Package 'logger'

October 19, 2021

Type Package

Title A Lightweight, Modern and Flexible Logging Utility

Description Inspired by the the 'futile.logger' R package and 'logging' Python module, this utility provides a flexible and extensible way of formatting and delivering log messages with low overhead.

Version 0.2.2

Date 2021-10-10

URL https://daroczig.github.io/logger/

BugReports https://github.com/daroczig/logger/issues

Encoding UTF-8

RoxygenNote 7.1.1

License AGPL-3

Imports utils

Suggests glue, pander, jsonlite, crayon, slackr (>= 1.4.1), RPushbullet, telegram, testthat, covr, knitr, rmarkdown, devtools, roxygen2, parallel, rsyslog, shiny, callr, txtq, botor, R.utils, syslognet

Enhances logging, futile.logger, log4r

VignetteBuilder knitr

NeedsCompilation no

Author Gergely Daróczi [aut, cre] (<https://orcid.org/0000-0003-3149-8537>), System1 [fnd]

Maintainer Gergely Daróczi <daroczig@rapporter.net>

Repository CRAN

Date/Publication 2021-10-19 05:30:08 UTC

R topics documented:

`appender_async` *Delays executing the actual appender function to the future in a background process to avoid blocking the main R session*

Description

Delays executing the actual appender function to the future in a background process to avoid blocking the main R session

Usage

```
appender_async(
  appender,
  batch = 1,
  namespace = "async_logger",
  init = function() log_info("Background process started")
)
```

Arguments

`appender`	a `log_appender` function with a `generator` attribute (TODO note not required, all fn will be passed if not)
`batch`	number of records to process from the queue at once
`namespace`	`logger` namespace to use for logging messages on starting up the background process
`init`	optional function to run in the background process that is useful to set up the environment required for logging, eg if the `appender` function requires some extra packages to be loaded or some environment variables to be set etc

Value

function taking `lines` argument

Note

This functionality depends on the **txtq** and **callr** packages. The R session's temp folder is used for staging files (message queue and other forms of communication between the parent and child processes).

See Also

This function is to be used with an actual `log_appender`, for example `appender_console`, `appender_file`, `appender_tee`, `appender_pushbullet`, `appender_telegram`, `appender_syslog` or `appender_kinesis`.

Examples

```
## Not run:
appender_file_slow <- function(file) {
  force(file)
  function(lines) {
    Sys.sleep(1)
    cat(lines, sep = '\n', file = file, append = TRUE)
  }
}

## log what's happening in the background
log_threshold(TRACE, namespace = 'async_logger')
log_appender(appender_console, namespace = 'async_logger')

## start async appender
t <- tempfile()
log_info('Logging in the background to {t}')
my_appender <- appender_async(appender_file_slow(file = t))

## use async appender
log_appender(my_appender)
log_info('Was this slow?')
system.time(for (i in 1:25) log_info(i))

readLines(t)
Sys.sleep(10)
readLines(t)

## check on the async appender (debugging, you will probably never need this)
attr(my_appender, 'async_writer_queue')$count()
attr(my_appender, 'async_writer_queue')$log()

attr(my_appender, 'async_writer_process')$get_pid()
attr(my_appender, 'async_writer_process')$get_state()
attr(my_appender, 'async_writer_process')$poll_process(1)
attr(my_appender, 'async_writer_process')$read()

attr(my_appender, 'async_writer_process')$is_alive()
attr(my_appender, 'async_writer_process')$read_error()

## End(Not run)
```

---

appender_console *Append log record to stderr*

Description

Append log record to stderr

Usage

```
appender_console(lines)

appender_stderr(lines)
```

Arguments

lines	character vector

See Also

This is a log_appender, for alternatives, see eg appender_stdout, appender_file, appender_tee, appender_slack, appender_pushbullet, appender_telegram, appender_syslog, appender_kinesis and appender_async for evaluate any log_appender function in a background process.

---

appender_file *Append log messages to a file*

Description

Log messages are written to a file with basic log rotation: when max number of lines or bytes is defined to be other than Inf, then the log file is renamed with a .1 suffix and a new log file is created. The renaming happens recursively (eg logfile.1 renamed to logfile.2) until the specified max_files, then the oldest file (logfile.{max_files-1}) is deleted.

Usage

```
appender_file(
  file,
  append = TRUE,
  max_lines = Inf,
  max_bytes = Inf,
  max_files = 1L
)
```

Arguments

file	path
append	boolean passed to cat defining if the file should be overwritten with the most recent log message instead of appending
max_lines	numeric specifying the maximum number of lines allowed in a file before rotating
max_bytes	numeric specifying the maximum number of bytes allowed in a file before rotating
max_files	integer specifying the maximum number of files to be used in rotation

Value

function taking `lines` argument

See Also

This is generator function for `log_appender`, for alternatives, see eg `appender_console`, `appender_tee`, `appender_slack`, `appender_pushbullet`, `appender_telegram`, `appender_syslog`, `appender_kinesis` and `appender_async` for evaluate any `log_appender` function in a background process.

