# Package 'vroom'

January 29, 2021

**Title** Read and Write Rectangular Text Data Quickly

**Version** 1.4.0

**Description** The goal of 'vroom' is to read and write data (like 'csv', 'tsv' and 'fwf') quickly. When reading it uses a quick initial indexing step, then reads the values lazily , so only the data you actually use needs to be read. The writer formats the data in parallel and writes to disk asynchronously from formatting.

**License** GPL-3

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

**BugReports** https://github.com/r-lib/vroom/issues

**Depends** R (>= 3.1)

**Imports** bit64,
  crayon,
  glue,
  hms,
  lifecycle,
  rlang (>= 0.4.2),
  stats,
  tibble (>= 2.0.0),
  vctrs (>= 0.2.0),
  tidyselect,
  withr

**Suggests** bench (>= 1.1.0),
  covr,
  curl,
  dplyr,
  forcats,
  fs,
  ggplot2,
  knitr,
  patchwork,
  prettyunits,
  purrr,
  readr (>= 1.3.1),
  rmarkdown,
  rstudioapi,
  scales,

> spelling,
> testthat (>= 2.1.0),
> tidyr,
> waldo,
> xml2

**LinkingTo** progress (>= 1.2.1),
  cpp11 (>= 0.2.0)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Copyright** file COPYRIGHTS

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

**SystemRequirements** C++11

# R **topics documented:**

---

| cols | *Create column specification* |
| --- | --- |

---

## Description

`cols()` includes all columns in the input data, guessing the column types as the default. `cols_only()` includes only the columns you explicitly specify, skipping the rest.

## Usage

```
cols(..., .default = col_guess(), .delim = NULL)

cols_only(...)

col_logical(...)

col_integer(...)

col_big_integer(...)

col_double(...)

col_character(...)

col_skip(...)

col_number(...)

col_guess(...)

col_factor(levels = NULL, ordered = FALSE, include_na = FALSE, ...)

col_datetime(format = "", ...)

col_date(format = "", ...)

col_time(format = "", ...)
```

## Arguments

| | |
| --- | --- |
| `...` | Either column objects created by col_*(), or their abbreviated character names (as described in the `col_types` argument of [`vroom()`](#)). If you're only overriding a few columns, it's best to refer to columns by name. If not named, the column types must match the column names exactly. In col_*() functions these are stored in the object. |
| `.default` | Any named columns not explicitly overridden in `...` will be read with this column type. |
| `.delim` | The delimiter to use when parsing. If the `delim` argument used in the call to vroom() it takes precedence over the one specified in `col_types`. |

| | |
|---|---|
| levels | Character vector providing set of allowed levels. if NULL, will generate levels based on the unique values of x, ordered by order of appearance in x. |
| ordered | Is it an ordered factor? |
| include_na | If NA are present, include as an explicit factor to level? |
| format | A format specification, as described below. If set to "", date times are parsed as ISO8601, dates and times used the date and time formats specified in the locale(). |
| | Unlike strptime(), the format specification must match the complete string. |

## Details

The available specifications are: (with string abbreviations in brackets)

- col_logical() [l], containing only T, F, TRUE or FALSE.
- col_integer() [i], integers.
- col_big_integer() [I], Big Integers (64bit), requires the bit64 package.
- col_double() [d], doubles.
- col_character() [c], everything else.
- col_factor(levels, ordered) [f], a fixed set of values.
- col_date(format = "") [D]: with the locale's date_format.
- col_time(format = "") [t]: with the locale's time_format.
- col_datetime(format = "") [T]: ISO8601 date times
- col_number() [n], numbers containing the grouping_mark
- col_skip() [_, -], don't import this column.
- col_guess() [?], parse using the "best" type based on the input.

## Examples

```
cols(a = col_integer())
cols_only(a = col_integer())

# You can also use the standard abbreviations
cols(a = "i")
cols(a = "i", b = "d", c = "_")

# You can also use multiple sets of column definitions by combining
# them like so:

t1 <- cols(
  column_one = col_integer(),
  column_two = col_number())

t2 <- cols(
 column_three = col_character())

t3 <- t1
t3$cols <- c(t1$cols, t2$cols)
t3
```

---

cols_condense  *Examine the column specifications for a data frame*

---

### Description

`cols_condense()` takes a spec object and condenses its definition by setting the default column type to the most frequent type and only listing columns with a different type.

