Package: tibble (via r-universe)

March 20, 2025

Title Simple Data Frames

Version 3.2.1.9046

Description Provides a 'tbl_df' class (the 'tibble') with stricter checking and better formatting than the traditional data frame.

License MIT + file LICENSE

URL https://tibble.tidyverse.org/, https://github.com/tidyverse/tibble

BugReports https://github.com/tidyverse/tibble/issues

Depends R (>= 3.4.0)

Imports cli, lifecycle (>= 1.0.0), magrittr, methods, pillar (>= 1.8.1), pkgconfig, rlang (>= 1.0.2), utils, vctrs (>= 0.4.2)

Suggests bench, bit64, blob, brio, callr, DiagrammeR, dplyr, evaluate, formattable, ggplot2, here, hms, htmltools, knitr, lubridate, nycflights13, pkgload, purrr, rmarkdown, stringi, testthat (>= 3.0.2), tidyr, withr

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2.9000

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first vignette-formats, as_tibble, add, invariants

Config/autostyle/scope line_breaks

Config/autostyle/strict true

Config/autostyle/rmd false

Config/Needs/website tidyverse/tidytemplate

Repository https://tidyverse.r-universe.dev

RemoteUrl https://github.com/tidyverse/tibble

RemoteRef HEAD

RemoteSha fe2edfe2bcc3482e1451931dfe6a5bd4c6d9518e

Contents

`tibble-package` *tibble: Simple Data Frames*

Description

Provides a 'tbl_df' class (the 'tibble') with stricter checking and better formatting than the traditional data frame.

Details

[Stable]

The tibble package provides utilities for handling **tibbles**, where "tibble" is a colloquial term for the S3 `tbl_df` class. The `tbl_df` class is a special case of the base `data.frame` class, developed in response to lessons learned over many years of data analysis with data frames.

Tibble is the central data structure for the set of packages known as the tidyverse, including dplyr, ggplot2, tidyr, and readr.

General resources:

- Website for the tibble package: `https://tibble.tidyverse.org`
- Vectors chapter in *Advanced R* (2nd edition), specifically the Data frames and tibbles section

Resources on specific topics:

- Create a tibble: `tibble()`, `as_tibble()`, `tribble()`, `enframe()`
- Inspect a tibble: `print.tbl()`, `glimpse()`
- Details on the S3 `tbl_df` class: `tbl_df`
- Package options: tibble_options

Author(s)

Maintainer: Kirill Müller <kirill@cynkra.com> (ORCID)

Authors:

- Hadley Wickham <hadley@rstudio.com>

Other contributors:

- Romain Francois <romain@r-enthusiasts.com> [contributor]
- Jennifer Bryan <jenny@rstudio.com> [contributor]
- RStudio [copyright holder, funder]

See Also

Useful links:

- https://tibble.tidyverse.org/
- https://github.com/tidyverse/tibble
- Report bugs at https://github.com/tidyverse/tibble/issues

`add_column` *Add columns to a data frame*

Description

This is a convenient way to add one or more columns to an existing data frame.

Usage

```
add_column(
  .data,
  ...,
  .before = NULL,
  .after = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)
```

Arguments

`.data`	Data frame to append to.
`...`	<dynamic-dots> Name-value pairs, passed on to `tibble()`. All values must have the same size of `.data` or size 1.
`.before, .after`	One-based column index or column name where to add the new columns, default: after last column.
`.name_repair`	Treatment of problematic column names:

- "minimal": No name repair or checks, beyond basic existence,
- "unique": Make sure names are unique and not empty,
- "check_unique": (default value), no name repair, but check they are unique,
- "universal": Make the names unique and syntactic
- a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R).
- A purrr-style anonymous function, see rlang::as_function()

This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.

See Also

Other addition: add_row()

Examples

```
# add_column ---------------------------------
df <- tibble(x = 1:3, y = 3:1)

df %>% add_column(z = -1:1, w = 0)
df %>% add_column(z = -1:1, .before = "y")

# You can't overwrite existing columns
try(df %>% add_column(x = 4:6))

# You can't create new observations
try(df %>% add_column(z = 1:5))
```

---

add_row *Add rows to a data frame*

Description

This is a convenient way to add one or more rows of data to an existing data frame. See tribble() for an easy way to create an complete data frame row-by-row. Use tibble_row() to ensure that the new data has only one row.

add_case() is an alias of add_row().

