

# Package: pillar (via r-universe)

July 2, 2024

**Title** Coloured Formatting for Columns

**Version** 1.9.0.9013

**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 (>= 2.3.0), glue, lifecycle, rlang (>= 1.0.2), utf8 (>= 1.1.0), utils, vctrs (>= 0.5.0)

**Suggests** bit64, DBI, debugme, DiagrammeR, dplyr, formattable, ggplot2, knitr, lubridate, nanotime, nycflights13, palmerpenguins, rmarkdown, scales, stringi, survival, testthat (>= 3.1.1), tibble, units (>= 0.7.2), vdiff, withr

**VignetteBuilder** knitr

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**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

**Config/autostyle/scope** line\_breaks

**Config/autostyle/strict** true

**Config/gha/extra-packages** DiagrammeR=?ignore-before-r=3.5.0

**Config/Needs/website** tidyverse/tidytemplate

**Repository** <https://r-lib.r-universe.dev>

**RemoteUrl** <https://github.com/r-lib/pillar>

**RemoteRef** HEAD

**RemoteSha** 6dce54de2da77d33806f432cd1007ee70cd01839

## Contents

|                                |    |
|--------------------------------|----|
| pillar-package . . . . .       | 2  |
| align . . . . .                | 3  |
| ctl_new_pillar . . . . .       | 4  |
| ctl_new_pillar_list . . . . .  | 6  |
| dim_desc . . . . .             | 8  |
| format_glimpse . . . . .       | 9  |
| format_type_sum . . . . .      | 10 |
| get_extent . . . . .           | 11 |
| glimpse . . . . .              | 11 |
| new_ornament . . . . .         | 12 |
| new_pillar . . . . .           | 13 |
| new_pillar_component . . . . . | 14 |
| new_pillar_shaft . . . . .     | 15 |
| new_pillar_title . . . . .     | 17 |
| new_pillar_type . . . . .      | 17 |
| pillar . . . . .               | 18 |
| pillar_options . . . . .       | 19 |
| pillar_shaft . . . . .         | 20 |
| style_num . . . . .            | 22 |
| tbl_format_body . . . . .      | 23 |
| tbl_format_footer . . . . .    | 24 |
| tbl_format_header . . . . .    | 25 |
| tbl_format_setup . . . . .     | 26 |
| tbl_sum . . . . .              | 27 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>29</b> |
|--------------|-----------|

---

|                |  |
|----------------|--|
| pillar-package | <i>pillar: Coloured Formatting for Columns</i> |
|----------------|--|

---

## 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.

## Author(s)

**Maintainer:** Kirill Müller <kirill@cynkra.com> ([ORCID](#))

Authors:

- Hadley Wickham

Other contributors:

- RStudio [copyright holder]

**See Also**

- [pillar\(\)](#) for formatting a single column,
- [print.tbl\(\)](#) for formatting data-frame-like objects,
- [pillar\\_options](#) for a list of package options.

**Examples**

```
pillar(1:3)
pillar(c(1, 2, 3))
pillar(factor(letters[1:3]), title = "letters")
tbl_format_setup(tibble::as_tibble(mtcars), width = 60)
```

---

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 <a href="#">get_max_extent()</a> ). |
| 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")
```

---

ctl\_new\_pillar      *Customize the appearance of simple pillars in your tibble subclass*

---

## Description

### [Experimental]

Gain full control over the appearance of the pillars of your tibble subclass in its body. This method is 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_rowid_pillar(controller, x, width, ..., title = NULL, type = NULL)
```

## Arguments

|            |  |
|------------|--|
| controller | The object of class "tbl" currently printed.                                 |
| x          | A simple (one-dimensional) vector.   |
| 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.                  |
| type       | String for specifying a row ID type. Current values in use are NULL and "*". |

## 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 modify the pillar components returned from the default implementation. Override this method to completely change the appearance of the pillars. Components are created with [new\\_pillar\\_component\(\)](#) or [pillar\\_component\(\)](#). In order to customize printing of row IDs, a method can be supplied for the [ctl\\_new\\_rowid\\_pillar\(\)](#) generic.

