

Package ‘dtlog’

September 15, 2026

Type Package

Title Logging for 'data.table' Operations

Version 0.1.0

Description Provides feedback about 'data.table' operations. 'dtlog' redefines the subsetting method for data tables as well as several functions exported by 'data.table' so that each operation prints a short message describing what it did: how many rows were removed, which columns were added, updated or dropped, how many groups an aggregation produced, and so on. The operations themselves are left untouched, including modification by reference. It also provides dttable(), which describes the variables a single data table holds and passes every other call on to base::table() unchanged. Inspired by the 'tidylog' package.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 3.5.0)

Imports data.table (>= 1.14.0), stats, utils

Suggests testthat (>= 3.0.0), dplyr, tidyr, tidylog, tibble, knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr, rmarkdown

URL <https://github.com/AkiShiroshita/dtlog>,
<https://akishiroshita.github.io/dtlog/>

BugReports <https://github.com/AkiShiroshita/dtlog/issues>

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Akihiro Shiroshita [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0003-0262-459X>>)

Maintainer Akihiro Shiroshita <akihirokun8@gmail.com>

Repository CRAN

Date/Publication 2026-09-15 10:50:27 UTC

Contents

as.data.table 2

dtlog_pause 3

dtlog_summary 3

dttable 4

dt_log 5

fread 6

fwrite 7

head_tail 7

merge.data.table 8

reshape 9

rows 9

set_functions 10

[.data.table 11

Index 13

as.data.table	Convert an object to a data table, with a log
---------------	---

Description

Convert an object to a data table, with a log

Usage

```
as.data.table(x, ...)
```

Arguments

- x The object to convert.
- ... All other arguments of `data.table::as.data.table()`.

Value

The `data.table` that `data.table::as.data.table()` returns.

Examples

```
data.table::as.data.table(head(mtcars, 3), keep.rownames = "car")
```

dtlog_pause	<i>Pause and resume logging</i>
-------------	---------------------------------

Description

dtlog_pause() turns off all dtlog messages without detaching the package, dtlog_resume() turns them back on. This is useful for a block of code that would otherwise produce a lot of output.

Usage

```
dtlog_pause()
```

```
dtlog_resume()
```

Value

Invisibly the logging state *before* the call: TRUE if logging was active, FALSE if it was paused. Both functions report the state they found rather than the one they left behind, so dtlog_pause() returns TRUE when it is the call that actually paused logging, and dtlog_resume() returns FALSE when it is the call that actually resumed it.

Examples

```
dtlog_pause()
dtlog_resume()
```

dtlog_summary	<i>Log a summary of a data table</i>
---------------	--------------------------------------

Description

Prints the number of rows and columns of a data table, along with its key, and returns the object unchanged, so that it can be used within a chain of operations.

Usage

```
dtlog_summary(.data)
```

Arguments

.data A data.table (or any data frame).

Value

.data, unchanged and returned visibly.

See Also

[dt_log\(\)](#) to write a transcript of a whole session to a file.

Examples

```
dt <- data.table::data.table(a = 1:3, b = 4:6)
dtlog_summary(dt)
```

dttable	<i>Describe the variables of a data table</i>
---------	---

Description

`dttable()` describes a `data.table` rather than cross tabulating it. Given a single `data.table` it reports one row per column – the name, the number of unique values, and the values themselves – and returns that description as a `data.table` with the columns `Variable`, `N_unique` and `Unique_value`.

Usage

```
dttable(...)
```

Arguments

... The vectors to tabulate, as in [base::table\(\)](#), or a single `data.table` to describe.

Details

`dttable()` is a function of its own: it does not mask [base::table\(\)](#), and loading `dtlog` leaves `table()` exactly as it was. Every call that is not a single `data.table` is handed to [base::table\(\)](#) unchanged, so `dttable(dt$sex, dt$death)`, `dttable(x, useNA = "ifany")` and `dttable(as.data.frame(dt))` return what [base::table\(\)](#) returns. Describing a single `data.table` is the only thing `dttable()` adds.

A column with 20 or more unique values is reported as *possibly continuous* rather than listed; the option `dtlog.table_max_unique` moves that point, and `Inf` lists every column however many values it holds. A list column is reported as such, and a list of values longer than 80 characters is truncated.

The values are listed in the order the column sorts in: numbers ascending, characters alphabetically, dates and times chronologically, factors and ordered factors by their levels, `FALSE` before `TRUE`. Each value is written the way its own class writes it, so an `ITime` is listed as `09:00:00` rather than as the seconds it is stored as. A type that cannot be sorted keeps the order its values appear in.

Missing values are listed and counted like any other value: `NA` (including `NA` as a level of a factor) appears as `Missing`, `NaN` as `NaN`, and both are counted in `N_unique`. They sort last, so a column that has any ends with `Missing`. An empty string is a value of its own, not a missing one.