Usage

```
add_row(.data, ..., .before = NULL, .after = NULL)
```

Arguments

.data	Data frame to append to.
...	<dynamic-dots> Name-value pairs, passed on to tibble(). Values can be defined only for columns that already exist in .data and unset columns will get an NA value.
.before, .after	One-based row index where to add the new rows, default: after last row.

See Also

Other addition: add_column()

Examples

```
# add_row ---------------------------------
df <- tibble(x = 1:3, y = 3:1)

df %>% add_row(x = 4, y = 0)

# You can specify where to add the new rows
df %>% add_row(x = 4, y = 0, .before = 2)

# You can supply vectors, to add multiple rows (this isn't
# recommended because it's a bit hard to read)
df %>% add_row(x = 4:5, y = 0:-1)

# Use tibble_row() to add one row only
df %>% add_row(tibble_row(x = 4, y = 0))
try(df %>% add_row(tibble_row(x = 4:5, y = 0:-1)))

# Absent variables get missing values
df %>% add_row(x = 4)

# You can't create new variables
try(df %>% add_row(z = 10))
```

---

as_tibble *Coerce lists, matrices, and more to data frames*

Description

as_tibble() turns an existing object, such as a data frame or matrix, into a so-called tibble, a data frame with class tbl_df. This is in contrast with tibble(), which builds a tibble from individual columns. as_tibble() is to tibble() as base::as.data.frame() is to base::data.frame().

as_tibble() is an S3 generic, with methods for:

- data.frame: Thin wrapper around the list method that implements tibble's treatment of rownames.
- matrix, poly, ts, table

- Default: Other inputs are first coerced with `base::as.data.frame()`.

`as_tibble_row()` converts a vector to a tibble with one row. If the input is a list, all elements must have size one.

`as_tibble_col()` converts a vector to a tibble with one column.

Usage

```
as_tibble(
  x,
  ...,
  .rows = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  rownames = pkgconfig::get_config("tibble::rownames", NULL)
)

## S3 method for class 'data.frame'
as_tibble(
  x,
  validate = NULL,
  ...,
  .rows = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  rownames = pkgconfig::get_config("tibble::rownames", NULL)
)

## S3 method for class 'list'
as_tibble(
  x,
  validate = NULL,
  ...,
  .rows = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)

## S3 method for class 'matrix'
as_tibble(x, ..., validate = NULL, .name_repair = NULL)

## S3 method for class 'table'
as_tibble(x, `_n` = "n", ..., n = `_n`, .name_repair = "check_unique")

## S3 method for class '`NULL`'
as_tibble(x, ...)

## Default S3 method:
as_tibble(x, ...)

as_tibble_row(
  x,
```

```
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)

as_tibble_col(x, column_name = "value")
```

Arguments

`x`	A data frame, list, matrix, or other object that could reasonably be coerced to a tibble.
`...`	Unused, for extensibility.
`.rows`	The number of rows, useful to create a 0-column tibble or just as an additional check.
`.name_repair`	Treatment of problematic column names: • `"minimal"`: No name repair or checks, beyond basic existence, • `"unique"`: Make sure names are unique and not empty, • `"check_unique"`: (default value), no name repair, but check they are `unique`, • `"universal"`: Make the names `unique` and syntactic • a function: apply custom name repair (e.g., `.name_repair = make.names` for names in the style of base R). • A purrr-style anonymous function, see `rlang::as_function()` This argument is passed on as `repair` to `vctrs::vec_as_names()`. See there for more details on these terms and the strategies used to enforce them.
`rownames`	How to treat existing row names of a data frame or matrix: • `NULL`: remove row names. This is the default. • `NA`: keep row names. • A string: the name of a new column. Existing rownames are transferred into this column and the `row.names` attribute is deleted. No name repair is applied to the new column name, even if `x` already contains a column of that name. Use `as_tibble(rownames_to_column(...))` to safeguard against this case. Read more in rownames.
`_n, validate`	**[Soft-deprecated]** For compatibility only, do not use for new code.
`n`	Name for count column, default: `"n"`.
`column_name`	Name of the column.

Row names

The default behavior is to silently remove row names.

New code should explicitly convert row names to a new column using the `rownames` argument.

For existing code that relies on the retention of row names, call `pkgconfig::set_config("tibble::rownames" = NA)` in your script or in your package's `.onLoad()` function.