`spec()` extracts the full column specification from a tibble created by readr.

### Usage

```
cols_condense(x)

spec(x)
```

### Arguments

x                 The data frame object to extract from

### Value

A col_spec object.

### Examples

```
df <- vroom(vroom_example("mtcars.csv"))
s <- spec(df)
s

cols_condense(s)
```

---

date_names  *Create or retrieve date names*

---

### Description

When parsing dates, you often need to know how weekdays of the week and months are represented as text. This pair of functions allows you to either create your own, or retrieve from a standard list. The standard list is derived from ICU (<http://site.icu-project.org>) via the *stringi* package.

### Usage

```
date_names(mon, mon_ab = mon, day, day_ab = day, am_pm = c("AM", "PM"))

date_names_lang(language)

date_names_langs()
```

## Arguments

| | |
|---|---|
| `mon`, `mon_ab` | Full and abbreviated month names. |
| `day`, `day_ab` | Full and abbreviated week day names. Starts with Sunday. |
| `am_pm` | Names used for AM and PM. |
| `language` | A BCP 47 locale, made up of a language and a region, e.g. `"en_US"` for American English. See `date_names_langs()` for a complete list of available locales. |

## Examples

```
date_names_lang("en")
date_names_lang("ko")
date_names_lang("fr")
```

---

| generators | *Generate individual vectors of the types supported by vroom* |
|---|---|

---

## Description

Generate individual vectors of the types supported by vroom

## Usage

```
gen_character(n, min = 5, max = 25, values = c(letters, LETTERS, 0:9), ...)

gen_double(n, f = stats::rnorm, ...)

gen_number(n, f = stats::rnorm, ...)

gen_integer(n, min = 1L, max = .Machine$integer.max, prob = NULL, ...)

gen_factor(
  n,
  levels = NULL,
  ordered = FALSE,
  num_levels = gen_integer(1L, 1L, 25L),
  ...
)

gen_time(n, min = 0, max = hms::hms(days = 1), fractional = FALSE, ...)

gen_date(n, min = as.Date("2001-01-01"), max = as.Date("2021-01-01"), ...)

gen_datetime(
  n,
  min = as.POSIXct("2001-01-01"),
  max = as.POSIXct("2021-01-01"),
  tz = "UTC",
  ...
)
```

```
gen_logical(n, ...)

gen_name(n)
```

## Arguments

| | |
|---|---|
| n | The size of the vector to generate |
| min | The minimum range for the vector |
| max | The maximum range for the vector |
| values | The explicit values to use. |
| ... | Additional arguments passed to internal generation functions |
| f | The random function to use. |
| prob | a vector of probability weights for obtaining the elements of the vector being sampled. |
| levels | The explicit levels to use, if NULL random levels are generated using gen_name(). |
| ordered | Should the factors be ordered factors? |
| num_levels | The number of factor levels to generate |
| fractional | Whether to generate times with fractional seconds |
| tz | The timezone to use for dates |

## Examples

```
# characters
gen_character(4)

# factors
gen_factor(4)

# logical
gen_logical(4)

# numbers
gen_double(4)
gen_integer(4)

# temporal data
gen_time(4)
gen_date(4)
gen_datetime(4)
```

---

  gen_tbl                  *Generate a random tibble*

---

## Description

This is useful for benchmarking, but also for bug reports when you cannot share the real dataset.

## Usage

```
gen_tbl(
  rows,
  cols = NULL,
  col_types = NULL,
  locale = default_locale(),
  missing = 0
)
```

## Arguments

| | |
|---|---|
| rows | Number of rows to generate |
| cols | Number of columns to generate, if NULL this is derived from col_types. |
| col_types | One of NULL, a [cols()](#) specification, or a string. See vignette("readr") for more details. |

> If NULL, all column types will be imputed from the first 1000 rows on the input. This is convenient (and fast), but not robust. If the imputation fails, you'll need to supply the correct types yourself.
>
> If a column specification created by [cols()](#), it must contain one column specification for each column. If you only want to read a subset of the columns, use [cols_only()](#).
>
> Alternatively, you can use a compact string representation where each character represents one column:
>
> - c = character
> - i = integer
> - n = number
> - d = double
> - l = logical
> - f = factor
> - D = date
> - T = date time
> - t = time
> - ? = guess
> - _ or - = skip
>   By default, reading a file without a column specification will print a message showing what readr guessed they were. To remove this message, use col_types = cols().

| locale | The locale controls defaults that vary from place to place. The default locale is US-centric (like R), but you can use [locale()](#) to create your own locale that controls things like the default time zone, encoding, decimal mark, big mark, and day/month names. |
|---|---|
| missing | The percentage (from 0 to 1) of missing data to use |

## Details

There is also a family of functions to generate individual vectors of each type.

