Package 'tibble'

March 15, 2019

Title Simple Data Frames

Version 2.1.1

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

License MIT + file LICENSE

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

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

Depends R (>= 3.1.0)

Imports cli (>= 1.0.1),
crayon (>= 1.3.4),
fansi (>= 0.4.0),
methods,
pillar (>= 1.3.1),
pkgconfig (>= 2.0.2),
rlang (>= 0.3.1),
utils

Suggests bench (>= 1.0.1),
covr (>= 3.2.1),
dplyr (>= 0.7.8),
htmltools (>= 0.3.6),
import (>= 1.1.0),
knitr (>= 1.21),
mockr (>= 0.1),
nycflights13 (>= 1.0.0),
rmarkdown (>= 1.11),
testthat (>= 2.0.1),
withr (>= 2.1.2)

VignetteBuilder knitr

Encoding UTF-8

LazyData yes

Roxygen list(markdown = TRUE, roclets = c(``collate'', ``namespace'', ``rd''))

RoxygenNote 6.1.1

Collate 'add.R'
'as_tibble.R'
'check-names.R'
'compat-lazyeval.R'
'compat-lifecycle.R'
'compat-name-repair.R'
'compat-purrr.R'
'tribble.R'
'deprecated.R'
'enframe.R'
'exports.R'
'glimpse.R'
'has-name.R'
'lst.R'
'msg-format.R'
'msg.R'
'new.R'
'repair-names.R'
'rownames.R'
'strrep.R'
'subsetting.R'
'tbl-df.r'
'tibble-package.R'
'tibble.R'
'type-sum.r'
'utils-format.r'
'utils.r'
'view.R'
'wrap.R'
'zzz.R'

R topics documented:

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

Description

Provides a 'tbl_df' class (the 'tibble') that provides 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`
- Tibbles chapter in *R for Data Science*

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`

Author(s)

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

Authors:

- Hadley Wickham <hadley@rstudio.com>

Other contributors:

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

See Also

Useful links:

- http://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)
```

Arguments

`.data`	Data frame to append to.
`...`	Name-value pairs, passed on to `tibble()`. All values must have one element for each row in the data frame, or be of length 1. These arguments are passed on to `tibble()`, and therefore also support unquote via `!!` and unquote-splice via `!!!`. However, unlike in **dplyr** verbs, columns in `.data` are not available for the expressions. Use `dplyr::mutate()` if you need to add a column based on existing data.
`.before, .after`	One-based column index or column name where to add the new columns, default: after last column.

See Also

Other addition: `add_row`

Examples

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

add_column(df, z = -1:1, w = 0)

# You can't overwrite existing columns
## Not run:
add_column(df, x = 4:6)

## End(Not run)
```

```
# You can't create new observations
## Not run:
add_column(df, z = 1:5)

## End(Not run)
```

`add_row` *Add rows to a data frame*

Description

Questioning

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.

`add_case()` is an alias of `add_row()`.

Usage

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

Arguments

`.data`	Data frame to append to.
`...`	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. These arguments are passed on to `tibble()`, and therefore also support unquote via `!!` and unquote-splice via `!!!`. However, unlike in **dplyr** verbs, columns in `.data` are not available for the expressions.
`.before, .after`	One-based row index where to add the new rows, default: after last row.

Life cycle

It is unclear if `add_row()` and its alias `add_cases()` should ensure that all columns have length one by wrapping in a list if necessary. See https://github.com/tidyverse/tibble/pull/503 and https://github.com/tidyverse/tibble/issues/205 for details.