Life cycle

Using `as_tibble()` for vectors is superseded as of version 3.0.0, prefer the more expressive `as_tibble_row()` and `as_tibble_col()` variants for new code.

See Also

`tibble()` constructs a tibble from individual columns. `enframe()` converts a named vector to a tibble with a column of names and column of values. Name repair is implemented using `vctrs::vec_as_names()`.

Examples

```
m <- matrix(rnorm(50), ncol = 5)
colnames(m) <- c("a", "b", "c", "d", "e")
df <- as_tibble(m)

as_tibble_row(c(a = 1, b = 2))
as_tibble_row(list(c = "three", d = list(4:5)))
as_tibble_row(1:3, .name_repair = "unique")

as_tibble_col(1:3)
as_tibble_col(
  list(c = "three", d = list(4:5)),
  column_name = "data"
)
```

---

char *Format a character vector*

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(
    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(
  back = char(lipsum, shorten = "back"),
```

```
  front = char(lipsum, shorten = "front"),
  mid = char(lipsum, shorten = "mid")
)
tibble(abbr = char(lipsum, shorten = "abbreviate"))
```

`enframe` *Converting vectors to data frames, and vice versa*

Description

`enframe()` converts named atomic vectors or lists to one- or two-column data frames. For a list, the result will be a nested tibble with a column of type `list`. For unnamed vectors, the natural sequence is used as name column.

`deframe()` converts two-column data frames to a named vector or list, using the first column as name and the second column as value. If the input has only one column, an unnamed vector is returned.

Usage

```
enframe(x, name = "name", value = "value")

deframe(x)
```

Arguments

`x`	A vector (for `enframe()`) or a data frame with one or two columns (for `deframe()`).
`name`, `value`	Names of the columns that store the names and values. If `name` is `NULL`, a one-column tibble is returned; `value` cannot be `NULL`.

Value

For `enframe()`, a tibble with two columns (if `name` is not `NULL`, the default) or one column (otherwise).

For `deframe()`, a vector (named or unnamed).

Examples

```
enframe(1:3)
enframe(c(a = 5, b = 7))
enframe(list(one = 1, two = 2:3, three = 4:6))
deframe(enframe(3:1))
deframe(tibble(a = 1:3))
deframe(tibble(a = as.list(1:3)))
```

`formatting` *Printing tibbles*

Description

One of the main features of the `tbl_df` class is the printing:

- Tibbles only print as many rows and columns as fit on one screen, supplemented by a summary of the remaining rows and columns.
- Tibble reveals the type of each column, which keeps the user informed about whether a variable is, e.g., `<chr>` or `<fct>` (character versus factor). See `vignette("types")` for an overview of common type abbreviations.

Printing can be tweaked for a one-off call by calling `print()` explicitly and setting arguments like `n` and `width`. More persistent control is available by setting the options described in pillar::pillar_options. See also `vignette("digits")` for a comparison to base options, and `vignette("numbers")` that showcases `num()` and `char()` for creating columns with custom formatting options.

As of tibble 3.1.0, printing is handled entirely by the **pillar** package. If you implement a package that extends tibble, the printed output can be customized in various ways. See `vignette("extending", package = "pillar")` for details, and pillar::pillar_options for options that control the display in the console.

Usage

```
## S3 method for class 'tbl_df'
print(
  x,
  width = NULL,
  ...,
  n = NULL,
  max_extra_cols = NULL,
  max_footer_lines = NULL
)

## S3 method for class 'tbl_df'
format(
  x,
  width = NULL,
  ...,
  n = NULL,
  max_extra_cols = NULL,
  max_footer_lines = NULL
)
```

Arguments

`x`	Object to format or print.

width	Width of text output to generate. This defaults to NULL, which means use the width option.
...	These dots are for future extensions and must be empty.
n	Number of rows to show. If NULL, the default, will print all rows if less than the print_max option. Otherwise, will print as many rows as specified by the print_min option.
max_extra_cols	Number of extra columns to print abbreviated information for, if the width is too small for the entire tibble. If NULL, the max_extra_cols option is used. The previously defined n_extra argument is soft-deprecated.
max_footer_lines	Maximum number of footer lines. If NULL, the max_footer_lines option is used.

Examples

