Package 'dtlog'

September 25, 2026

Type Package

Title Logging for 'data.table' Operations

Version 0.2.0

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 (>= 1.16.0), 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 2026-09-25 18:40:10 UTC

Contents

`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")
```

copy *Copy a data table, with a log*

Description

`data.table` modifies by reference, and `data.table::copy()` is the way out of that: it is the point where a second, independent table comes into existence. dtlog says so, and says how big the copy was.

Usage

```
copy(x)
```

Arguments

`x`	The object to copy.

Value

The deep copy that `data.table::copy()` returns.

Examples

```
dt <- data.table::data.table(a = 1:3)
safe <- data.table::copy(dt)
```

dtlog_pause *Pause and resume logging*

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` *Log a summary of a data table*

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` *Describe the variables of a data table*

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(dtsex, dtdeath), 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

file	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. NULL ends the current transcript, so dt_log(NULL) is the same as dt_log_end().
append	Append to an existing file instead of overwriting it.
code	Write the call above its log. Set to FALSE for the messages alone.
echo	Keep printing to the console as well. FALSE 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")
```

---

foverlaps *Join two data tables on overlapping intervals, with a log*

Description

Reports how many rows went in and came out, and which columns the join added. foverlaps() matches a row of x against every row of y whose interval overlaps it, so the result can be longer as well as shorter than x, and the row count is the part worth seeing.

Usage

```
foverlaps(x, y, ...)
```

Arguments

x, y	The data tables to join. Both carry the interval as two columns, and y has to be keyed on them.
...	All other arguments of data.table::foverlaps().

Value

Whatever data.table::foverlaps() returns: the matched rows, or, with which = TRUE, the row numbers that matched.

Examples

```
x <- data.table::data.table(s = c(1, 5), e = c(3, 7))
y <- data.table::data.table(s = c(2, 6), e = c(4, 8))
data.table::setkey(y, s, e)
data.table::foverlaps(x, y)
```

---

fread *Read a file into a data table, with a log*

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 — *Write a data table to a file, with a log*

Description

Write a data table to a file, with a log

Usage

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

Arguments

x	The table to write.
...	All other arguments of `data.table::fwrite()`.

Value

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

Examples

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

grouping_sets — *Aggregate over several grouping sets, with a log*

Description

`data.table::rollup()`, `data.table::cube()` and `data.table::groupingsets()` run one aggregation per grouping set and stack the results. Each reports the size of the table it produced next to the size of the table it started from, in the wording an aggregating `j` uses.

Usage

```
rollup(x, ...)

cube(x, ...)

groupingsets(x, ...)
```

Arguments

x	The data table to aggregate.
...	All other arguments, as in the corresponding `data.table` function.

Value

Whatever the `data.table` function returns.

Examples

```
dt <- data.table::data.table(g = c("a", "a", "b"), h = c("x", "y", "x"),
                             v = 1:3)
data.table::rollup(dt, j = sum(v), by = c("g", "h"))
data.table::cube(dt, j = sum(v), by = c("g", "h"))
```

`head_tail` *First or last rows of a data table, with a log*

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 `utils::head()` and `utils::tail()`, i.e. `n`.

Value

The same rows `data.table` would return.

Examples

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

merge.data.table *Merge two data tables, with a log*

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

`x, y`	The data tables to merge.
`...`	All other arguments of `data.table::merge.data.table()`.

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 *Reshape a data table, with a log*

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

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

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 *Row operations with a log*

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, ...)

fsetequal(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, ...)

setnafill(x, ...)

setdroplevels(x, ...)
```

Arguments

x	The data table (or, for setDT(), the object to convert).
...	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)
```

---

split.data.table *Split a data table, with a log*

Description

Reports how many tables the rows were split into, and how many rows each of them holds.

Usage

```
## S3 method for class 'data.table'
split(x, f, drop = FALSE, ...)
```

Arguments

x	The data table to split.
f	The grouping factor, as in base::split(). data.table's own by= is the usual way to give it instead.
drop	Whether to drop unused levels, as in base::split().
...	All other arguments of data.table's split() method, i.e. by, sorted, keep.by and flatten.

Value

The list of data tables that data.table returns.

Examples

```
dt <- data.table::data.table(g = c("a", "a", "b"), v = 1:3)
split(dt, by = "g")
```

`[.data.table` *Subset, aggregate and update a data.table, with a log*

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

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

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