Package 'lumberjack'

July 20, 2018

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

License GPL-3

Title Track Changes in Data

LazyData no

Type Package

LazyLoad yes

Description A function composition ('pipe') operator and extensible framework that allows for easy logging of changes in data.

Version 0.3.0

URL https://github.com/markvanderloo/lumberjack

BugReports https://github.com/markvanderloo/lumberjack/issues

Imports utils, R6

Depends R (>= 3.4.0)

Suggests testthat, knitr, rmarkdown

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Mark van der Loo [aut, cre]

Repository CRAN

Date/Publication 2018-07-20 08:20:02 UTC

R topics documented:

`cellwise` *The cellwise logger.*

Description

The cellwise logger registres the row, column, old, and new value of cells that changed, along with a step number, timestamp, and the expression used to alter a dataset.

Usage

```
cellwise
```

Format

An `R6` class object.

Creating a logger

```
cellwise$new(verbose=TRUE, file=tempfile()
```

`verbose`	toggle verbosity
`key`	`[character|integer]` index to column that uniquely identifies a row.
`verbose`	`[logical]` toggle verbosity.
`file`	[character] filename for temporaty log storage.

Dump options

```
$dump(key, verbose=TRUE, file="cellwise.csv")
```

`file`	`[character]` location to write final output to.

Getting data from the logger

`$logdata()` Returns a data.frame (it dumps, then reads the current log).

Details

At initialization, the cellwise logger opens a connection to a temporary file. All logging info is written to that connection. When `dump_log` is called, the temporary file is closed, copied to the output file, and reopened for writing. The connection is closed automatically when the logger is

destroyed, for example when calling `dump_log(stop=TRUE)` (the default), or `stop_log()` in the lumberjack pipeline.

See Also

Other loggers: `expression_logger`, `filedump`, `simple`

Examples

```
logfile <- tempfile(fileext=".csv")

# convert height from inch to cm and log changes.
# we need to set a unique key.
women$sleutel <- 1:nrow(women)
out <- women %L>%
  start_log(log=cellwise$new(key="sleutel")) %L>%
  {.$height <- .$height*2.54; .} %L>%
  dump_log(file=logfile, stop=TRUE)

read.csv(logfile) %L>% head()

# work with an externally defined logger.
iris$id <- seq_len(nrow(iris))
logger <- cellwise$new(key="id")
iris %L>%
  start_log(logger) %L>%
  head() %L>%
  stop_log()
logger$logdata()
```

---

dump_log *Dump a log*

Description

Dump a log

Usage

```
dump_log(data, stop = TRUE, ...)
```

Arguments

`data`	data set containing a log
`stop`	stop logging after the dump?
`...`	Arguments past to the dump method of the logger.

Value

The data, invisibly

`expression_logger` *The expression logger.*

Description

The expression logger records the result of one or more user-defined expressions. It can be used, for example to track aggregates (mean, min, max) of variables as they get processed in the data pipeline.

Usage

```
expression_logger
```

Format

An R6 class object.

Creating a logger

```
expression_logger$new(..., file="expression_log.csv")
```

`...`	comma-separated `name = expression` pairs
`file`	[character] filename for temporaty log storage.

Dump options

```
$dump()
```

Currently no options are implemented.

See Also

Other loggers: `cellwise`, `filedump`, `simple`

Examples

```
logfile <- file.path(tempfile(fileext=".csv"))
e <- expression_logger$new(mean=mean(height), sd=sd(height),file=logfile)

out <- women %L>%
  start_log(e) %L>%
  within(height <- height * 2) %L>%
  within(height <- height * 3) %L>%
```

```
  dump_log(stop=TRUE)

read.csv(logfile)
```

`filedump` *The file dumping logger.*

Description

The file dumping logger dumps the most recent version of a dataset to csv in a directory of choice.

Usage

```
filedump
```

Format

An `R6` class object.

Creating a logger

```
filedump$new(dir=file.path(tempdir(),"timber"), prefix="step%03d.csv", verbose=TRUE)
```

`dir`	Where to write the file dumps.
`filename`	filename template, used with `sprintf` to create a file name.
`verbose`	toggle verbosity

Dump options

```
$dump(...)
```

`...`	Currently unused.