```
print(as_tibble(mtcars))
print(as_tibble(mtcars), n = 1)
print(as_tibble(mtcars), n = 3)

print(as_tibble(trees), n = 100)

print(mtcars, width = 10)

mtcars2 <- as_tibble(cbind(mtcars, mtcars), .name_repair = "unique")
print(mtcars2, n = 25, max_extra_cols = 3)


print(nycflights13::flights, max_footer_lines = 1)
print(nycflights13::flights, width = Inf)
```

---

frame_matrix *Row-wise matrix creation*

Description

Create matrices laying out the data in rows, similar to matrix(..., byrow = TRUE), with a nicer-to-read syntax. This is useful for small matrices, e.g. covariance matrices, where readability is important. The syntax is inspired by tribble().

Usage

```
frame_matrix(...)
```

Arguments

...	<dynamic-dots> Arguments specifying the structure of a frame_matrix. Column names should be formulas, and may only appear before the data. These arguments are processed with rlang::list2() and support unquote via rlang::!! and unquote-splice via rlang::!!!.

Value

A matrix.

See Also

See rlang::quasiquotation for more details on tidy dots semantics, i.e. exactly how the ... argument is processed.

Examples

```
frame_matrix(
  ~col1, ~col2,
  1,     3,
  5,     2
)
```

`is_tibble` *Test if the object is a tibble*

Description

This function returns `TRUE` for tibbles or subclasses thereof, and `FALSE` for all other objects, including regular data frames.

Usage

```
is_tibble(x)
```

Arguments

`x`	An object

Value

`TRUE` if the object inherits from the `tbl_df` class.

`lst` *Build a list*

Description

`lst()` constructs a list, similar to `base::list()`, but with some of the same features as `tibble()`. `lst()` builds components sequentially. When defining a component, you can refer to components created earlier in the call. `lst()` also generates missing names automatically.

See `rlang::list2()` for a simpler and faster alternative without tibble's evaluation and auto-name semantics.

Usage

```
lst(...)
```

Arguments

`...`	<dynamic-dots> A set of name-value pairs. These arguments are processed with `rlang::quos()` and support unquote via `rlang::!!` and unquote-splice via `rlang::!!!`. Use `:=` to create columns that start with a dot. Arguments are evaluated sequentially. You can refer to previously created elements directly or using the rlang::.data pronoun. To refer explicitly to objects in the calling environment, use `rlang::!!` or rlang::.env, e.g. `!!.data` or `.env$.data` for the special case of an object named `.data`.

Value

A named list.

Examples

```
# the value of n can be used immediately in the definition of x
lst(n = 5, x = runif(n))

# missing names are constructed from user's input
lst(1:3, z = letters[4:6], runif(3))

a <- 1:3
b <- letters[4:6]
lst(a, b)

# pre-formed quoted expressions can be used with lst() and then
# unquoted (with !!) or unquoted and spliced (with !!!)
n1 <- 2
n2 <- 3
n_stuff <- quote(n1 + n2)
x_stuff <- quote(seq_len(n))
lst(!!!list(n = n_stuff, x = x_stuff))
```

```
lst(n = !!n_stuff, x = !!x_stuff)
lst(n = 4, x = !!x_stuff)
lst(!!!list(n = 2, x = x_stuff))
```

`new_tibble` *Tibble constructor and validator*

Description

Creates or validates a subclass of a tibble. These function is mostly useful for package authors that implement subclasses of a tibble, like **sf** or **tsibble**.

`new_tibble()` creates a new object as a subclass of `tbl_df`, `tbl` and `data.frame`. This function is optimized for performance, checks are reduced to a minimum. See `vctrs::new_data_frame()` for details.

`validate_tibble()` checks a tibble for internal consistency. Correct behavior can be guaranteed only if this function runs without raising an error.

Usage

```
new_tibble(x, ..., nrow = NULL, class = NULL, subclass = NULL)

validate_tibble(x)
```

Arguments

`x`	A tibble-like object.
`...`	Name-value pairs of additional attributes.
`nrow`	The number of rows, inferred from `x` if omitted.
`class`	Subclasses to assign to the new object, default: none.
`subclass`	Deprecated, retained for compatibility. Please use the `class` argument.

Construction

For `new_tibble()`, `x` must be a list. The `nrow` argument may be omitted as of tibble 3.1.4. If present, every element of the list `x` should have `vctrs::vec_size()` equal to this value. (But this is not checked by the constructor). This takes the place of the "row.names" attribute in a data frame. `x` must have names (or be empty), but the names are not checked for correctness.

Validation

`validate_tibble()` checks for "minimal" names and that all columns are vectors, data frames or matrices. It also makes sure that all columns have the same length, and that `vctrs::vec_size()` is consistent with the data.

See Also

tibble() and as_tibble() for ways to construct a tibble with recycling of scalars and automatic name repair, and vctrs::df_list() and vctrs::new_data_frame() for lower-level implementations.

Examples