Examples

```
## Not run:
## ########################################################################
## simple example logging to a file
t <- tempfile()
log_appender(appender_file(t))
for (i in 1:25) log_info(i)
readLines(t)

## ########################################################################
## more complex example of logging to file
## rotated after every 3rd line up to max 5 files

## create a folder storing the log files
t <- tempfile(); dir.create(t)
f <- file.path(t, 'log')

## define the file logger with log rotation enabled
log_appender(appender_file(f, max_lines = 3, max_files = 5L))

## log 25 messages
for (i in 1:25) log_info(i)

## see what was logged
lapply(list.files(t, full.names = TRUE), function(t) {
  cat('\n##', t, '\n')
  cat(readLines(t), sep = '\n')
})

## enable internal logging to see what's actually happening in the logrotate steps
log_threshold(TRACE, namespace = '.logger')
## run the above commands again

## End(Not run)
```

`appender_kinesis` *Send log messages to a Amazon Kinesis stream*

Description

Send log messages to a Amazon Kinesis stream

Usage

```
appender_kinesis(stream)
```

Arguments

`stream`	name of the Kinesis stream

Value

function taking `lines` and optional `partition_key` argument

Note

This functionality depends on the **botor** package.

See Also

This is generator function for `log_appender`, for alternatives, see eg `appender_console`, `appender_file`, `appender_tee`, `appender_pushbullet`, `appender_telegram`, `appender_syslog` and `appender_async` for evaluate any `log_appender` function in a background process.

`appender_pushbullet` *Send log messages to Pushbullet*

Description

Send log messages to Pushbullet

Usage

```
appender_pushbullet(...)
```

Arguments

`...`	parameters passed to `pbPost`, such as `recipients` or `apikey`, although it's probably much better to set all these in the `~/.rpushbullet.json` as per package docs at `http://dirk.eddelbuettel.com/code/rpushbullet.html`

Note

This functionality depends on the **RPushbullet** package.

See Also

This is generator function for log_appender, for alternatives, see eg appender_console, appender_file, appender_tee, appender_slack, appender_telegram, appender_syslog, appender_kinesis and appender_async for evaluate any log_appender function in a background process.

---

appender_slack *Send log messages to a Slack channel*

Description

Send log messages to a Slack channel

Usage

```
appender_slack(
  channel = Sys.getenv("SLACK_CHANNEL"),
  username = Sys.getenv("SLACK_USERNAME"),
  icon_emoji = Sys.getenv("SLACK_ICON_EMOJI"),
  api_token = Sys.getenv("SLACK_API_TOKEN"),
  preformatted = TRUE
)
```

Arguments

channel	Slack channel name with a hashtag prefix for public channel and no prefix for private channels
username	Slack (bot) username
icon_emoji	optional override for the bot icon
api_token	Slack API token
preformatted	use code tags around the message?

Value

function taking lines argument

Note

This functionality depends on the **slackr** package.

See Also

This is generator function for log_appender, for alternatives, see eg appender_console, appender_file, appender_tee, appender_pushbullet, appender_telegram, appender_syslog, appender_kinesis and appender_async for evaluate any log_appender function in a background process.

appender_stdout *Append log record to stdout*

Description

Append log record to stdout

Usage

```
appender_stdout(lines)
```

Arguments

lines	character vector

See Also

This is a log_appender, for alternatives, see eg appender_console, appender_file, appender_tee, appender_slack, appender_pushbullet

appender_syslog *Send log messages to the POSIX system log*

Description

Send log messages to the POSIX system log

Usage

```
appender_syslog(identifier, ...)
```

Arguments

identifier	A string identifying the process.
...	Further arguments passed on to open_syslog.

Value

function taking lines argument

Note

This functionality depends on the **rsyslog** package.

See Also

This is generator function for log_appender, for alternatives, see eg appender_console, appender_file, appender_tee, appender_pushbullet, appender_telegram, appender_kinesis and appender_async for evaluate any log_appender function in a background process.

Examples

```
## Not run:
if (requireNamespace("rsyslog", quietly = TRUE)) {
  log_appender(appender_syslog("test"))
  log_info("Test message.")
}

## End(Not run)
```

---

appender_syslognet *Send log messages to a network syslog server*

Description

Send log messages to a network syslog server

Usage

```
appender_syslognet(identifier, server, port = 601L)
```

Arguments

identifier	program/function identification (string).
server	machine where syslog daemon runs (string).
port	port where syslog daemon listens (integer).

Value

A function taking a lines argument.

Note

This functionality depends on the **syslognet** package.

Examples

```
## Not run:
if (requireNamespace("syslognet", quietly = TRUE)) {
  log_appender(appender_syslognet("test_app", 'remoteserver'))
  log_info("Test message.")
}

## End(Not run)
```

appender_tee *Append log messages to a file and stdout as well*

Description

This appends log messages to both console and a file. The same rotation options are available as in appender_file.

Usage

```
appender_tee(
  file,
  append = TRUE,
  max_lines = Inf,
  max_bytes = Inf,
  max_files = 1L
)
```

Arguments

file	path
append	boolean passed to cat defining if the file should be overwritten with the most recent log message instead of appending
max_lines	numeric specifying the maximum number of lines allowed in a file before rotating
max_bytes	numeric specifying the maximum number of bytes allowed in a file before rotating
max_files	integer specifying the maximum number of files to be used in rotation

Value

function taking lines argument

See Also

This is generator function for log_appender, for alternatives, see eg appender_console, appender_file, appender_slack, appender_pushbullet, appender_telegram, appender_syslog, appender_kinesis and appender_async for evaluate any log_appender function in a background process.