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.

## See Also

See [ctl\\_new\\_pillar\\_list\(\)](#) for creating pillar objects for compound columns: packed data frames, matrices, or arrays.

**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
)

# 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, ...) {
  out <- NextMethod()
  new_pillar(list(
    title = out$title,
    type = out$type,
    lines = new_pillar_component(list(lines("=")), width = 1),
    data = out$data
  ))
}

ctl_new_rowid_pillar.line_tbl <- function(controller, x, width, ...) {
  out <- NextMethod()
  new_pillar(
    list(
      title = out$title,
      type = out$type,
      lines = new_pillar_component(list(lines("=")), width = 1),
      data = out$data
    ),
    width = as.integer(floor(log10(max(nrow(x), 1))) + 1)
  )
}

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

```

```

ctl_new_rowid_pillar.roman_tbl <- function(controller, x, width, ...) {
  out <- NextMethod()
  rowid <- utils::as.roman(seq_len(nrow(x)))
  width <- max(nchar(as.character(rowid)))
  new_pillar(
    list(
      title = out$title,
      type = out$type,
      data = pillar_component(
        new_pillar_shaft(list(row_ids = rowid),
          width = width,
          class = "pillar_rif_shaft"
        )
      )
    ),
    width = width
  )
}

vctrs::new_data_frame(
  list(a = 1:3, b = letters[1:3]),
  class = c("roman_tbl", "tbl")
)

```

---

ctl\_new\_pillar\_list *Customize the appearance of compound pillars in your tibble subclass*

---

## Description

### [Experimental]

Gain full control over the appearance of the pillars of your tibble subclass in its body. This method is intended for implementers of subclasses of the "tbl" class. Users will rarely need them, and we also expect the default implementation to be sufficient for the vast majority of cases.

## Usage

```

ctl_new_pillar_list(
  controller,
  x,
  width,
  ...,
  title = NULL,
  first_pillar = NULL
)

```

**Arguments**

|              |   |
|--------------|---|
| controller   | The object of class "tbl" currently printed.  |
| x            | A vector, can also be a data frame, matrix, or array.   |
| width        | The available width, can be a vector for multiple tiers. If NULL, only the first pillar is instantiated.  |
| ...          | These dots are for future extensions and must be empty.   |
| title        | The title, derived from the name of the column in the data.   |
| first_pillar | Can be passed to this method if the first pillar for a compound pillar (or the pillar itself for a simple pillar) has been constructed already. |

**Details**

ctl\_new\_pillar\_list() is called to construct a list of pillars. If x is a regular (one-dimensional) vector, the list contains one pillar constructed by [ctl\\_new\\_pillar\(\)](#). This method also works for compound columns: columns that are data frames, matrices or arrays, with the following behavior:

- If width is NULL, the method always returns a list of length one containing one pillar object that represents the first sub-column in this compound column.
- Otherwise, the returned list contains one pillar object for all sub-columns that can be fit in the available horizontal space. These pillar objects are obtained by calling [ctl\\_new\\_pillar\\_list\(\)](#) with width = NULL on each sub-column until the available width is exhausted.

This method is called to initiate the construction of all pillars in the tibble to be printed. To ensure that all packed columns that fit the available space are printed, [ctl\\_new\\_pillar\\_list\(\)](#) may be called twice on the same input: once with width = NULL, and once with width corresponding to the then known available space and with first\_pillar set to the pillar object constructed in the first call.

**Examples**

```
# Simple column
ctl_new_pillar_list(
  tibble::tibble(),
  palmerpenguins::penguins$weight[1:3],
  width = 10
)

# Packed data frame: unknown width
ctl_new_pillar_list(
  tibble::tibble(),
  palmerpenguins::penguins[1:3, ],
  width = NULL
)

# Packed data frame: known width
ctl_new_pillar_list(
  tibble::tibble(),
  palmerpenguins::penguins,
  width = 60
)
```

```
)  
  