```
# The nrow argument can be omitted:
new_tibble(list(a = 1:3, b = 4:6))

# Existing row.names attributes are ignored:
try(validate_tibble(new_tibble(trees, nrow = 3)))

# The length of all columns must be compatible with the nrow argument:
try(validate_tibble(new_tibble(list(a = 1:3, b = 4:6), nrow = 2)))
```

---

num *Format a numeric vector*

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,
  extra_sigfig = NULL
)
```

```
set_num_opts(
  x,
  ...,
  sigfig = NULL,
  digits = NULL,
  label = NULL,
  scale = NULL,
  notation = c("fit", "dec", "sci", "eng", "si"),
  fixed_exponent = NULL,
  extra_sigfig = 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 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 exponents.
extra_sigfig	If TRUE, increase the number of significant digits if the data consists of numbers of the same magnitude with subtle differences.

See Also

Other vector classes: char()

Examples

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

# Significant figures
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(
  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(
  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(
  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(
  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(
  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(
  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)
)
```

```
#' Extra significant digits
tibble(
  default = num(100 + 1:3 * 0.001),
  extra1 = num(100 + 1:3 * 0.001, extra_sigfig = TRUE),
  extra2 = num(100 + 1:3 * 0.0001, extra_sigfig = TRUE),
  extra3 = num(10000 + 1:3 * 0.00001, extra_sigfig = TRUE)
)
```

---

rownames *Tools for working with row names*

Description

While a tibble can have row names (e.g., when converting from a regular data frame), they are removed when subsetting with the [operator. A warning will be raised when attempting to assign non-NULL row names to a tibble. Generally, it is best to avoid row names, because they are basically a character column with different semantics than every other column.

These functions allow to you detect if a data frame has row names (has_rownames()), remove them (remove_rownames()), or convert them back-and-forth between an explicit column (rownames_to_column() and column_to_rownames()). Also included is rowid_to_column(), which adds a column at the start of the dataframe of ascending sequential row ids starting at 1. Note that this will remove any existing row names.

Usage

```
has_rownames(.data)

remove_rownames(.data)

rownames_to_column(.data, var = "rowname")

rowid_to_column(.data, var = "rowid")

column_to_rownames(.data, var = "rowname")
```

Arguments

.data	A data frame.
var	Name of column to use for rownames.

Value

column_to_rownames() always returns a data frame. has_rownames() returns a scalar logical. All other functions return an object of the same class as the input.

Examples

```
# Detect row names ----------------------------------------------------
has_rownames(mtcars)
has_rownames(trees)

# Remove row names ----------------------------------------------------
remove_rownames(mtcars) %>% has_rownames()

# Convert between row names and column --------------------------------
mtcars_tbl <- rownames_to_column(mtcars, var = "car") %>% as_tibble()
mtcars_tbl
column_to_rownames(mtcars_tbl, var = "car") %>% head()

# Adding rowid as a column --------------------------------------------
rowid_to_column(trees) %>% head()
```

`subsetting` *Subsetting tibbles*

Description

Accessing columns, rows, or cells via `$`, `[[`, or `[` is mostly similar to regular data frames. However, the behavior is different for tibbles and data frames in some cases:

- `[` always returns a tibble by default, even if only one column is accessed.
- Partial matching of column names with `$` and `[[` is not supported, and `NULL` is returned. For `$`, a warning is given.
- Only scalars (vectors of length one) or vectors with the same length as the number of rows can be used for assignment.
- Rows outside of the tibble's boundaries cannot be accessed.
- When updating with `[[<-` and `[<-`, type changes of entire columns are supported, but updating a part of a column requires that the new value is coercible to the existing type. See `vctrs::vec_slice()` for the underlying implementation.

Unstable return type and implicit partial matching can lead to surprises and bugs that are hard to catch. If you rely on code that requires the original data frame behavior, coerce to a data frame via `as.data.frame()`.