`appender_telegram` *Send log messages to a Telegram chat*

Description

Send log messages to a Telegram chat

Usage

```
appender_telegram(
  chat_id = Sys.getenv("TELEGRAM_CHAT_ID"),
  bot_token = Sys.getenv("TELEGRAM_BOT_TOKEN"),
  parse_mode = NULL
)
```

Arguments

`chat_id`	Unique identifier for the target chat or username of the target channel (in the format @channelusername)
`bot_token`	Telegram Authorization token
`parse_mode`	Message parse mode. Allowed values: Markdown or HTML

Value

function taking `lines` argument

Note

This functionality depends on the **telegram** package.

See Also

This is generator function for `log_appender`, for alternatives, see eg `appender_console`, `appender_file`, `appender_tee`, `appender_pushbullet`, `appender_syslog`, `appender_kinesis` and `appender_async` for evaluate any `log_appender` function in a background process.

colorize_by_log_level *Colorize string by the related log level*

Description

Adding color to a string to be used in terminal output. Supports ANSI standard colors 8 or 256.

Usage

```
colorize_by_log_level(msg, level)
```

Arguments

msg	string
level	see log_levels

Value

string with ANSI escape code

Examples

```
## Not run:
cat(colorize_by_log_level(FATAL, 'foobar'), '\n')
cat(colorize_by_log_level(ERROR, 'foobar'), '\n')
cat(colorize_by_log_level(WARN, 'foobar'), '\n')
cat(colorize_by_log_level(SUCCESS, 'foobar'), '\n')
cat(colorize_by_log_level(INFO, 'foobar'), '\n')
cat(colorize_by_log_level(DEBUG, 'foobar'), '\n')
cat(colorize_by_log_level(TRACE, 'foobar'), '\n')

## End(Not run)
```

deparse_to_one_line *Deparse and join all lines into a single line*

Description

Calling deparse and joining all the returned lines into a single line, separated by whitespace, and then cleaning up all the duplicated whitespace (except for excessive whitespace in strings between single or double quotes).

Usage

```
deparse_to_one_line(x)
```

Arguments

x	object to deparse

Value

string

`fail_on_missing_package` *Check if R package can be loaded and fails loudly otherwise*

Description

Check if R package can be loaded and fails loudly otherwise

Usage

```
fail_on_missing_package(pkg, min_version)
```

Arguments

pkg	string
min_version	optional minimum version needed

Examples

```
## Not run:
f <- function() fail_on_missing_package('foobar')
f()
g <- function() fail_on_missing_package('stats')
g()

## End(Not run)
```

`FATAL` *Log levels*

Description

The standard Apache logj4 log levels plus a custom level for `SUCCESS`. For the full list of these log levels and suggested usage, check the below Details.

Usage

```
TRACE

DEBUG

INFO

SUCCESS

WARN

ERROR

FATAL
```

Format

An object of class `loglevel` (inherits from `integer`) of length 1.

Details

List of supported log levels:

1. `FATAL` severe error that will prevent the application from continuing
2. `ERROR` An error in the application, possibly recoverable
3. `WARN` An event that might possible lead to an error
4. `SUCCESS` An explicit success event above the INFO level that you want to log
5. `INFO` An event for informational purposes
6. `DEBUG` A general debugging event
7. `TRACE` A fine-grained debug message, typically capturing the flow through the application.

References

https://logging.apache.org/log4j/2.0/log4j-api/apidocs/org/apache/logging/log4j/Level.html, https://logging.apache.org/log4j/2.x/manual/customloglevels.html

---

formatter_glue *Apply* glue *to convert R objects into a character vector*

Description

Apply `glue` to convert R objects into a character vector

Usage

```
formatter_glue(
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`...`	passed to `glue` for the text interpolation
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

Note

Although this is the default log message formatter function, but when **glue** is not installed, `formatter_sprintf` will be used as a fallback.

See Also

This is a `log_formatter`, for alternatives, see `formatter_paste`, `formatter_sprintf`, `formatter_glue_or_sprintf`, `formatter_logging`, `formatter_json`, `formatter_pander` and `skip_formatter` for marking a string not to apply the formatter on it.

---

formatter_glue_or_sprintf

Apply `glue` *and* `sprintf`

Description

The best of both words: using both formatter functions in your log messages, which can be useful eg if you are migrating from `sprintf` formatted log messages to `glue` or similar.

Usage

```
formatter_glue_or_sprintf(
  msg,
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

msg	passed to sprintf as fmt or handled as part of ... in glue
...	passed to glue for the text interpolation
.logcall	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
.topcall	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
.topenv	original frame of the .topcall calling function where the formatter function will be evaluated and that is used to look up the namespace as well via logger:::top_env_name

Details

Note that this function tries to be smart when passing arguments to glue and sprintf, but might fail with some edge cases, and returns an unformatted string.

Value

character vector

See Also

This is a log_formatter, for alternatives, see formatter_paste, formatter_sprintf, formatter_glue_or_sprintf, formatter_logging, formatter_json, formatter_pander and skip_formatter for marking a string not to apply the formatter on it.