## See Also

[generators](#) to generate individual vectors.

## Examples

```
# random 10 x 5 table with random column types
rand_tbl <- gen_tbl(10, 5)
rand_tbl

# all double 25 x 4 table
dbl_tbl <- gen_tbl(25, 4, col_types = "dddd")
dbl_tbl

# Use the dots in long form column types to change the random function and options
types <- rep(times = 4, list(col_double(f = stats::runif, min = -10, max = 25)))
types
dbl_tbl2 <- gen_tbl(25, 4, col_types = types)
dbl_tbl2
```

---

| guess_type | *Guess the type of a vector* |
|---|---|

---

## Description

Guess the type of a vector

## Usage

```
guess_type(
  x,
  na = c("", "NA"),
  locale = default_locale(),
  guess_integer = FALSE
)
```

## Arguments

| | |
|---|---|
| x | Character vector of values to parse. |
| na | Character vector of strings to interpret as missing values. Set this option to `character()` to indicate no missing values. |
| locale | The locale controls defaults that vary from place to place. The default locale is US-centric (like R), but you can use [locale()] to create your own locale that controls things like the default time zone, encoding, decimal mark, big mark, and day/month names. |
| guess_integer | If TRUE, guess integer types for whole numbers, if FALSE guess numeric type for all numbers. |

## Examples

```
# Logical vectors
guess_type(c("FALSE", "TRUE", "F", "T"))
# Integers and doubles
guess_type(c("1","2","3"))
guess_type(c("1.6","2.6","3.4"))
# Numbers containing grouping mark
guess_type("1,234,566")
```

```
# ISO 8601 date times
guess_type(c("2010-10-10"))
guess_type(c("2010-10-10 01:02:03"))
guess_type(c("01:02:03 AM"))
```

---

locale                          *Create locales*

---

### Description

A locale object tries to capture all the defaults that can vary between countries. You set the locale in once, and the details are automatically passed on down to the columns parsers. The defaults have been chosen to match R (i.e. US English) as closely as possible. See vignette("locales") for more details.

### Usage

```
locale(
  date_names = "en",
  date_format = "%AD",
  time_format = "%AT",
  decimal_mark = ".",
  grouping_mark = ",",
  tz = "UTC",
  encoding = "UTF-8"
)

default_locale()
```

### Arguments

| | |
|---|---|
| date_names | Character representations of day and month names. Either the language code as string (passed on to [date_names_lang()](#)) or an object created by [date_names()](#). |
| date_format, time_format | |
| | Default date and time formats. |
| decimal_mark, grouping_mark | |
| | Symbols used to indicate the decimal place, and to chunk larger numbers. Decimal mark can only be , or . . |
| tz | Default tz. This is used both for input (if the time zone isn't present in individual strings), and for output (to control the default display). The default is to use "UTC", a time zone that does not use daylight savings time (DST) and hence is typically most useful for data. The absence of time zones makes it approximately 50x faster to generate UTC times than any other time zone. |
| | Use "" to use the system default time zone, but beware that this will not be reproducible across systems. |
| | For a complete list of possible time zones, see [OlsonNames()](#). Americans, note that "EST" is a Canadian time zone that does not have DST. It is *not* Eastern Standard Time. It's better to use "US/Eastern", "US/Central" etc. |
| encoding | Default encoding. |

## Examples

```
locale()
locale("fr")

# South American locale
locale("es", decimal_mark = ",")
```

---

problems                          *Retrieve parsing problems*

---

### Description

vroom will only fail to parse a file if the file is invalid in a way that is unrecoverable. However there are a number of non-fatal problems that you might want to know about. You can retrieve a data frame of these problems with this function.

### Usage

```
problems(x)
```

### Arguments

x                    A data frame from `vroom::vroom()`.