See Also

Other addition: `add_column`

Examples

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

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

# You can specify where to add the new rows
add_row(df, 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)
add_row(df, x = 4:5, y = 0:-1)

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

# You can't create new variables
## Not run:
add_row(df, z = 10)

## End(Not run)
```

---

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

Description

Maturing

`as_tibble()` turns an existing object, such as a data frame, list, 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.
- list
- `matrix`, `poly`, `ts`, `table`
- Default: An atomic vector is first coerced to a list and, unlike `base::as.data.frame()`, the returned tibble has one column per element. Other inputs are first coerced with `base::as.data.frame()`.

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`)

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

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

Arguments

x	A data frame, list, matrix, or other object that could reasonably be coerced to a tibble.
...	Other arguments passed on to individual methods.
.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() See name-repair 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. Read more in rownames.
_n, validate	For compatibility only, do not use for new code.
n	Name for count column, default: "n".

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.

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 documents the details of name repair.

Examples

```
l <- list(x = 1:500, y = runif(500), z = 500:1)
df <- as_tibble(l)

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

as_tibble(as.list(1:3), .name_repair = "unique")

# Prefer enframe() for vectors
enframe(1:3)
enframe(1:3, name = NULL)

# For list-like inputs, `as_tibble()` is considerably simpler than
# `as.data.frame()` and hence faster
## Not run:
if (requireNamespace("bench", quietly = TRUE)) {
  l2 <- replicate(26, sample(letters), simplify = FALSE)
  names(l2) <- letters
  bench::mark(
    as_tibble(l2, .name_repair = "universal"),
    as_tibble(l2, .name_repair = "unique"),
    as_tibble(l2, .name_repair = "minimal"),
    as_tibble(l2),
    as.data.frame(l2),
    check = FALSE
  )
}

## End(Not run)
```

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

Description

Maturing

`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`	An atomic 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

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

Examples

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

formatting *Printing tibbles*

Description

Maturing

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).

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 below.

Usage

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

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

trunc_mat(x, n = NULL, width = NULL, n_extra = NULL)
```

Arguments

x	Object to format or print.
...	Other arguments passed on to individual methods.
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.
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.
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, will print information about at most tibble.max_extra_cols extra columns.

Package options

Options used by the tibble and pillar packages to format and print tbl_df objects. Used by the formatting workhorse trunc_mat() and, therefore, indirectly, by print.tbl().

- tibble.print_max: Row number threshold: Maximum number of rows printed. Set to Inf to always print all rows. Default: 20.

- `tibble.print_min`: Number of rows printed if row number threshold is exceeded. Default: 10.
- `tibble.width`: Output width. Default: `NULL` (use `width` option).
- `tibble.max_extra_cols`: Number of extra columns printed in reduced form. Default: 100.
- `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.

Examples

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

print(as_tibble(iris), n = 100)

print(mtcars, width = 10)

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

trunc_mat(mtcars)

if (requireNamespace("nycflights13", quietly = TRUE)) {
  print(nycflights13::flights, n_extra = 2)
  print(nycflights13::flights, width = Inf)
}
```

`frame_matrix` *Row-wise matrix creation*

Description

Maturing

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

`...`	Arguments specifying the structure of a `frame_matrix`. Column names should be formulas, and may only appear before the data. These arguments support tidy dots.

Value

A matrix.

Examples

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

---

glimpse *Get a glimpse of your data*

Description

Maturing

This 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.)

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.
`...`	Other arguments passed on to individual methods.

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)

if (requireNamespace("nycflights13", quietly = TRUE)) {
  glimpse(nycflights13::flights)
}
```

---

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

Questioning

`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.

Usage

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

Arguments

...	A set of name-value pairs. Arguments are evaluated sequentially, so you can refer to previously created elements. These arguments are processed with `rlang::quos()` and support unquote via `!!` and unquote-splice via `!!!`. Use `:=` to create columns that start with a dot.

Value

A named list.

Life cycle

The `lst()` function is in the questioning stage. It is essentially `rlang::list2()`, but with a couple features copied from `tibble()`. It's not clear that a function for creating lists belongs in the tibble package. Consider using `rlang::list2()` instead.