Usage

```
## S3 method for class 'tbl_df'
x$name

## S3 method for class 'tbl_df'
x[[i, j, ..., exact = TRUE]]
```

```
## S3 method for class 'tbl_df'
x[i, j, drop = FALSE, ...]

## S3 replacement method for class 'tbl_df'
x$name <- value

## S3 replacement method for class 'tbl_df'
x[[i, j, ...]] <- value

## S3 replacement method for class 'tbl_df'
x[i, j, ...] <- value
```

Arguments

x	A tibble.
name	A name or a string.
i, j	Row and column indices. If j is omitted, i is used as column index.
...	Ignored.
exact	Ignored, with a warning.
drop	Coerce to a vector if fetching one column via tbl[, j] . Default FALSE, ignored when accessing a column via tbl[j] .
value	A value to store in a row, column, range or cell. Tibbles are stricter than data frames in what is accepted here.

Details

For better compatibility with older code written for regular data frames, [supports a drop argument which defaults to FALSE. New code should use [[to turn a column into a vector.

Examples

```
df <- data.frame(a = 1:3, bc = 4:6)
tbl <- tibble(a = 1:3, bc = 4:6)

# Subsetting single columns:
df[, "a"]
tbl[, "a"]
tbl[, "a", drop = TRUE]
as.data.frame(tbl)[, "a"]

# Subsetting single rows with the drop argument:
df[1, , drop = TRUE]
tbl[1, , drop = TRUE]
as.list(tbl[1, ])


# Accessing non-existent columns:
df$b
tbl$b
```

```
df[["b", exact = FALSE]]
tbl[["b", exact = FALSE]]

df$bd <- c("n", "e", "w")
tbl$bd <- c("n", "e", "w")
df$b
tbl$b


df$b <- 7:9
tbl$b <- 7:9
df$b
tbl$b

# Identical behavior:
tbl[1, ]
tbl[1, c("bc", "a")]
tbl[, c("bc", "a")]
tbl[c("bc", "a")]
tbl["a"]
tbl$a
tbl[["a"]]
```

`tbl_df-class` `tbl_df` *class*

Description

The `tbl_df` class is a subclass of `data.frame`, created in order to have different default behaviour. The colloquial term "tibble" refers to a data frame that has the `tbl_df` class. Tibble is the central data structure for the set of packages known as the tidyverse, including dplyr, ggplot2, tidyr, and readr.

The general ethos is that tibbles are lazy and surly: they do less and complain more than base data.frames. This forces problems to be tackled earlier and more explicitly, typically leading to code that is more expressive and robust.

Properties of `tbl_df`

Objects of class `tbl_df` have:

- A `class` attribute of `c("tbl_df", "tbl", "data.frame")`.
- A base type of `"list"`, where each element of the list has the same `vctrs::vec_size()`.
- A `names` attribute that is a character vector the same length as the underlying list.
- A `row.names` attribute, included for compatibility with data.frame. This attribute is only consulted to query the number of rows, any row names that might be stored there are ignored by most tibble methods.

Behavior of `tbl_df`

How default behaviour of tibbles differs from that of data.frames, during creation and access:

- Column data is not coerced. A character vector is not turned into a factor. List-columns are expressly anticipated and do not require special tricks. Internal names are never stripped from column data. Read more in `tibble()`.
- Recycling only happens for a length 1 input. Read more in `vctrs::vec_recycle()`.
- Column names are not munged, although missing names are auto-populated. Empty and duplicated column names are strongly discouraged, but the user must indicate how to resolve. Read more in `vctrs::vec_as_names()`.
- Row names are not added and are strongly discouraged, in favor of storing that info as a column. Read about in rownames.
- `df[, j]` returns a tibble; it does not automatically extract the column inside. `df[, j, drop = FALSE]` is the default. Read more in subsetting.
- There is no partial matching when `$` is used to index by name. `df$name` for a nonexistent name generates a warning. Read more in subsetting.

See `vignette("invariants")` for a detailed description of the behavior.

Furthermore, printing and inspection are a very high priority. The goal is to convey as much information as possible, in a concise way, even for large and complex tibbles. Read more in formatting.