The description goes through the same output as every other `dtlog` message, so it obeys `dtlog.display`, is silenced by [dtlog_pause\(\)](#), and is written to the transcript opened by [dt_log\(\)](#).

Value

For a single `data.table`, a `data.table` with the columns `Variable`, `N_unique` and `Unique_value`, returned invisibly. For anything else, whatever `base::table()` returns.

See Also

`dtlog_summary()` for the size and key of a table alone.

Examples

```
dttable(data.table::data.table(a = 1:3, b = c("x", "y", "x")))
dttable(c("a", "b", "a"))
```

dt_log

Write the code and its log to a text file

Description

`dt_log()` starts a transcript: from that point on, every operation that `dtlog` reports is appended to a text file, together with the call that produced it. `dt_log_end()` closes the transcript. Start and end are up to you; nothing is written before the first call or after the second.

Usage

```
dt_log(file, append = FALSE, code = TRUE, echo = TRUE)
```

```
dt_log_end()
```

```
dt_log_file()
```

Arguments

<code>file</code>	Path of the text file. There is no default: name a path yourself, so that nothing is ever written to a place you did not choose. <code>NULL</code> ends the current transcript, so <code>dt_log(NULL)</code> is the same as <code>dt_log_end()</code> .
<code>append</code>	Append to an existing file instead of overwriting it.
<code>code</code>	Write the call above its log. Set to <code>FALSE</code> for the messages alone.
<code>echo</code>	Keep printing to the console as well. <code>FALSE</code> writes only to the file.

Details

Each operation is appended with a plain `cat()` that opens and closes the file again, so the transcript stays readable while a long script is running and survives a session that ends without `dt_log_end()` (only the closing line is then missing).

Value

The path of the transcript, invisibly.

Examples

```
path <- tempfile(fileext = ".txt")
dt_log(path, echo = FALSE)
dt <- data.table::as.data.table(mtcars)
dt[mpg > 20]
dt[, kpl := mpg * 0.425]
dt_log_end()
cat(readLines(path), sep = "\n")
```

fread	<i>Read a file into a data table, with a log</i>
-------	--

Description

Read a file into a data table, with a log

Usage

```
fread(...)
```

Arguments

... All arguments of `data.table::fread()`.

Value

The data table that `data.table::fread()` returns.

Examples

```
fread(text = "a,b\n1,2\n3,4")
```

fwrite	<i>Write a data table to a file, with a log</i>
--------	---

Description

Write a data table to a file, with a log

Usage

```
fwrite(x, ...)
```

Arguments

x	The table to write.
...	All other arguments of <code>data.table::fwrite()</code> .

Value

NULL, invisibly, as `data.table::fwrite()` returns it.

Examples

```
dt <- data.table::data.table(a = 1:2, b = 3:4)
fwrite(dt, tempfile())
```

head_tail	<i>First or last rows of a data table, with a log</i>
-----------	---

Description

First or last rows of a data table, with a log

Usage

```
## S3 method for class 'data.table'
head(x, ...)

## S3 method for class 'data.table'
tail(x, ...)
```

Arguments

x	The data table.
...	All other arguments of <code>utils::head()</code> and <code>utils::tail()</code> , i.e. n.

Value

The same rows `data.table` would return.

Examples

```
head(data.table::as.data.table(mtcars), 3)
```

merge.data.table	<i>Merge two data tables, with a log</i>
------------------	--

Description

Reports the columns the merge added and how the rows of the two inputs were matched, in the style of tidylog's join messages. The counts of unmatched rows are only computed when `options(dtlog.detail = "full")` (the default); they cost two additional matching passes over the inputs.

Usage

```
## S3 method for class 'data.table'
merge(x, y, ...)
```

Arguments

<code>x, y</code>	The data tables to merge.
<code>...</code>	All other arguments of <code>data.table::merge.data.table()</code> .

Value

The merged data table, exactly as `data.table::merge.data.table()` returns it.

Examples

```
a <- data.table::data.table(id = 1:3, v = 1:3)
b <- data.table::data.table(id = 2:4, w = 4:6)
merge(a, b, by = "id")
```

reshape	<i>Reshape a data table, with a log</i>
---------	---

Description

Reports which columns were reorganized into which, and how the dimensions of the table changed, in the style of tidylog's `pivot_longer()` and `pivot_wider()` messages.

Usage

```
melt(data, ...)

## S3 method for class 'data.table'
melt(data, ...)

dcast(data, ...)

## S3 method for class 'data.table'
dcast(data, ...)
```

Arguments

<code>data</code>	The table to reshape.
<code>...</code>	All other arguments of <code>data.table::melt.data.table()</code> and <code>data.table::dcast.data.table()</code> .

Value

The reshaped data table.