### Value

A data frame with one row for each problem and four columns:

- row,col - Row and column of problem

- expected - What vroom expected to find

- actual - What it actually found

- file - The file with the problem

---

vroom                          *Read a delimited file into a tibble*

---

### Description

Read a delimited file into a tibble

## Usage

```
vroom(
  file,
  delim = NULL,
  col_names = TRUE,
  col_types = NULL,
  col_select = NULL,
  id = NULL,
  skip = 0,
  n_max = Inf,
  na = c("", "NA"),
  quote = "\"",
  comment = "",
  trim_ws = TRUE,
  escape_double = TRUE,
  escape_backslash = FALSE,
  locale = default_locale(),
  guess_max = 100,
  altrep = TRUE,
  altrep_opts = deprecated(),
  num_threads = vroom_threads(),
  progress = vroom_progress(),
  .name_repair = "unique"
)
```

## Arguments

| | |
|---|---|
| file | path to a local file. |
| delim | One or more characters used to delimit fields within a file. If NULL the delimiter is guessed from the set of c(",","\t"," ","|",":",";"). |
| col_names | Either TRUE, FALSE or a character vector of column names. |
| | If TRUE, the first row of the input will be used as the column names, and will not be included in the data frame. If FALSE, column names will be generated automatically: X1, X2, X3 etc. |
| | If col_names is a character vector, the values will be used as the names of the columns, and the first row of the input will be read into the first row of the output data frame. |
| | Missing (NA) column names will generate a warning, and be filled in with dummy names X1, X2 etc. Duplicate column names will generate a warning and be made unique with a numeric suffix. |
| col_types | One of NULL, a [cols()](#) specification, or a string. See vignette("readr") for more details. |
| | If NULL, all column types will be imputed from the first 1000 rows on the input. This is convenient (and fast), but not robust. If the imputation fails, you'll need to supply the correct types yourself. |
| | If a column specification created by [cols()](#), it must contain one column specification for each column. If you only want to read a subset of the columns, use [cols_only()](#). |
| | Alternatively, you can use a compact string representation where each character represents one column: |

- c = character
- i = integer
- n = number
- d = double
- l = logical
- f = factor
- D = date
- T = date time
- t = time
- ? = guess
- _ or - = skip

  By default, reading a file without a column specification will print a message showing what `readr` guessed they were. To remove this message, use `col_types = cols()`.

| | |
|---|---|
| col_select | One or more selection expressions, like in `dplyr::select()`. Use `c()` or `list()` to use more than one expression. See `?dplyr::select` for details on available selection options. |
| id | Either a string or 'NULL'. If a string, the output will contain a variable with that name with the filename(s) as the value. If 'NULL', the default, no variable will be created. |
| skip | Number of lines to skip before reading data. |
| n_max | Maximum number of records to read. |
| na | Character vector of strings to interpret as missing values. Set this option to `character()` to indicate no missing values. |
| quote | Single character used to quote strings. |
| comment | A string used to identify comments. Any text after the comment characters will be silently ignored. |
| trim_ws | Should leading and trailing whitespace be trimmed from each field before parsing it? |
| escape_double | Does the file escape quotes by doubling them? i.e. If this option is `TRUE`, the value '""' represents a single quote, '"'. |
| escape_backslash | |
| | Does the file use backslashes to escape special characters? This is more general than `escape_double` as backslashes can be used to escape the delimiter character, the quote character, or to add special characters like \\n. |
| locale | The locale controls defaults that vary from place to place. The default locale is US-centric (like R), but you can use [locale()](locale()) to create your own locale that controls things like the default time zone, encoding, decimal mark, big mark, and day/month names. |
| guess_max | Maximum number of records to use for guessing column types. |
| altrep | Control which column types use Altrep representations, either a character vector of types, `TRUE` or `FALSE`. See [vroom_altrep()](vroom_altrep()) for for full details. |
| altrep_opts | **Deprecated** |
| num_threads | Number of threads to use when reading and materializing vectors. If your data contains newlines within fields the parser will automatically be forced to use a single thread only. |

progress          Display a progress bar? By default it will only display in an interactive session
                  and not while knitting a document. The display is updated every 50,000 values
                  and will only display if estimated reading time is 5 seconds or more. The auto-
                  matic progress bar can be disabled by setting option readr.show_progress to
                  FALSE.

.name_repair      Handling of column names. By default, vroom ensures column names are not
                  empty and unique. See .name_repair as documented in [tibble::tibble()]
                  for additional options including supplying user defined name repair functions.