Examples

```
## Not run:
formatter_glue_or_sprintf("{a} + {b} = %s", a = 2, b = 3, 5)
formatter_glue_or_sprintf("{pi} * {2} = %s", pi*2)
formatter_glue_or_sprintf("{pi} * {2} = {pi*2}")

formatter_glue_or_sprintf("Hi ", "{c('foo', 'bar')}, did you know that 2*4={2*4}")
formatter_glue_or_sprintf("Hi {c('foo', 'bar')}, did you know that 2*4={2*4}")
formatter_glue_or_sprintf("Hi {c('foo', 'bar')}, did you know that 2*4=%s", 2*4)
formatter_glue_or_sprintf("Hi %s, did you know that 2*4={2*4}", c('foo', 'bar'))
formatter_glue_or_sprintf("Hi %s, did you know that 2*4=%s", c('foo', 'bar'), 2*4)

## End(Not run)
```

`formatter_json` *Transforms all passed R objects into a JSON list*

Description

Transforms all passed R objects into a JSON list

Usage

```
formatter_json(
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`...`	passed to `toJSON` wrapped into a `list`
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

Note

This functionality depends on the **jsonlite** package.

See Also

This is a `log_formatter` potentially to be used with `layout_json_parser`, for alternatives, see `formatter_paste`, `formatter_sprintf`, `formatter_glue`, `formatter_glue_or_sprintf`, `formatter_logging`, `formatter_pander` and `skip_formatter` for marking a string not to apply the formatter on it.

Examples

```
## Not run:
log_formatter(formatter_json())
log_layout(layout_json_parser())
log_info(everything = 42)
log_info(mtcars = mtcars, species = iris$Species)
```

```
## End(Not run)
```

`formatter_logging` *Mimic the default formatter used in the* **logging** *package*

Description

The **logging** package uses a formatter that behaves differently when the input is a string or other R object. If the first argument is a string, then `sprintf` is being called – otherwise it does something like `log_eval` and logs the R expression(s) and the result(s) as well.

Usage

```
formatter_logging(
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`...`	string and further params passed to `sprintf` or R expressions to be evaluated
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

See Also

This is a `log_formatter`, for alternatives, see `formatter_paste`, `formatter_glue`, `formatter_glue_or_sprintf`, `formatter_json`, `formatter_pander` and `skip_formatter` for marking a string not to apply the formatter on it.

Examples

```
## Not run:
log_formatter(formatter_logging)
log_info('42')
log_info(42)
log_info(4+2)
log_info('foo %s', 'bar')
log_info('vector %s', 1:3)
log_info(12, 1+1, 2 * 2)

## End(Not run)
```

`formatter_pander` *Formats R objects with pander*

Description

Formats R objects with pander

Usage

```
formatter_pander(
  x,
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`x`	object to be logged
`...`	optional parameters passed to `pander`
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

Note

This functionality depends on the **pander** package.

See Also

This is a log_formatter, for alternatives, see formatter_paste, formatter_sprintf, formatter_glue, formatter_glue_or_sprintf, formatter_logging

Examples

```
## Not run:
log_formatter(formatter_pander)
log_info('42')
log_info(42)
log_info(4+2)
log_info(head(iris))
log_info(head(iris), style = 'simple')
log_info(lm(hp ~ wt, mtcars))

## End(Not run)
```

---

formatter_paste *Concatenate R objects into a character vector via* paste

---

Description

Concatenate R objects into a character vector via paste

Usage

```
formatter_paste(
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

...	passed to paste
.logcall	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
.topcall	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
.topenv	original frame of the .topcall calling function where the formatter function will be evaluated and that is used to look up the namespace as well via logger:::top_env_name

Value

character vector

See Also

This is a log_formatter, for alternatives, see formatter_sprintf, formatter_glue, formatter_glue_or_sprintf, formatter_logging, formatter_json, formatter_pander and skip_formatter for marking a string not to apply the formatter on it.

`formatter_sprintf` *Apply* `sprintf` *to convert R objects into a character vector*

Description

Apply `sprintf` to convert R objects into a character vector

Usage

```
formatter_sprintf(
  fmt,
  ...,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`fmt`	passed to `sprintf`
`...`	passed to `sprintf`
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

See Also

This is a log_formatter, for alternatives, see formatter_paste, formatter_glue, formatter_glue_or_sprintf, formatter_logging, formatter_json, formatter_pander and skip_formatter for marking a string not to apply the formatter on it.

`get_logger_meta_variables` *Collect useful information about the logging environment to be used in log messages*

Description

Available variables to be used in the log formatter functions, eg in `layout_glue_generator`:

- levelr: log level as an R object, eg `INFO`
- level: log level as a string, eg `INFO`
- time: current time as `POSIXct`
- node: name by which the machine is known on the network as reported by `Sys.info`
- arch: machine type, typically the CPU architecture
- os_name: Operating System's name
- os_release: Operating System's release
- os_version: Operating System's version
- user: name of the real user id as reported by `Sys.info`
- pid: the process identification number of the R session
- node: name by which the machine is known on the network as reported by `Sys.info`
- r_version: R's major and minor version as a string
- ns: namespace usually defaults to `global` or the name of the holding R package of the calling the logging function
- ns_pkg_version: the version of `ns` when it's a package
- ans: same as `ns` if there's a defined `logger` for the namespace, otherwise a fallback namespace (eg usually `global`)
- topenv: the name of the top environment from which the parent call was called (eg R package name or `GlobalEnv`)
- call: parent call (if any) calling the logging function
- fn: function's (if any) name calling the logging function

Usage

```
get_logger_meta_variables(
  log_level = NULL,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

log_level	log level as per log_levels
namespace	string referring to the logger environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
.logcall	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
.topcall	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
.topenv	original frame of the .topcall calling function where the formatter function will be evaluated and that is used to look up the namespace as well via logger:::top_env_name

Value

list

See Also

layout_glue_generator

---

grayscale_by_log_level

Render a string with light/dark gray based on the related log level

---

Description

Adding color to a string to be used in terminal output. Supports ANSI standard colors 8 or 256.

