Package ‘logger’

January 2, 2019

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 0.1
Date 2018-12-20
URL https://github.com/daroczig/logger
Encoding UTF-8
RoxygenNote 6.1.0
License AGPL-3
Imports utils
Suggests glue, jsonlite, crayon, slackr, RPushbullet, testthat, covr, knitr, rmarkdown, devtools, roxygen2, parallel
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 2019-01-02 15:30:03 UTC

R topics documented:

appender_console ................................................................. 2
appender_file ................................................................. 3
appender_pushbullet .............................................................. 3
appender_slack ................................................................. 4
appender_tee ................................................................. 5
colorize_by_log_level ........................................................ 5
appender_console

Append log record to stdout

Description

Append log record to stdout

Usage

appender_console(lines)

Arguments

lines character vector

See Also

This is a log_appender, for alternatives, see eg appender_file, appender_tee, appender_slack, appender_pushbullet
appender_file  Append log messages to a file

Description

Append log messages to a file

Usage

appender_file(file)

Arguments

file   path

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_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 RPusih bullet package.
See Also

This is generator function for log_appender, for alternatives, see eg appender_console, appender_file, appender_tee, appender_slack

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"),
icong_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_tee**

Append log messages to a file and stdout as well

**Description**

Append log messages to a file and stdout as well

**Usage**

appender_tee(file)

**Arguments**

- **file**: path

**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

---

**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

```r
## 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)
```

### Log levels

<table>
<thead>
<tr>
<th>Log level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FATAL</strong></td>
</tr>
</tbody>
</table>

**Description**

The standard Apache log4j 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

---

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_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

**Examples**

```r
## 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_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), .topev = 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)
.topev 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

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)
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_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
...
.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
**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`
- `ns`: namespace usually defaults to `global` or the name of the holding R package of the calling the logging function
- `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**

```r
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`
grayscale_by_log_level

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**

```r
## 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)
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

```r
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_simple`, `layout_glue_colors`, `layout_json`, or generator functions such as `layout_glue_generator`
**Description**

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

**Usage**

```r
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_simple`, `layout_glue`, `layout_json`, or generator functions such as `layout_glue_generator`
Examples

```plaintext
## 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

```plaintext
layout_glue_generator(format = "(level) [{format(time, "%Y-%d-%m %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

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

log_layout(ex)
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_simple, layout_glue, layout_glue_colors or generator functions such as layout_glue_generator

**Examples**

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

## 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

```r
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_glue, layout_glue_colors, layout_json`, or generator functions such as `layout_glue_generator`

Examples

```r
## 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)
```
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_glue`, `layout_glue_colors`, `layout_json`, or generator functions such as `layout_glue_generator`
**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**

```r
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, "\(x^2 = \{x^2\}\)"
```

**Description**
Get or set log record appender function

**Usage**

```r
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**

```r
## 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_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 <- 1:3
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)
```
```r
## doing something computationally intensive
log_eval(system.time(for(i in 1:100) mad(runif(1000))), multiline = TRUE)
## End(Not run)
```

---

### log_formatter

*Get or set log message formatter*

#### Description

Get or set log message formatter

#### Usage

```r
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

```r
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
log_level

See Also

logger, log_threshold, log_appender and log_formatter

Examples

```r
## 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

```r
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.
log_threshold

.\logcall\nthe logging call being evaluated (useful in formatters and layouts when you want to have access to the raw, unevaluated R expression)

.topcall\nR expression from which the logging function was called (useful in formatters and layouts to extract the calling function’s name or arguments)

.topenv\norIGINAL 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\n
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)
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

```r
## 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)

## End(Not run)
```

---

```
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
```
with_log_threshold

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)
Index

*Topic datasets
   FATAL, 6

appender_console, 2, 3–5, 20, 21
appender_file, 2, 3, 4, 5, 21
appender_pushbullet, 2, 3, 3, 4, 5
appender_slack, 2–4, 4, 5
appender_tee, 2–4, 5, 21

colorize_by_log_level, 5

DEBUG (FATAL), 6

ERROR (FATAL), 6

FATAL, 6
formatter_glue, 7, 9–11, 20, 23
formatter_glue_or_sprintf, 7, 8, 9–11, 23
formatter_logging, 7, 8, 9, 10, 11, 23
formatter_paste, 7–9, 10, 11, 23
formatter_sprintf, 7, 8, 10, 11, 20, 23

get_logger_meta_variables, 11, 16
grayscale_by_log_level, 13

INFO, 12, 20
INFO (FATAL), 6

layout_glue, 14, 15–19, 23
layout_glue_colors, 14, 15, 16–19, 23
layout_glue_generator, 12–15, 16, 17–19, 23
layout_json, 14, 15, 17, 18, 19, 23
layout_logging, 18
layout_simple, 14, 15, 17, 19, 20, 23
log_appender, 2–5, 20, 21, 23, 24, 26
log_debug (log_level), 24
log_error (log_level), 24
log_eval, 9, 22
log_fatal (log_level), 24
log_formatter, 7–11, 20, 21, 23, 24, 26
log_info, 20
log_info (log_level), 24
log_layout, 14–21, 23, 26
log_level, 24
log_levels, 5, 12–15, 18–20, 22, 24, 26, 27
log_levels (FATAL), 6
log_levels (FATAL), 24
log_success (log_level), 24
log_threshold, 20, 21, 23, 24, 25
log_trace (log_level), 24
log_warn (log_level), 24
logger, 12, 20, 21, 23–26

skip_formatter, 20, 26
sprintf, 9
SUCCESS (FATAL), 6

TRACE (FATAL), 6

WARN (FATAL), 6

with_log_threshold, 27

28