See Also

`tibble()`, `as_tibble()`, `tribble()`, `print.tbl()`, `glimpse()`

`tibble` *Build a data frame*

Description

`tibble()` constructs a data frame. It is used like `base::data.frame()`, but with a couple notable differences:

- The returned data frame has the class `tbl_df`, in addition to `data.frame`. This allows so-called "tibbles" to exhibit some special behaviour, such as enhanced printing. Tibbles are fully described in `tbl_df`.
- `tibble()` is much lazier than `base::data.frame()` in terms of transforming the user's input.
  - Character vectors are not coerced to factor.
  - List-columns are expressly anticipated and do not require special tricks.
  - Column names are not modified.
  - Inner names in columns are left unchanged.
- `tibble()` builds columns sequentially. When defining a column, you can refer to columns created earlier in the call. Only columns of length one are recycled.
- If a column evaluates to a data frame or tibble, it is nested or spliced. If it evaluates to a matrix or a array, it remains a matrix or array, respectively. See examples.

`tibble_row()` constructs a data frame that is guaranteed to occupy one row. Vector columns are required to have size one, non-vector columns are wrapped in a list.

Usage

```
tibble(
  ...,
  .rows = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)

tibble_row(
  ...,
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)
```

Arguments

`...`	<dynamic-dots> A set of name-value pairs. These arguments are processed with `rlang::quos()` and support unquote via `rlang::!!` and unquote-splice via `rlang::!!!`. Use `:=` to create columns that start with a dot. Arguments are evaluated sequentially. You can refer to previously created elements directly or using the rlang::.data pronoun. To refer explicitly to objects in the calling environment, use `rlang::!!` or rlang::.env, e.g. `!!.data` or `.env$.data` for the special case of an object named `.data`.
`.rows`	The number of rows, useful to create a 0-column tibble or just as an additional check.
`.name_repair`	Treatment of problematic column names: • `"minimal"`: No name repair or checks, beyond basic existence, • `"unique"`: Make sure names are unique and not empty, • `"check_unique"`: (default value), no name repair, but check they are `unique`, • `"universal"`: Make the names `unique` and syntactic • a function: apply custom name repair (e.g., `.name_repair = make.names` for names in the style of base R). • A purrr-style anonymous function, see `rlang::as_function()` This argument is passed on as `repair` to `vctrs::vec_as_names()`. See there for more details on these terms and the strategies used to enforce them.

Value

A tibble, which is a colloquial term for an object of class `tbl_df`. A `tbl_df` object is also a data frame, i.e. it has class `data.frame`.

See Also

Use `as_tibble()` to turn an existing object into a tibble. Use `enframe()` to convert a named vector into a tibble. Name repair is detailed in `vctrs::vec_as_names()`. See rlang::quasiquotation for more details on tidy dots semantics, i.e. exactly how the `...` argument is processed.

Examples

```
# Unnamed arguments are named with their expression:
a <- 1:5
tibble(a, a * 2)

# Scalars (vectors of length one) are recycled:
tibble(a, b = a * 2, c = 1)

# Columns are available in subsequent expressions:
tibble(x = runif(10), y = x * 2)

# tibble() never coerces its inputs,
str(tibble(letters))
str(tibble(x = list(diag(1), diag(2))))

# or munges column names (unless requested),
tibble(`a + b` = 1:5)

# but it forces you to take charge of names, if they need repair:
try(tibble(x = 1, x = 2))
tibble(x = 1, x = 2, .name_repair = "unique")
tibble(x = 1, x = 2, .name_repair = "minimal")

## By default, non-syntactic names are allowed,
df <- tibble(`a 1` = 1, `a 2` = 2)
## because you can still index by name:
df[["a 1"]]
df$`a 1`
with(df, `a 1`)

## Syntactic names are easier to work with, though, and you can request them:
df <- tibble(`a 1` = 1, `a 2` = 2, .name_repair = "universal")
df$a.1

## You can specify your own name repair function:
tibble(x = 1, x = 2, .name_repair = make.unique)

fix_names <- function(x) gsub("\\s+", "_", x)
tibble(`year 1` = 1, `year 2` = 2, .name_repair = fix_names)

## purrr-style anonymous functions and constants
## are also supported
tibble(x = 1, x = 2, .name_repair = ~ make.names(., unique = TRUE))

tibble(x = 1, x = 2, .name_repair = ~ c("a", "b"))

# Tibbles can contain columns that are tibbles or matrices
# if the number of rows is compatible. Unnamed tibbled are
# spliced, i.e. the inner columns are inserted into the
# tibble under construction.
tibble(
  a = 1:3,
```

```
  tibble(
    b = 4:6,
    c = 7:9
  ),
  d = tibble(
    e = tibble(
      f = b
    )
  )
)
tibble(
  a = 1:3,
  b = diag(3),
  c = cor(trees),
  d = Titanic[1:3, , , ]
)

# Data can not contain tibbles or matrices with incompatible number of rows:
try(tibble(a = 1:3, b = tibble(c = 4:7)))

# Use := to create columns with names that start with a dot:
tibble(.dotted := 3)

# This also works, but might break in the future:
tibble(.dotted = 3)

# You can unquote an expression:
x <- 3
tibble(x = 1, y = x)
tibble(x = 1, y = !!x)

# You can splice-unquote a list of quosures and expressions:
tibble(!!!list(x = rlang::quo(1:10), y = quote(x * 2)))

# Use .data, .env and !! to refer explicitly to columns or outside objects
a <- 1
tibble(a = 2, b = a)
tibble(a = 2, b = .data$a)
tibble(a = 2, b = .env$a)
tibble(a = 2, b = !!a)
try(tibble(a = 2, b = .env$bogus))
try(tibble(a = 2, b = !!bogus))

# Use tibble_row() to construct a one-row tibble:
tibble_row(a = 1, lm = lm(Height ~ Girth + Volume, data = trees))
```