Usage

```
grayscale_by_log_level(msg, level)
```

Arguments

msg	string
level	see log_levels

Value

string with ANSI escape code

Examples

```
## Not run:
cat(grayscale_by_log_level(FATAL, 'foobar'), '\n')
cat(grayscale_by_log_level(ERROR, 'foobar'), '\n')
cat(grayscale_by_log_level(WARN, 'foobar'), '\n')
cat(grayscale_by_log_level(SUCCESS, 'foobar'), '\n')
cat(grayscale_by_log_level(INFO, 'foobar'), '\n')
cat(grayscale_by_log_level(DEBUG, 'foobar'), '\n')
cat(grayscale_by_log_level(TRACE, 'foobar'), '\n')

## End(Not run)
```

---

layout_blank *Format a log record by including the raw message without anything added or modified*

Description

Format a log record by including the raw message without anything added or modified

Usage

```
layout_blank(
  level,
  msg,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

level	log level, see log_levels for more details
msg	string message
namespace	string referring to the logger environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
.logcall	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
.topcall	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
.topenv	original frame of the .topcall calling function where the formatter function will be evaluated and that is used to look up the namespace as well via logger:::top_env_name

Value

character vector

See Also

This is a log_layout, for alternatives, see layout_simple, layout_glue_colors, layout_json, or generator functions such as layout_glue_generator

---

`layout_glue` *Format a log message with* `glue`

Description

By default, this layout includes the log level of the log record as per log_levels, the current timestamp and the actual log message – that you can override via calling layout_glue_generator directly. For colorized output, see layout_glue_colors.

Usage

```
layout_glue(
  level,
  msg,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`level`	log level, see log_levels for more details
`msg`	string message
`namespace`	string referring to the `logger` environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

See Also

This is a log_layout, for alternatives, see layout_blank, layout_simple, layout_glue_colors, layout_json, layout_json_parser, or generator functions such as layout_glue_generator

---

layout_glue_colors *Format a log message with* glue *and ANSI escape codes to add colors*

Description

Format a log message with glue and ANSI escape codes to add colors

Usage

```
layout_glue_colors(
  level,
  msg,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

level	log level, see log_levels for more details
msg	string message
namespace	string referring to the logger environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
.logcall	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
.topcall	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
.topenv	original frame of the .topcall calling function where the formatter function will be evaluated and that is used to look up the namespace as well via logger:::top_env_name

Value

character vector

Note

This functionality depends on the **crayon** package.

See Also

This is a log_layout, for alternatives, see layout_blank, layout_simple, layout_glue, layout_json, layout_json_parser, or generator functions such as layout_glue_generator

Examples

```
## Not run:
log_layout(layout_glue_colors)
log_threshold(TRACE)
log_info('Starting the script...')
log_debug('This is the second line')
log_trace('That is being placed right after the first one.')
log_warn('Some errors might come!')
log_error('This is a problem')
log_debug('Getting an error is usually bad')
log_error('This is another problem')
log_fatal('The last problem.')

## End(Not run)
```

---

layout_glue_generator *Generate log layout function using common variables available via glue syntax*

Description

format is passed to glue with access to the below variables:

- msg: the actual log message
- further variables set by get_logger_meta_variables

Usage

```
layout_glue_generator(
  format = "{level} [{format(time, \"%Y-%m-%d %H:%M:%S\")}] {msg}"
)
```

Arguments

format	glue-flavored layout of the message using the above variables

Value

function taking level and msg arguments - keeping the original call creating the generator in the generator attribute that is returned when calling log_layout for the currently used layout

See Also

See example calls from layout_glue and layout_glue_colors.

Examples

```
## Not run:
example_layout <- layout_glue_generator(
  format = '{node}/{pid}/{ns}/{ans}/{topenv}/{fn} {time} {level}: {msg}')
example_layout(INFO, 'try {runif(1)}')

log_layout(example_layout)
log_info('try {runif(1)}')

## End(Not run)
```

`layout_json` *Generate log layout function rendering JSON*

Description

Generate log layout function rendering JSON

Usage

```
layout_json(
  fields = c("time", "level", "ns", "ans", "topenv", "fn", "node", "arch", "os_name",
    "os_release", "os_version", "pid", "user", "msg")
)
```

Arguments

`fields`	character vector of field names to be included in the JSON

Value

character vector

Note

This functionality depends on the **jsonlite** package.

See Also

This is a `log_layout`, for alternatives, see `layout_blank`, `layout_simple`, `layout_glue`, `layout_glue_colors`, `layout_json_parser`, or generator functions such as `layout_glue_generator`

Examples

```
## Not run:
log_layout(layout_json())
log_info(42)
log_info('ok {1:3} + {1:3} = {2*(1:3)}')

## End(Not run)
```

layout_json_parser *Generate log layout function rendering JSON after merging meta fields with parsed list from JSON message*

Description

Generate log layout function rendering JSON after merging meta fields with parsed list from JSON message

Usage

