the footer.

This information is computed once in `tbl_format_setup()`. The result is passed on to the `tbl_format_header()`, `tbl_format_body()`, and `tbl_format_footer()` methods. If you need to customize parts of the printed output independently, override these methods instead.

Usage

```
tbl_format_setup(x, width = NULL, ..., n = NULL, max_extra_cols = NULL)

## S3 method for class 'tbl'
tbl_format_setup(x, width, ..., n, max_extra_cols)
```

Arguments

`x`	An object.
`width`	Actual width for printing, a numeric greater than zero. This argument is mandatory for all implementations of this method.
`...`	Extra arguments to `print.tbl()` or `format.tbl()`.
`n`	Actual number of rows to print. No options should be considered by implementations of this method.
`max_extra_cols`	Number of columns to print abbreviated information for, if the width is too small for the entire tibble. No options should be considered by implementations of this method.

Details

Extend this method to prepare information that is used in several parts of the printed output of a tibble-like object, or to collect additional arguments passed via `...` to `print.tbl()` or `format.tbl()`.

We expect that `tbl_format_setup()` is extended only rarely, and overridden only in exceptional circumstances, if at all. If you override this method, you must also implement `tbl_format_header()`, `tbl_format_body()`, and `tbl_format_footer()` for your class.

Implementing a method allows to override printing and formatting of the entire object without overriding the `print()` and `format()` methods directly. This allows to keep the logic of the `width` and `n` arguments.

The default method for the `"tbl"` class collects information for standard printing for tibbles. See `new_tbl_format_setup()` for details on the returned object.

Value

An object that can be passed as `setup` argument to `tbl_format_header()`, `tbl_format_body()`, and `tbl_format_footer()`.

Examples

```
tbl_format_setup(palmerpenguins::penguins)
```

`tbl_sum` *Provide a succinct summary of an object*

Description

`tbl_sum()` gives a brief textual description of a table-like object, which should include the dimensions and the data source in the first element, and additional information in the other elements (such as grouping for **dplyr**). The default implementation forwards to `obj_sum()`.

Usage

```
tbl_sum(x)
```

Arguments

`x`	Object to summarise.

Details

This generic will be moved to **pillar**, and reexported from there as soon as it becomes available.

Value

A named character vector, describing the dimensions in the first element and the data source in the name of the first element.

See Also

`type_sum()`

Index