## Examples

```
# get path to example file
input_file <- vroom_example("mtcars.csv")
input_file


# Read from a path


# Input sources ---------------------------------------------------------
# Read from a path
vroom(input_file)
# You can also use literal paths directly
# vroom("mtcars.csv")


## Not run:
# Including remote paths
vroom("https://github.com/r-lib/vroom/raw/master/inst/extdata/mtcars.csv")


## End(Not run)


# Or directly from a string (must contain a trailing newline)
vroom("x,y\n1,2\n3,4\n")


# Column selection ------------------------------------------------------
# Pass column names or indexes directly to select them
vroom(input_file, col_select = c(model, cyl, gear))
vroom(input_file, col_select = c(1, 3, 11))


# Or use the selection helpers
vroom(input_file, col_select = starts_with("d"))


# You can also rename specific columns
vroom(input_file, col_select = list(car = model, everything()))


# Column types ----------------------------------------------------------
# By default, vroom guesses the columns types, looking at 1000 rows
# throughout the dataset.
# You can specify them explcitly with a compact specification:
vroom("x,y\n1,2\n3,4\n", col_types = "dc")


# Or with a list of column types:
vroom("x,y\n1,2\n3,4\n", col_types = list(col_double(), col_character()))


# File types ------------------------------------------------------------
# csv
vroom("a,b\n1.0,2.0\n", delim = ",")
# tsv
```

```
vroom("a\tb\n1.0\t2.0\n")
# Other delimiters
vroom("a|b\n1.0|2.0\n", delim = "|")

# Read datasets across multiple files -------------------------------------
mtcars_by_cyl <- vroom_example(vroom_examples("mtcars-"))
mtcars_by_cyl

# Pass the filenames directly to vroom, they are efficiently combined
vroom(mtcars_by_cyl)
```

---

vroom_altrep                      *Show which column types are using Altrep*

---

### Description

vroom_altrep() can be used directly as input to the altrep argument of [vroom()](#).

### Usage

```
vroom_altrep(which = NULL)
```

### Arguments

which          A character vector of column types to use Altrep for. Can also take TRUE or
               FALSE to use Altrep for all possible or none of the types

### Details

Alternatively there is also a family of environment variables to control use of the Altrep framework.
These can then be set in your .Renviron file, e.g. with usethis::edit_r_environ(). For versions of R where the Altrep framework is unavailable (R < 3.5.0) they are automatically turned off
and the variables have no effect. The variables can take one of true, false, TRUE, FALSE, 1, or 0.

- VROOM_USE_ALTREP_NUMERICS - If set use Altrep for *all* numeric types (default false).

There are also individual variables for each type. Currently only VROOM_USE_ALTREP_CHR defaults
to true.

- VROOM_USE_ALTREP_CHR
- VROOM_USE_ALTREP_FCT
- VROOM_USE_ALTREP_INT
- VROOM_USE_ALTREP_BIG_INT
- VROOM_USE_ALTREP_DBL
- VROOM_USE_ALTREP_NUM
- VROOM_USE_ALTREP_LGL
- VROOM_USE_ALTREP_DTTM
- VROOM_USE_ALTREP_DATE
- VROOM_USE_ALTREP_TIME

## Examples

```
vroom_altrep()
vroom_altrep(c("chr", "fct", "int"))
vroom_altrep(TRUE)
vroom_altrep(FALSE)
```

---

vroom_altrep_opts            *Show which column types are using Altrep*

---

## Description

**Deprecated** This function is deprecated in favor of `vroom_altrep()`.

## Usage

```
vroom_altrep_opts(which = NULL)
```

## Arguments

which              A character vector of column types to use Altrep for. Can also take TRUE or
                   FALSE to use Altrep for all possible or none of the types

---

vroom_example               *Get path to vroom examples*

---

## Description

vroom comes bundled with a number of sample files in its 'inst/extdata' directory. Use `vroom_examples()`
to list all the available examples and `vroom_example()` to retrieve the path to one example.

## Usage

```
vroom_example(path)

vroom_examples(pattern = NULL)
```

## Arguments

path               Name of file.

pattern            A regular expression of filenames to match. If NULL all available files are re-
                   turned. listed.