```
layout_json_parser(
 fields = c("time", "level", "ns", "ans", "topenv", "fn", "node", "arch", "os_name",
    "os_release", "os_version", "pid", "user")
)
```

Arguments

fields	character vector of field names to be included in the JSON

Note

This functionality depends on the **jsonlite** package.

See Also

This is a log_layout potentially to be used with formatter_json, for alternatives, see layout_simple, layout_glue, layout_glue_colors, layout_json or generator functions such as layout_glue_generator

Examples

```
## Not run:
log_formatter(formatter_json)
log_info(everything = 42)
log_layout(layout_json_parser())
log_info(everything = 42)
log_layout(layout_json_parser(fields = c('time', 'node')))
log_info(cars = row.names(mtcars), species = unique(iris$Species))

## End(Not run)
```

`layout_logging` *Format a log record as the logging package does by default*

Description

Format a log record as the logging package does by default

Usage

```
layout_logging(
  level,
  msg,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`level`	log level, see `log_levels` for more details
`msg`	string message
`namespace`	string referring to the `logger` environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

See Also

This is a `log_layout`, for alternatives, see `layout_blank`, `layout_glue`, `layout_glue_colors`, `layout_json`, `layout_json_parser`, or generator functions such as `layout_glue_generator`

Examples

```
## Not run:
log_layout(layout_logging)
log_info(42)
log_info(42, namespace = 'everything')

devtools::load_all(system.file('demo-packages/logger-tester-package', package = 'logger'))
logger_tester_function(INFO, 42)

## End(Not run)
```

`layout_simple` *Format a log record by concatenating the log level, timestamp and message*

Description

Format a log record by concatenating the log level, timestamp and message

Usage

```
layout_simple(
  level,
  msg,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`level`	log level, see `log_levels` for more details
`msg`	string message
`namespace`	string referring to the `logger` environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

character vector

See Also

This is a log_layout, for alternatives, see layout_blank, layout_glue, layout_glue_colors, layout_json, layout_json_parser, or generator functions such as layout_glue_generator

---

`layout_syslognet` *Format a log record for syslognet*

Description

Format a log record for syslognet. This function converts the logger log level to a log severity level according to RFC 5424 "The Syslog Protocol".

Usage

```
layout_syslognet(
  level,
  msg,
  namespace = NA_character_,
  .logcall = sys.call(),
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

`level`	log level, see log_levels for more details
`msg`	string message
`namespace`	string referring to the `logger` environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
`.logcall`	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
`.topcall`	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
`.topenv`	original frame of the `.topcall` calling function where the formatter function will be evaluated and that is used to look up the `namespace` as well via `logger:::top_env_name`

Value

A character vector with a severity attribute.

logger *Generate logging utility*

Description

A logger consists of a log level `threshold`, a log message `formatter` function, a log record `layout` formatting function and the `appender` function deciding on the destination of the log record. For more details, see the package `README.md`.

Usage

```
logger(threshold, formatter, layout, appender)
```

Arguments

`threshold`	omit log messages below this `log_levels`
`formatter`	function pre-processing the message of the log record when it's not wrapped in a `skip_formatter` call
`layout`	function rendering the layout of the actual log record
`appender`	function writing the log record

Details

By default, a general logger definition is created when loading the `logger` package, that uses

1. `INFO` as the log level threshold
2. `layout_simple` as the layout function showing the log level, timestamp and log message
3. `formatter_glue` (or `formatter_sprintf` if **glue** is not installed) as the default formatter function transforming the R objects to be logged to a character vector
4. `appender_console` as the default log record destination

Value

function taking `level` and `msg` arguments

Note

It's quite unlikely that you need to call this function directly, but instead set the logger parameters and functions at `log_threshold`, `log_formatter`, `log_layout` and `log_appender` and then call `log_levels` and its derivatives, such as `log_info` directly.

References

For more details, see the Anatomy of a Log Request vignette at https://daroczig.github.io/logger/articles/anatomy.html.

Examples

```
## Not run:
do.call(logger, logger:::namespaces$global[[1]])(INFO, 42)
do.call(logger, logger:::namespaces$global[[1]])(INFO, '{pi}')
x <- 42
do.call(logger, logger:::namespaces$global[[1]])(INFO, '{x}^2 = {x^2}')

## End(Not run)
```

`log_appender` *Get or set log record appender function*

Description

Get or set log record appender function

Usage

```
log_appender(appender, namespace = "global", index = 1)
```

Arguments

appender	function delivering a log record to the destination, eg appender_console, appender_file or appender_tee
namespace	logger namespace
index	index of the logger within the namespace

See Also

logger, log_threshold, log_layout and log_formatter

Examples

```
## Not run:
## change appender to "tee" that writes to the console and a file as well
t <- tempfile()
log_appender(appender_tee(t))
log_info(42)
log_info(42:44)
readLines(t)

## poor man's tee by stacking loggers in the namespace
t <- tempfile()
log_appender(appender_console)
log_appender(appender_file(t), index = 2)
log_info(42)
readLines(t)

## End(Not run)
```

log_errors *Injects a logger call to standard errors*

Description

This function uses `trace` to add a `log_error` function call when `stop` is called to log the error messages with the `logger` layout and appender.