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))
```

`name-repair` *Repair the names of a vector*

Description

Maturing

tibble deals with a few levels of name repair:

- `minimal` names exist. The `names` attribute is not `NULL`. The name of an unnamed element is `""` and never `NA`. Tibbles created by the tibble package have names that are, at least, `minimal`.

- `unique` names are `minimal`, have no duplicates, and can be used where a variable name is expected. Empty names, and `...` or `..` followed by a sequence of digits are banned.
  - All columns can be accessed by name via `df[["name"]]` and `df$\`name\`` and `with(df, \`name\`)`.
- `universal` names are `unique` and syntactic (see Details for more).
  - Names work everywhere, without quoting: `df$name` and `with(df, name)` and `lm(name1 ~ name2, data = df)` and `dplyr::select(df, name)` all work.

`universal` implies `unique`, `unique` implies `minimal`. These levels are nested.

The `.name_repair` argument of `tibble()` and `as_tibble()` refers to these levels. Alternatively, the user can pass their own name repair function. It should anticipate `minimal` names as input and should, likewise, return names that are at least `minimal`.

The existing functions `tidy_names()`, `set_tidy_names()`, and `repair_names()` are soft-deprecated.

`minimal` names

`minimal` names exist. The `names` attribute is not `NULL`. The name of an unnamed element is `""` and never `NA`.

Examples:

```
Original names of a vector with length 3: NULL
                           minimal names: "" "" ""

                          Original names: "x" NA
                           minimal names: "x" ""
```

Request `.name_repair = "minimal"` to suppress almost all name munging. This is useful when the first row of a data source – allegedly variable names – actually contains *data* and the resulting tibble is destined for reshaping with, e.g., `tidyr::gather()`.

`unique` names

`unique` names are `minimal`, have no duplicates, and can be used (possibly with backticks) in contexts where a variable is expected. Empty names, and `...` or `..` followed by a sequence of digits are banned If a data frame has `unique` names, you can index it by name, and also access the columns by name. In particular, `df[["name"]]` and `df$\`name\`` and also `with(df, \`name\`)` always work.

There are many ways to make names `unique`. We append a suffix of the form `...j` to any name that is `""` or a duplicate, where `j` is the position. We also change `..#` and `...` to `...#`.

Example:

```
Original names:     ""     "x"     "" "y"     "x"  "..2"  "..."
  unique names: "...1" "x...2" "...3" "y" "x...5" "...6" "...7"
```

Pre-existing suffixes of the form `...j` are always stripped, prior to making names `unique`, i.e. reconstructing the suffixes. If this interacts poorly with your names, you should take control of name repair.

`universal` names

`universal` names are `unique` and syntactic, meaning they:

- Are never empty (inherited from `unique`).
- Have no duplicates (inherited from `unique`).
- Are not `...`. Do not have the form `..i`, where `i` is a number (inherited from `unique`).
- Consist of letters, numbers, and the dot `.` or underscore `_` characters.
- Start with a letter or start with the dot `.` not followed by a number.
- Are not a reserved word, e.g., `if` or `function` or `TRUE`.

If a data frame has `universal` names, variable names can be used "as is" in code. They work well with nonstandard evaluation, e.g., `df$name` works.

Tibble has a different method of making names syntactic than `base::make.names()`. In general, tibble prepends one or more dots `.` until the name is syntactic.

Examples:

```
 Original names:     ""     "x"    NA      "x"
universal names: "...1" "x...2" "...3" "x...4"

  Original names: "(y)"  "_z"  ".2fa"  "FALSE"
 universal names: ".y." "._z" "..2fa" ".FALSE"
```

See Also

`rlang::names2()` returns the names of an object, after making them `minimal`.

The Names attribute section in the "tidyverse package development principles".

Examples

```
## Not run:
## by default, duplicate names are not allowed
tibble(x = 1, x = 2)

## End(Not run)
## you can authorize duplicate names
tibble(x = 1, x = 2, .name_repair = "minimal")
## or request that the names be made unique
tibble(x = 1, x = 2, .name_repair = "unique")