Retrieve log data

`$logdata()` returns a list of data frames, sorted in the order returned by `base::dir()`

Details

If `dir` does not exist it is created.

See Also

Other loggers: `cellwise`, `expression_logger`, `simple`

get_log *Get log object from a data item*

Description

Get log object from a data item

Usage

```
get_log(data)
```

Arguments

data	An R object carrying data.

Value

A logging object, or NULL if none exists.

lumberjack *The pipe operator that logs*

Description

The pipe operator that logs

Overview

The lumberjack %L>% behaves much like other function composition ('pipe') operators available in R (e.g. magrittr , yapo, pipeR) with one exception: it allows for logging the changes made to the data by the functions acting on it.

The actual logging mechanism is completely flexible and extensible. This package comes with a few predefined loggers, but users and package authors can write their own logger that follows the lumberjack API.

See the Introductory vignette to start logging or the extending lumberjack manual to start writing your own loggers.

Happy logging!

simple *The simple logger.*

Description

The simple logger registers the expression applied to an object; a POSIXct timestamp and a logical indicating whether the input is identical to the output.

Usage

```
simple
```

Format

An R6 class object.

Creating a logger

```
simple$new(verbose=TRUE)
```

verbose	toggle verbosity

Dump options

```
$dump(file="simple_log.csv",...)
```

file	filename or connection to write output to.
...	extra options passed to write.csv, except row.names, which is set to FALSE.

Get data

`$logdata()` Returns a data.frame.

See Also

Other loggers: cellwise, expression_logger, filedump

Examples

```
logfile <- tempfile(fileext=".csv")
out <- women %L>%
  start_log(log=simple$new(verbose=FALSE)) %L>%
  identity() %L>%
  head() %L>%
  dump_log(file=logfile, stop=TRUE)
```

```
read.csv(logfile,stringsAsFactors=FALSE)
```

`start_log` *Start logging changes on a dataset.*

Description

Start logging changes on a dataset.

Usage

```
start_log(data, log = simple$new())
```

Arguments

`data`	An R object carrying data.
`log`	A logging object (typically an environment wrapped in an S3 class)

`stop_log` *Stop logging*

Description

Calls the logger's `$stop()` method if it exists, and removes the logger as attribute from `data`.

Usage

```
stop_log(data, ...)
```

Arguments

`data`	An R object carrying data
`...`	Passed to the logger's `stop` method, if it exists.

`%>>%` *The lumberjack operator*

Description

A not-a-pipe operator that logs

Usage

```
lhs %>>% rhs

lhs %L>% rhs
```

Arguments

`lhs`	Input value
`rhs`	Function call or 'dotted' expression (see below). as value

Piping

The operators `%L>%` and `%>>%` are synonyms. The `%L>%` is the default since version 0.3.0 to avoid confusion with the `%>>%` operator of the `pipeR` package but `%>>%` still works.

The lumberjack operator behaves as a simplified version of the `magrittr` pipe operator. The basic behavior of `lhs %>>% rhs` is the following:

- If the `rhs` uses dot-variables (`.`), these are interpreted as the left-hand side, except in formulas where dots already have a special meaning.
- If the `rhs` is a function call, with no dot-variables used, the `lhs` is used as its first argument.

The most notable differences with 'magrittr' are the following.

- it does not allow you to define functions in the magrittr style, like `a <- . %>% sin(.)`
- there is no assignment-pipe like `%<>%`.
- you cannot do things like `x %>% sin` (without the brackets).

Logging

If the left-hand-side is tagged for logging, the lumberjack will update the log by calling the logger's `$add()` method, with arguments `meta`, `input`, `output`. Here, `meta` is a list with information on the operations performed, and input and output are the left-hand-side and the result, respectively (See also: extending lumberjack).

Examples

```
# pass arguments to a function
1:3 %L>% mean()

# pass arguments using "."
TRUE %L>% mean(c(1,NA,3), na.rm = .)

# pass arguments to an expression, using "."
1:3 %L>% { 3 * .}

# in a more complicated expression, return "." explicitly
women %L>% { .$height <- 2*.$height; . }
```

Index