Examples

```
dt <- data.table::data.table(id = 1:2, a = 3:4, b = 5:6)
long <- data.table::melt(dt, id.vars = "id")
data.table::dcast(long, id ~ variable)
```

rows	<i>Row operations with a log</i>
------	----------------------------------

Description

These functions behave exactly like their `data.table` counterparts and report how many rows they removed, kept or combined.

Usage

```
## S3 method for class 'data.table'
unique(x, ...)

## S3 method for class 'data.table'
duplicated(x, ...)

## S3 method for class 'data.table'
na.omit(object, ...)

rbindlist(l, ...)

funion(x, y, ...)

fintersect(x, y, ...)

fsetdiff(x, y, ...)
```

Arguments

`x, y, object, l` The inputs, as in the corresponding `data.table` function.
`...` All other arguments, passed on unchanged.

Value

Whatever the `data.table` function returns.

Examples

```
dt <- data.table::data.table(a = c(1, 1, 2), b = c(NA, 2, 3))
unique(dt, by = "a")
stats::na.omit(dt)
```

set_functions

Modify a data table by reference, with a log

Description

These functions are the `data.table` `set*`() functions. They still change their input by reference and return exactly what `data.table` returns; they only report what they changed.

Usage

```
setnames(x, ...)

setcolorder(x, ...)
```

```

setkey(x, ...)
setkeyv(x, ...)
setorder(x, ...)
setorderv(x, ...)
setindex(x, ...)
setindexv(x, ...)
set(x, ...)
setDT(x, ...)
setDF(x, ...)
setattr(x, ...)

```

Arguments

<code>x</code>	The data table (or, for <code>setDT()</code> , the object to convert).
<code>...</code>	All other arguments, passed on unchanged.

Value

Whatever the corresponding `data.table` function returns.

Examples

```

dt <- data.table::data.table(a = 3:1, b = 1:3)
data.table::setnames(dt, "a", "alpha")
data.table::setkey(dt, alpha)

```

[.data.table	<i>Subset, aggregate and update a data.table, with a log</i>
--------------	--

Description

`dtlog` redefines the `[` method for data tables. The call is passed on to `data.table` unchanged – same arguments, same evaluation environment, same return value, same modification by reference – and a message describing what happened is printed afterwards.

Usage

```

## S3 method for class 'data.table'
x[...]
```

Arguments

<code>x</code>	A <code>data.table</code> .
<code>...</code>	All other arguments of <code>[.data.table]</code> , i.e. <code>i</code> , <code>j</code> , <code>by</code> , <code>keyby</code> , <code>with</code> , <code>nomatch</code> , <code>mult</code> , <code>roll</code> , <code>rollends</code> , <code>which</code> , <code>.SDcols</code> , <code>verbose</code> , <code>allow.cartesian</code> , <code>drop</code> , <code>on</code> , <code>env</code> and <code>showProgress</code> . They are never touched by <code>dtlog</code> .

Details

Depending on the call, the message uses the vocabulary of tidylog: `filter` (rows removed by `i`), `arrange` (rows reordered), `join` (`i` is a table or `on=` was given), `select` (`j` only picks existing columns), `mutate` (`:=`), `group_by/summarize` (`by=`/`keyby=`).

Value

Whatever `data.table`'s `[` returns, with the same visibility.

Examples

```
dt <- data.table::as.data.table(mtcars)
dt[mpg > 20]
dt[, mpg_per_cyl := mpg / cyl]
```

Index

[.data.table, 11

as.data.table, 2

base::table(), 4, 5

cat(), 5

data.table::as.data.table(), 2

data.table::dcast.data.table(), 9

data.table::fread(), 6

data.table::fwrite(), 7

data.table::melt.data.table(), 9

data.table::merge.data.table(), 8

dcast (reshape), 9

dt_log, 5

dt_log(), 4

dt_log_end (dt_log), 5

dt_log_file (dt_log), 5

dtlog_pause, 3

dtlog_pause(), 4

dtlog_resume (dtlog_pause), 3

dtlog_summary, 3

dtlog_summary(), 5

dttable, 4

duplicated.data.table (rows), 9

fintersect (rows), 9

fread, 6

fsetdiff (rows), 9

funion (rows), 9

fwrite, 7

head.data.table (head_tail), 7

head_tail, 7

melt (reshape), 9

merge.data.table, 8

na.omit.data.table (rows), 9

rbindlist (rows), 9

reshape, 9

rows, 9

set (set_functions), 10

set_functions, 10

setattr (set_functions), 10

setcolorder (set_functions), 10

setDF (set_functions), 10

setDT (set_functions), 10

setDT(), 11

setindex (set_functions), 10

setindexv (set_functions), 10

setkey (set_functions), 10

setkeyv (set_functions), 10

setnames (set_functions), 10

setorder (set_functions), 10

setorderv (set_functions), 10

tail.data.table (head_tail), 7

unique.data.table (rows), 9

utils::head(), 7

utils::tail(), 7