## Examples

```
# List all available examples
vroom_examples()

# Get path to one example
vroom_example("mtcars.csv")
```

---

vroom_format *Convert a data frame to a delimited string*

---

### Description

This is equivalent to [vroom_write()](), but instead of writing to disk, it returns a string. It is primarily useful for examples and for testing.

### Usage

```
vroom_format(
  x,
  delim = "\t",
  eol = "\n",
  na = "NA",
  col_names = TRUE,
  escape = c("double", "backslash", "none"),
  quote = c("needed", "all", "none"),
  bom = FALSE
)
```

### Arguments

| | |
|---|---|
| x | A data frame or tibble to write to disk. |
| delim | Delimiter used to separate values. Defaults to \t to write tab separated value (TSV) files. |
| eol | The end of line character to use. Most commonly either "\n" for Unix style newlines, or "\r\n" for Windows style newlines. |
| na | String used for missing values. Defaults to 'NA'. |
| col_names | If FALSE, column names will not be included at the top of the file. If TRUE, column names will be included. If not specified, col_names will take the opposite value given to append. |
| escape | The type of escape to use when quotes are in the data. <br>• double - quotes are escaped by doubling them. <br>• backslash - quotes are escaped by a preceding backslash. <br>• none - quotes are not escaped. |
| quote | How to handle fields which contain characters that need to be quoted. <br>• needed - Only quote fields which need them. <br>• all - Quote all fields. <br>• none - Never quote fields. |
| bom | If TRUE add a UTF-8 BOM at the beginning of the file. This is recommended when saving data for consumption by excel, as it will force excel to read the data with the correct encoding (UTF-8) |

---

vroom_fwf                        *Read a fixed width file into a tibble*

---

### Description

Read a fixed width file into a tibble

### Usage

```
vroom_fwf(
  file,
  col_positions = fwf_empty(file, skip, n = guess_max),
  col_types = NULL,
  col_select = NULL,
  id = NULL,
  locale = default_locale(),
  na = c("", "NA"),
  comment = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = 100,
  altrep = TRUE,
  altrep_opts = deprecated(),
  num_threads = vroom_threads(),
  progress = vroom_progress(),
  .name_repair = "unique"
)

fwf_empty(file, skip = 0, col_names = NULL, comment = "", n = 100L)

fwf_widths(widths, col_names = NULL)

fwf_positions(start, end = NULL, col_names = NULL)

fwf_cols(...)
```

### Arguments

file              Either a path to a file, a connection, or literal data (either a single string or a raw
                  vector).

                  Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically uncompressed.
                  Files starting with http://, https://, ftp://, or ftps:// will be automatically down-
                  loaded. Remote gz files can also be automatically downloaded and decom-
                  pressed.

                  Literal data is most useful for examples and tests. It must contain at least one
                  new line to be recognised as data (instead of a path) or be a vector of greater
                  than length 1.

                  Using a value of [clipboard()](clipboard()) will read from the system clipboard.

| | |
|---|---|
| col_positions | Column positions, as created by [fwf_empty()](), [fwf_widths()]() or [fwf_positions()](). To read in only selected fields, use [fwf_positions()](). If the width of the last column is variable (a ragged fwf file), supply the last end position as NA. |
| col_types | One of NULL, a [cols()]() specification, or a string. See vignette("readr") for more details. |

If NULL, all column types will be imputed from the first 1000 rows on the input. This is convenient (and fast), but not robust. If the imputation fails, you'll need to supply the correct types yourself.

If a column specification created by [cols()](), it must contain one column specification for each column. If you only want to read a subset of the columns, use [cols_only()]().

Alternatively, you can use a compact string representation where each character represents one column:

- c = character
- i = integer
- n = number
- d = double
- l = logical
- f = factor
- D = date
- T = date time
- t = time
- ? = guess
- _ or - = skip
  By default, reading a file without a column specification will print a message showing what readr guessed they were. To remove this message, use col_types = cols().