Usage

```
log_errors()
```

Examples

```
## Not run:
log_errors()
stop('foobar')

## End(Not run)
```

log_eval *Evaluate an expression and log results*

Description

Evaluate an expression and log results

Usage

```
log_eval(expr, level = TRACE, multiline = FALSE)
```

Arguments

`expr`	R expression to be evaluated while logging the expression itself along with the result
`level`	`log_levels`
`multiline`	setting to `FALSE` will print both the expression (enforced to be on one line by removing line-breaks if any) and its result on a single line separated by `=>`, while setting to `TRUE` will log the expression and the result in separate sections reserving line-breaks and rendering the printed results

Examples

```
## Not run:
log_eval(pi * 2, level = INFO)

## lowering the log level threshold so that we don't have to set a higher level in log_eval
log_threshold(TRACE)
log_eval(x <- 4)
log_eval(sqrt(x))

## log_eval can be called in-line as well as returning the return value of the expression
x <- log_eval(mean(runif(1e3)))
x

## https://twitter.com/krlmlr/status/1067864829547999232
f <- sqrt
g <- mean
x <- 1:31
log_eval(f(g(x)), level = INFO)
log_eval(y <- f(g(x)), level = INFO)

## returning a function
log_eval(f <- sqrt)
log_eval(f)

## evaluating something returning a wall of "text"
log_eval(f <- log_eval)
log_eval(f <- log_eval, multiline = TRUE)

## doing something computationally intensive
log_eval(system.time(for(i in 1:100) mad(runif(1000))), multiline = TRUE)

## End(Not run)
```

`log_failure` *Logs the error message to console before failing*

Description

Logs the error message to console before failing

Usage

```
log_failure(expression)
```

Arguments

`expression`	call

Examples

```
## Not run:
log_failure('foobar')
log_failure(foobar)

## End(Not run)
```

`log_formatter` *Get or set log message formatter*

Description

Get or set log message formatter

Usage

```
log_formatter(formatter, namespace = "global", index = 1)
```

Arguments

`formatter`	function defining how R objects are converted into a single string, eg `formatter_paste`, `formatter_sprintf`, `formatter_glue`, `formatter_glue_or_sprintf`, `formatter_logging`
`namespace`	logger namespace
`index`	index of the logger within the namespace

See Also

`logger`, `log_threshold`, `log_appender` and `log_layout`

`log_layout` *Get or set log record layout*

Description

Get or set log record layout

Usage

```
log_layout(layout, namespace = "global", index = 1)
```

Arguments

`layout`	function defining the structure of a log record, eg `layout_simple`, `layout_glue` or `layout_glue_colors`, `layout_json`, or generator functions such as `layout_glue_generator`
`namespace`	logger namespace
`index`	index of the logger within the namespace

See Also

logger, log_threshold, log_appender and log_formatter

Examples

```
## Not run:
log_layout(layout_json())
log_info(42)

## End(Not run)
```

---

log_level *Log a message with given log level*

Description

Log a message with given log level

Usage

```
log_level(level, ..., namespace = NA_character_,
  .logcall = sys.call(), .topcall = sys.call(-1), .topenv = parent.frame())

log_trace(...)

log_debug(...)

log_info(...)

log_success(...)

log_warn(...)

log_error(...)

log_fatal(...)
```

Arguments

level	log level, see log_levels for more details
...	R objects that can be converted to a character vector via the active message formatter function
namespace	string referring to the logger environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.

.logcall	the logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)
.topcall	R expression from which the logging function was called (useful in formatters and layouts to extract the calling function's name or arguments)
.topenv	original frame of the .topcall calling function where the formatter function will be evaluated and that is used to look up the namespace as well via logger:::top_env_name

See Also

logger

Examples

```
## Not run:
log_level(INFO, 'hi there')
log_info('hi there')

## output omitted
log_debug('hi there')

## lower threshold and retry
log_threshold(TRACE)
log_debug('hi there')

## multiple lines
log_info('ok {1:3} + {1:3} = {2*(1:3)}')

log_layout(layout_json())
log_info('ok {1:3} + {1:3} = {2*(1:3)}')

## note for the JSON output, glue is not automatically applied
log_info(glue::glue('ok {1:3} + {1:3} = {2*(1:3)}'))

## End(Not run)
```

---

log_messages *Injects a logger call to standard messages*

Description

This function uses trace to add a log_info function call when message is called to log the informative messages with the logger layout and appender.

Usage

```
log_messages()
```

Examples

```
## Not run:
log_messages()
message('hi there')

## End(Not run)
```

`log_namespaces` *Looks up logger namespaces*

Description

Looks up logger namespaces

Usage

```
log_namespaces()
```

Value

character vector of namespace names

`log_separator` *Logs a long line to stand out from the console*

Description

Logs a long line to stand out from the console

Usage

```
log_separator(
  level = INFO,
  namespace = NA_character_,
  separator = "=",
  width = 80
)
```

Arguments

`level`	log level, see `log_levels` for more details
`namespace`	string referring to the `logger` environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.
`separator`	character to be used as a separator
`width`	max width of message – longer text will be wrapped into multiple lines

See Also

log_with_separator

Examples

```
log_separator()
log_separator(ERROR, separator = '!', width = 60)
```

---

log_shiny_input_changes

Auto logging input changes in Shiny app

Description

This is to be called in the server section of the Shiny app.