# Deeply packed data frame with known width:  
# showing only the first sub-column even if the width is sufficient  
ctl_new_pillar_list(  
  tibble::tibble(),  
  tibble::tibble(x = tibble::tibble(b = 1, c = 2), y = 3),  
  width = 60  
)  
  
# Packed matrix: unknown width  
ctl_new_pillar_list(tibble::tibble(), matrix(1:6, ncol = 2), width = NULL)  
  
# Packed matrix: known width  
ctl_new_pillar_list(tibble::tibble(), matrix(1:6, ncol = 2), width = 60)  
  
# Packed array  
ctl_new_pillar_list(tibble::tibble(), Titanic, width = 60)
```

---

dim\_desc

*Format dimensions*

---

## Description

Multi-dimensional objects are formatted as  $a \times b \times \dots$ , 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 | <i>Format a vector for horizontal printing</i> |
|----------------|--|

---

## 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

|                  |                              |
|------------------|------------------------------|
| <code>x</code>   | A vector.                    |
| <code>...</code> | 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_type_sum | <i>Format a type summary</i> |
|-----------------|------------------------------|

---

## 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 <code>type_sum()</code>  |
| 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 | <i>Calculate display width</i> |
|------------|--------------------------------|

---

**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 | <i>Get a glimpse of your data</i> |
|---------|-----------------------------------|

---

**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 width [option](#) (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, can use ANYI styles e.g provided by the <code>cli</code> package. |
| 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, extra = deprecated())
```

## Arguments

|                         |  |
|-------------------------|--|
| <code>components</code> | A named list of components constructed with <a href="#">pillar_component()</a> . |
| <code>...</code>        | These dots are for future extensions and must be empty.                          |
| <code>width</code>      | Default width, optional.   |
| <code>class</code>      | Name of subclass.  |
| <code>extra</code>      | Deprecated.  |

## 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\\_pillar\\_list\(\)](#).

## 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

### [Experimental]

new\_pillar\_component() constructs an object of class "pillar\_component". It is used by custom `ctl_new_pillar()` methods to create pillars with nonstandard components.

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 of length one (for new_pillar_component()), or an object with "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 for pillars created by `pillar()` are: title (may be missing), type, and data. Each component is a "pillar\_component" object.

This class captures contents that can be fitted in a simple column. Compound columns are represented by multiple pillar objects, each with their own components.

## 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 | <i>Constructor for column data</i> |
|------------------|------------------------------------|

---

## 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,  
  type_sum = NULL,  
  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"),
    short_formatted = NULL
  )

```

## Arguments

|                 |   |
|-----------------|---|
| x               | An object   |
| ...             | Passed on to <code>new_pillar_shaft()</code> .  |
| width           | The maximum column width.   |
| min_width       | The minimum allowed column width, width if omitted.   |
| type_sum        | <b>[Experimental]</b><br>Override the type summary displayed at the top of the data. This argument, if given, takes precedence over the type summary provided by <code>type_sum()</code> .  |
| class           | The name of the subclass.   |
| subclass        | Deprecated, pass the <code>class</code> argument instead.   |
| formatted       | The data to show, an object coercible to <code>character</code> .   |
| align           | Alignment of the column.  |
| na              | String to use as NA value, defaults to "NA" styled with <code>style_na()</code> with fallback if color is not available.  |
| na_indent       | Indentation of NA values.   |
| shorten         | How to abbreviate the data if necessary: <ul style="list-style-type: none"> <li>• "back" (default): add an ellipsis at the end</li> <li>• "front": add an ellipsis at the front</li> <li>• "mid": add an ellipsis in the middle</li> <li>• "abbreviate": use <code>abbreviate()</code></li> </ul> |
| short_formatted | If provided, a character vector of the same length as <code>formatted</code> , to be used when the available width is insufficient to show the full output.   |

## Details

The `formatted` argument may also contain ANSI escapes to change color or other attributes of the text, provided e.g. by the `cli` package.

---

|                  |  |
|------------------|--|
| new_pillar_title | <i>Prepare a column title for formatting</i> |
|------------------|--|

---

**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(trees)))
```