| | |
|---|---|
| col_select | One or more selection expressions, like in dplyr::select(). Use c() or list() to use more than one expression. See ?dplyr::select for details on available selection options. |
| id | Either a string or 'NULL'. If a string, the output will contain a variable with that name with the filename(s) as the value. If 'NULL', the default, no variable will be created. |
| locale | The locale controls defaults that vary from place to place. The default locale is US-centric (like R), but you can use [locale()]() to create your own locale that controls things like the default time zone, encoding, decimal mark, big mark, and day/month names. |
| na | Character vector of strings to interpret as missing values. Set this option to character() to indicate no missing values. |
| comment | A string used to identify comments. Any text after the comment characters will be silently ignored. |
| trim_ws | Should leading and trailing whitespace be trimmed from each field before parsing it? |
| skip | Number of lines to skip before reading data. |
| n_max | Maximum number of records to read. |
| guess_max | Maximum number of records to use for guessing column types. |

| | |
|---|---|
| altrep | Control which column types use Altrep representations, either a character vector of types, TRUE or FALSE. See [vroom_altrep()](#) for for full details. |
| altrep_opts | **Deprecated** |
| num_threads | Number of threads to use when reading and materializing vectors. If your data contains newlines within fields the parser will automatically be forced to use a single thread only. |
| progress | Display a progress bar? By default it will only display in an interactive session and not while knitting a document. The display is updated every 50,000 values and will only display if estimated reading time is 5 seconds or more. The automatic progress bar can be disabled by setting option readr.show_progress to FALSE. |
| .name_repair | Handling of column names. By default, vroom ensures column names are not empty and unique. See .name_repair as documented in [tibble::tibble()](#) for additional options including supplying user defined name repair functions. |
| col_names | Either NULL, or a character vector column names. |
| n | Number of lines the tokenizer will read to determine file structure. By default it is set to 100. |
| widths | Width of each field. Use NA as width of last field when reading a ragged fwf file. |
| start, end | Starting and ending (inclusive) positions of each field. Use NA as last end field when reading a ragged fwf file. |
| ... | If the first element is a data frame, then it must have all numeric columns and either one or two rows. The column names are the variable names. The column values are the variable widths if a length one vector, and if length two, variable start and end positions. The elements of . . . are used to construct a data frame with or or two rows as above. |

### Examples

```
fwf_sample <- vroom_example("fwf-sample.txt")
cat(readLines(fwf_sample))

# You can specify column positions in several ways:
# 1. Guess based on position of empty columns
vroom_fwf(fwf_sample, fwf_empty(fwf_sample, col_names = c("first", "last", "state", "ssn")))
# 2. A vector of field widths
vroom_fwf(fwf_sample, fwf_widths(c(20, 10, 12), c("name", "state", "ssn")))
# 3. Paired vectors of start and end positions
vroom_fwf(fwf_sample, fwf_positions(c(1, 30), c(20, 42), c("name", "ssn")))
# 4. Named arguments with start and end positions
vroom_fwf(fwf_sample, fwf_cols(name = c(1, 20), ssn = c(30, 42)))
# 5. Named arguments with column widths
vroom_fwf(fwf_sample, fwf_cols(name = 20, state = 10, ssn = 12))
```

---

| vroom_lines | *Read lines from a file* |
|---|---|

---

**Description**

vroom_lines() is similar to readLines(), however it reads the lines lazily like vroom(), so operations like length(), head(), tail() and sample() can be done much more efficiently without reading all the data into R.

**Usage**

```
vroom_lines(
  file,
  n_max = Inf,
  skip = 0,
  locale = default_locale(),
  altrep = TRUE,
  altrep_opts = deprecated(),
  num_threads = vroom_threads(),
  progress = vroom_progress()
)
```

**Arguments**

| | |
|---|---|
| file | path to a local file. |
| n_max | Maximum number of records to read. |
| skip | Number of lines to skip before reading data. |
| locale | The locale controls defaults that vary from place to place. The default locale is US-centric (like R), but you can use locale() to create your own locale that controls things like the default time zone, encoding, decimal mark, big mark, and day/month names. |
| altrep | Control which column types use Altrep representations, either a character vector of types, TRUE or FALSE. See vroom_altrep() for for full details. |
| altrep_opts | **Deprecated** |
| num_threads | Number of threads to use when reading and materializing vectors. If your data contains newlines within fields the parser will automatically be forced to use a single thread only. |
| progress | Display a progress bar? By default it will only display in an interactive session and not while knitting a document. The display is updated every 50,000 values and will only display if estimated reading time is 5 seconds or more. The automatic progress bar can be disabled by setting option readr.show_progress to FALSE. |

**Examples**

```
lines <- vroom_lines(vroom_example("mtcars.csv"))

length(lines)
head(lines, n = 2)
tail(lines, n = 2)
sample(lines, size = 2)
```

---

vroom_progress                *Determine if progress bars should be shown*

---

### Description

Progress bars are shown *unless* one of the following is TRUE

- The bar is explicitly disabled by setting Sys.getenv("VROOM_SHOW_PROGRESS"="false")
- The code is run in a non-interactive session (interactive() is FALSE).
- The code is run in an RStudio notebook chunk.
- The code is run by knitr / rmarkdown.
- The code is run by testthat (the TESTTHAT envvar is true).