---

tibble_options *Package options*

Description

Options that affect interactive display. See pillar::pillar_options for options that affect display on the console, and `cli::num_ansi_colors()` for enabling and disabling colored output via ANSI sequences like `[3m[38;5;246m[39m[23m`.

Usage

```
tibble_options
```

Details

These options can be set via `options()` and queried via `getOption()`. For this, add a `tibble.` prefix (the package name and a dot) to the option name. Example: for an option `foo`, use `options(tibble.foo = value)` to set it and `getOption("tibble.foo")` to retrieve the current value. An option value of `NULL` means that the default is used.

Options for the tibble package

- `view_max`: Maximum number of rows shown by `view()` if the input is not a data frame, passed on to `head()`. Default: `1000`.

Examples

```
# Default setting:
getOption("tibble.view_max")

# Change for the duration of the session:
old <- options(tibble.view_max = 100)

# view() would show only 100 rows e.g. for a lazy data frame

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

# Local scope:
local({
  rlang::local_options(tibble.view_max = 100)
  # view() would show only 100 rows e.g. for a lazy data frame
})
# view() would show the default 1000 rows e.g. for a lazy data frame
```

---

tribble *Row-wise tibble creation*

Description

Create tibbles using an easier to read row-by-row layout. This is useful for small tables of data where readability is important. Please see tibble-package for a general introduction.

Usage

```
tribble(...)
```

Arguments

...	<dynamic-dots> Arguments specifying the structure of a tibble. Variable names should be formulas, and may only appear before the data. These arguments are processed with rlang::list2() and support unquote via rlang::!! and unquote-splice via rlang::!!!.

Value

A tibble.

See Also

See rlang::quasiquotation for more details on tidy dots semantics, i.e. exactly how the ... argument is processed.

Examples

```
tribble(
  ~colA, ~colB,
  "a",   1,
  "b",   2,
  "c",   3
)

# tribble will create a list column if the value in any cell is
# not a scalar
tribble(
  ~x,  ~y,
  "a", 1:3,
  "b", 4:6
)


# Use dplyr::mutate(dplyr::across(...)) to assign an explicit type
tribble(
  ~a, ~b, ~c,
  1, "2000-01-01", "1.5"
) %>%
  dplyr::mutate(
    dplyr::across(a, as.integer),
    dplyr::across(b, as.Date)
  )
```

`view` *View an object*

Description

[Experimental]

Calls `utils::View()` on the input and returns it, invisibly. If the input is not a data frame, it is processed using a variant of `as.data.frame(head(x, n))`. A message is printed if the number of rows exceeds `n`. This function has no effect in noninteractive sessions.

Usage

```
view(x, title = NULL, ..., n = NULL)
```

Arguments

`x`	The object to display.
`title`	The title to use for the display, by default the deparsed expression is used.
`...`	Unused, must be empty.
`n`	Maximum number of rows to display. Only used if `x` is not a data frame. Uses the `view_max` option by default.

Details

The RStudio IDE overrides `utils::View()`, this is picked up correctly.