## 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`

## 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("%", " percent", x)
tibble(`25%` = 1, `75%` = 2, .name_repair = fix_names)

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"))

## the names attibute will be non-NULL, with "" as the default element
df <- as_tibble(list(1:3, letters[1:3]), .name_repair = "minimal")
names(df)
```

`new_tibble` *Tibble constructor and validator*

Description

Maturing

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.

`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, class = NULL, subclass = NULL)

validate_tibble(x)
```

Arguments

`x`	A tibble-like object
`...`	Passed on to `structure()`
`nrow`	The number of rows, required
`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 `...` argument allows adding more attributes to the subclass. An `nrow` argument is required. This should be an integer of length 1, and every element of the list `x` should have `NROW()` 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 `NROW()` is consistent with the data. 1d arrays are not supported.

See Also

`tibble()` and `as_tibble()` for ways to construct a tibble with recycling of scalars and automatic name repair.

Examples

```
# The nrow argument is always required:
new_tibble(list(a = 1:3, b = 4:6), nrow = 3)

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

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

---

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 to 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

```
has_rownames(mtcars)
has_rownames(iris)
has_rownames(remove_rownames(mtcars))

head(rownames_to_column(mtcars))

mtcars_tbl <- as_tibble(rownames_to_column(mtcars))
mtcars_tbl
head(column_to_rownames(mtcars_tbl))
```

---

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, a warning is given and NULL is returned.

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[[i, j, ..., exact = TRUE]]

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

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

Arguments

x	data frame.
i, j	Row and column indices. If j is omitted, i is used as column index.
...	Ignored.
exact	Ignored, with a warning.
name	A literal character string or a name (possibly backtick quoted).
drop	Coerce to a vector if fetching one column via tbl[, j] . Default FALSE, ignored when accessing a column via tbl[j] .

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 `NROW()`.
- A `names` attribute that is a character vector the same length as the underlying list.
- A `row.names` attribute, included for compatibility with the base data.frame class. 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. Read more in `tibble()`.

- Recycling only happens for a length 1 input.
- 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 name-repair.
- 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.
- 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()`

`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 `pillar::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

`pillar::type_sum()`, `pillar::is_vector_s3()`

`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.
- `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.

Usage

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

Arguments

`...`	A set of name-value pairs. Arguments are evaluated sequentially, so you can refer to previously created elements. These arguments are processed with `rlang::quos()` and support unquote via `!!` and unquote-splice via `!!!`. Use `:=` to create columns that start with a dot.
`.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()` See name-repair 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 tibble. Name repair is detailed in name-repair. rlang::list2() provides more details on tidy dots semantics, i.e. exactly how quasiquotation works for the ... argument.

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 consistent:
tibble(
  a = 1:3,
  b = tibble(
    c = 4:6,
    d = 7:9
  ),
  e = tibble(
    f = tibble(
      g = letters[1:3]
    )
  )
)
tibble(
  a = 1:4,
  b = diag(4),
  c = cov(iris[1:4])
)

# data can not contain POSIXlt columns, or tibbles or matrices
# with inconsistent number of rows:
try(tibble(y = strptime("2000/01/01", "%x")))
try(tibble(a = 1:3, b = tibble(c = 4:7)))

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

`tribble` *Row-wise tibble creation*

Description

Maturing

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

`...`	Arguments specifying the structure of a `tibble`. Variable names should be formulas, and may only appear before the data. These arguments support tidy dots.

Value

A tibble.

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
)
```

`view` *View an object*

Description

Experimental

Calls `utils::View()` on the input and returns it, invisibly. The RStudio IDE overrides `utils::View()`, this is picked up correctly.

Usage

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

Arguments

`x`	The object to display.
`title`	The title to use for the display, by default the deparsed expression is used.
`...`	Unused, for extensibility.

Index