### Usage

```
vroom_progress()
```

### Examples

```
vroom_progress()
```

---

vroom_str                     *Structure of objects*

---

### Description

Similar to str() but with more information for Altrep objects.

### Usage

```
vroom_str(x)
```

### Arguments

x                 a vector

### Examples

```
# when used on non-altrep objects altrep will always be false
vroom_str(mtcars)

mt <- vroom(vroom_example("mtcars.csv"), ",", altrep = c("chr", "dbl"))
vroom_str(mt)
```

---

vroom_write          *Write a data frame to a delimited file*

---

### Description

Write a data frame to a delimited file

### Usage

```
vroom_write(
  x,
  path,
  delim = "\t",
  eol = "\n",
  na = "NA",
  col_names = !append,
  append = FALSE,
  quote = c("needed", "all", "none"),
  escape = c("double", "backslash", "none"),
  bom = FALSE,
  num_threads = vroom_threads(),
  progress = vroom_progress()
)
```

### Arguments

| | |
|---|---|
| x | A data frame or tibble to write to disk. |
| path | **Deprecated** |
| delim | Delimiter used to separate values. Defaults to \t to write tab separated value (TSV) files. |
| eol | The end of line character to use. Most commonly either "\n" for Unix style newlines, or "\r\n" for Windows style newlines. |
| na | String used for missing values. Defaults to 'NA'. |
| col_names | If FALSE, column names will not be included at the top of the file. If TRUE, column names will be included. If not specified, col_names will take the opposite value given to append. |
| append | If FALSE, will overwrite existing file. If TRUE, will append to existing file. In both cases, if the file does not exist a new file is created. |
| quote | How to handle fields which contain characters that need to be quoted. |

- needed - Only quote fields which need them.
- all - Quote all fields.
- none - Never quote fields.

| | |
|---|---|
| escape | The type of escape to use when quotes are in the data. |

- double - quotes are escaped by doubling them.
- backslash - quotes are escaped by a preceding backslash.
- none - quotes are not escaped.

| | |
|---|---|
| bom | If `TRUE` add a UTF-8 BOM at the beginning of the file. This is recommended when saving data for consumption by excel, as it will force excel to read the data with the correct encoding (UTF-8) |
| num_threads | Number of threads to use when reading and materializing vectors. If your data contains newlines within fields the parser will automatically be forced to use a single thread only. |
| progress | Display a progress bar? By default it will only display in an interactive session and not while knitting a document. The display is updated every 50,000 values and will only display if estimated reading time is 5 seconds or more. The automatic progress bar can be disabled by setting option `readr.show_progress` to `FALSE`. |

## Examples

```
# If you only specify a file name, vroom_write() will write
# the file to your current working directory.
out_file <- tempfile(fileext = "csv")
vroom_write(mtcars, out_file, ",")

# You can also use a literal filename
# vroom_write(mtcars, "mtcars.tsv")

# If you add an extension to the file name, write_()* will
# automatically compress the output.
# vroom_write(mtcars, "mtcars.tsv.gz")
# vroom_write(mtcars, "mtcars.tsv.bz2")
# vroom_write(mtcars, "mtcars.tsv.xz")
```

---

vroom_write_lines        *Write lines to a file*

---

## Description

Write lines to a file

## Usage

```
vroom_write_lines(x, path, eol = "\n", na = "NA", append = FALSE)
```

## Arguments

| | |
|---|---|
| x | A data frame or tibble to write to disk. |
| path | **Deprecated** |
| eol | The end of line character to use. Most commonly either `"\n"` for Unix style newlines, or `"\r\n"` for Windows style newlines. |
| na | String used for missing values. Defaults to 'NA'. |
| append | If `FALSE`, will overwrite existing file. If `TRUE`, will append to existing file. In both cases, if the file does not exist a new file is created. |

# Index