---

|                 |   |
|-----------------|---|
| new_pillar_type | <i>Prepare a column type for formatting</i> |
|-----------------|---|

---

**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("a"))  
format(new_pillar_type(factor("a")))
```

---

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

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

### 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\_options                      *Package options*

---

## Description

Options that affect display of tibble-like output.

## Details

These options can be set via [options\(\)](#) and queried via [getOption\(\)](#).

## Options for the pillar package

- `pillar.print_max`: Maximum number of rows printed, default: 20. Set to `Inf` to always print all rows. For compatibility reasons, `getOption("tibble.print_max")` and `getOption("dplyr.print_max")` are also consulted, this will be soft-deprecated in pillar v2.0.0.
- `pillar.print_min`: Number of rows printed if the table has more than `print_max` rows, default: 10. For compatibility reasons, `getOption("tibble.print_min")` and `getOption("dplyr.print_min")` are also consulted, this will be soft-deprecated in pillar v2.0.0.
- `pillar.width`: Output width. Default: `NULL` (use `getOption("width")`). This can be larger than `getOption("width")`, in this case the output of the table's body is distributed over multiple tiers for wide tibbles. For compatibility reasons, `getOption("tibble.width")` and `getOption("dplyr.width")` are also consulted, this will be soft-deprecated in pillar v2.0.0.
- `pillar.max_footer_lines`: The maximum number of lines in the footer, default: 7. Set to `Inf` to turn off truncation of footer lines. The `max_extra_cols` option still limits the number of columns printed.
- `pillar.max_extra_cols`: The maximum number of columns printed in the footer, default: 100. Set to `Inf` to show all columns. Set the more predictable `max_footer_lines` to control the number of footer lines instead.
- `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 `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 `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: 20. 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: 3. 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.
- `pillar.bidi`: Set to TRUE for experimental support for bidirectional scripts. Default: FALSE. When this option is set, "left right override" and "first strong isolate" **Unicode controls** are inserted to ensure that text appears in its intended direction and that the column headings correspond to the correct columns.
- `pillar.superdigit_sep`: The string inserted between superscript digits and column names in the footnote. Defaults to a "\u200b", a zero-width space, on UTF-8 platforms, and to ":" on non-UTF-8 platforms.
- `pillar.advice`: Should advice be displayed in the footer when columns or rows are missing from the output? Defaults to TRUE for interactive sessions, and to FALSE otherwise.

## Examples

```
df <- tibble::tibble(x = c(1.234567, NA, 5:10))
df

# Change for the duration of the session:
old <- options(
  pillar.sigfig = 6,
  pillar.print_max = 5,
  pillar.print_min = 5,
  pillar.advice = FALSE
)
df

# Change back to the original value:
options(old)
df
```

## 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 'glue'
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**

|                        |   |
|------------------------|---|
| <code>x</code>         | A vector to format  |
| <code>...</code>       | Arguments passed to methods.  |
| <code>width</code>     | Width for printing and formatting.  |
| <code>sigfig</code>    | Deprecated, use <code>num()</code> or <code>set_num_opts()</code> on the data instead.  |
| <code>min_width</code> | Deprecated, use <code>char()</code> or <code>set_char_opts()</code> on the data instead.  |
| <code>na_indent</code> | Indentation of NA values.   |
| <code>shorten</code>   | How to abbreviate the data if necessary: <ul style="list-style-type: none"> <li>• "back" (default): add an ellipsis at the end</li> <li>• "front": add an ellipsis at the front</li> <li>• "mid": add an ellipsis in the middle</li> <li>• "abbreviate": use <code>abbreviate()</code></li> </ul> |

**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 `subtle` option.

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

`style_bold()` is affected by the `bold` option, which is FALSE by default.

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

**See Also**

[pillar\\_options](#) 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 | <i>Format the body of a tibble</i> |
|-----------------|------------------------------------|

---

**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_abbrev()` 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 <code>tbl_format_setup()</code> . |
| ...   | 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 | <i>Format the footer of a tibble</i> |
|-------------------|--------------------------------------|

---

**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 <code>tbl_format_setup()</code> . |
| ...   | 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 | <i>Format the header of a tibble</i> |
|-------------------|--------------------------------------|

---

## 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 <code>tbl_format_setup()</code> . |
| ...   | 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,
  max_footer_lines = NULL,
  focus = NULL
)

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

### 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 <code>print.tbl()</code> or <code>format.tbl()</code> .   |
| n                | Actual number of rows to print. No <code>options</code> 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 <code>options</code> should be considered by implementations of this method. |
| max_footer_lines | Maximum number of lines for the footer. No <code>options</code> should be considered by implementations of this method.  |
| focus            | <b>[Experimental]</b><br>Names of columns to show preferentially if space is tight.  |

## 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.

## 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\(\)](#)

**Examples**