Usage

```
log_shiny_input_changes(input, level = INFO, excluded_inputs = character())
```

Arguments

input	passed from Shiny's server
level	log level
excluded_inputs	character vector of input names to exclude from logging

Examples

```
## Not run:
library(shiny)

ui <- bootstrapPage(
    numericInput('mean', 'mean', 0),
    numericInput('sd', 'sd', 1),
    textInput('title', 'title', 'title'),
    textInput('foo', 'This is not used at all, still gets logged', 'foo'),
    passwordInput('password', 'Password not to be logged', 'secret'),
    plotOutput('plot')
)

server <- function(input, output) {

    logger::log_shiny_input_changes(input, excluded_inputs = 'password')

    output$plot <- renderPlot({
        hist(rnorm(1e3, input$mean, input$sd), main = input$title)
    })
```

```
}

shinyApp(ui = ui, server = server)

## End(Not run)
```

`log_threshold` *Get or set log level threshold*

Description

Get or set log level threshold

Usage

```
log_threshold(level, namespace = "global", index = 1)
```

Arguments

`level`	see `log_levels`
`namespace`	logger namespace
`index`	index of the logger within the namespace

Value

currently set log level threshold

See Also

`logger`, `log_layout`, `log_formatter`, `log_appender`

Examples

```
## Not run:
## check the currently set log level threshold
log_threshold()

## change the log level threshold to WARN
log_threshold(WARN)
log_info(1)
log_warn(2)

## add another logger with a lower log level threshold and check the number of logged messages
log_threshold(INFO, index = 2)
log_info(1)
log_warn(2)

## set the log level threshold in all namespaces to ERROR
```

```
log_threshold(ERROR, namespace =  log_namespaces())

## End(Not run)
```

---

log_tictoc *Tic-toc logging*

Description

Tic-toc logging

Usage

```
log_tictoc(..., level = INFO, namespace = NA_character_)
```

Arguments

...	passed to log_level
level	x
namespace	x

Author(s)

Thanks to Neal Fultz for the idea and original implementation!

Examples

```
## Not run:
log_tictoc('warming up')
Sys.sleep(0.1)
log_tictoc('running')
Sys.sleep(0.1)
log_tictoc('running')
Sys.sleep(runif(1))
log_tictoc('and running')

## End(Not run)
```

log_warnings *Injects a logger call to standard warnings*

Description

This function uses `trace` to add a `log_warn` function call when `warning` is called to log the warning messages with the `logger` layout and appender.

Usage

```
log_warnings()
```

Examples

```
## Not run:
log_warnings()
for (i in 1:5) { Sys.sleep(runif(1)); warning(i) }

## End(Not run)
```

log_with_separator *Logs a message in a very visible way*

Description

Logs a message in a very visible way

Usage

```
log_with_separator(
  ...,
  level = INFO,
  namespace = NA_character_,
  separator = "=",
  width = 80
)
```

Arguments

`...`	R objects that can be converted to a character vector via the active message formatter function
`level`	log level, see `log_levels` for more details
`namespace`	string referring to the `logger` environment / config to be used to override the target of the message record to be used instead of the default namespace, which is defined by the R package name from which the logger was called, and falls back to a common, global namespace.

separator	character to be used as a separator
width	max width of message – longer text will be wrapped into multiple lines

See Also

log_separator

Examples

```
log_with_separator('An important message')
log_with_separator('Some critical KPI down!!!', separator = '$')
log_with_separator('This message is worth a {1e3} words')
log_with_separator(paste(
  'A very important message with a bunch of extra words that will',
  'eventually wrap into a multi-line message for our quite nice demo :wow:'))
log_with_separator(paste(
  'A very important message with a bunch of extra words that will',
  'eventually wrap into a multi-line message for our quite nice demo :wow:'),
  width = 60)
log_with_separator('Boo!', level = FATAL)
```

---

skip_formatter *Adds the skip_formatter attribute to an object so that logger will skip calling the formatter function on the object(s) to be logged*

Description

Adds the skip_formatter attribute to an object so that logger will skip calling the formatter function on the object(s) to be logged

Usage

```
skip_formatter(message, ...)
```

Arguments

message	character vector directly passed to the appender function in logger
...	should be never set

Value

character vector with skip_formatter attribute set to TRUE

`with_log_threshold` *Evaluate R expression with a temporarily updated log level threshold*

Description

Evaluate R expression with a temporarily updated log level threshold

Usage

```
with_log_threshold(
  expression,
  threshold = ERROR,
  namespace = "global",
  index = 1
)
```

Arguments

`expression`	R command
`threshold`	log_levels
`namespace`	logger namespace
`index`	index of the logger within the namespace

Examples

```
## Not run:
log_threshold(TRACE)
log_trace('Logging everything!')
x <- with_log_threshold({
  log_info('Now we are temporarily suppressing eg INFO messages')
  log_warn('WARN')
  log_debug('Debug messages are suppressed as well')
  log_error('ERROR')
  invisible(42)
}, threshold = WARN)
x
log_trace('DONE')

## End(Not run)
```

`%except%` *Try to evaluate an expressions and evaluate another expression on exception*

Description

Try to evaluate an expressions and evaluate another expression on exception

Usage

```
try %except% except
```

Arguments

`try`	R expression
`except`	fallback R expression to be evaluated if `try` fails

Note

Suppress log messages in the `except` namespace if you don't want to throw a `WARN` log message on the exception branch.

Examples

```
everything %except% 42
everything <- '640kb'
everything %except% 42

Mean(1:10) %except% sum(1:10) / length(1:10)
Mean(1:10) %except% (sum(1:10) / length(1:10))
Mean(1:10) %except% MEAN(1:10) %except% mean(1:10)
Mean(1:10) %except% (MEAN(1:10) %except% mean(1:10))
```

Index