```
tbl_sum(1:10)
tbl_sum(matrix(1:10))
tbl_sum(data.frame(a = 1))
tbl_sum(Sys.Date())
tbl_sum(Sys.time())
tbl_sum(mean)
```

# Index

- \* **datasets**
  - pillar\_options, 19
- abbreviate(), 16, 22
- align, 3
- char(), 22
- character, 16
- ctl\_new\_pillar, 4
- ctl\_new\_pillar(), 7, 13, 14
- ctl\_new\_pillar\_list, 6
- ctl\_new\_pillar\_list(), 4, 13
- ctl\_new\_rowid\_pillar (ctl\_new\_pillar), 4
- dim\_desc, 8
- format(), 12, 17, 18, 21, 22, 27
- format.pillar\_shaft (pillar\_shaft), 20
- format.tbl(), 26, 27
- format\_glimpse, 9
- format\_glimpse(), 11
- format\_type\_sum, 10
- get\_extent, 11
- get\_max\_extent (get\_extent), 11
- get\_max\_extent(), 3
- getOption(), 19
- glimpse, 11
- glimpse(), 9
- new\_ornament, 12
- new\_pillar, 13
- new\_pillar\_component, 14
- new\_pillar\_component(), 4
- new\_pillar\_shaft, 15
- new\_pillar\_shaft(), 15, 16
- new\_pillar\_shaft\_simple
  - (new\_pillar\_shaft), 15
- new\_pillar\_title, 17
- new\_pillar\_title(), 18
- new\_pillar\_type, 17
- new\_pillar\_type(), 18
- new\_tbl\_format\_setup(), 27
- num(), 22
- obj\_sum(), 27
- option, 11, 23
- options, 26
- options(), 19
- pillar, 18
- pillar(), 3, 4, 13, 14
- pillar-package, 2
- pillar\_component
  - (new\_pillar\_component), 14
- pillar\_component(), 4, 13
- pillar\_options, 3, 19, 23
- pillar\_shaft, 20
- pillar\_shaft(), 12, 18, 23
- print(), 21, 27
- print.pillar\_shaft (pillar\_shaft), 20
- print.tbl(), 3, 26, 27
- set\_char\_opts(), 22
- set\_num\_opts(), 22
- str(), 11, 12
- style\_bold (style\_num), 22
- style\_na (style\_num), 22
- style\_na(), 16
- style\_neg (style\_num), 22
- style\_num, 22
- style\_subtle (style\_num), 22
- style\_subtle\_num (style\_num), 22
- tbl\_format\_body, 23
- tbl\_format\_body(), 26, 27
- tbl\_format\_footer, 24
- tbl\_format\_footer(), 26, 27
- tbl\_format\_header, 25
- tbl\_format\_header(), 26, 27
- tbl\_format\_setup, 26

`tbl_format_setup()`, [24](#), [25](#)  
`tbl_sum`, [27](#)  
`tbl_sum()`, [25](#)  
`type_sum()`, [10](#), [16–18](#), [27](#)  
`vctrs::vec_ptype_abbr()`, [23](#)