Package: log4r (via r-universe)

October 18, 2024

Type Package

Title A Fast and Lightweight Logging System for R, Based on 'log4j'

Version 0.4.4.9000

Description The log4r package is meant to provide a fast, lightweight, object-oriented approach to logging in R based on the widely-emulated 'log4j' system and etymology.

License Artistic-2.0

URL https://github.com/johnmyleswhite/log4r, https://log4r.r-lib.org

BugReports https://github.com/johnmyleswhite/log4r/issues

Imports cli, lifecycle, rlang

Suggests futile.logger, httr, jsonlite, knitr, lgr, logger, logging, loggit, microbenchmark, rlog, rmarkdown, rsyslog, testthat (>= 3.0.0)

Encoding UTF-8

LazyLoad yes

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

Collate 'appenders.R' 'logfuncs.R' 'deprecated.R' 'layouts.R' 'level.R' 'log4r-package.R' 'logger.R'

Config/Needs/website tidyverse/tidytemplate

Repository https://r-lib.r-universe.dev

RemoteUrl https://github.com/r-lib/log4r

RemoteRef HEAD

RemoteSha ab0b7e97c12bf4b7db4209e1de379f4985d39d36

Contents

`appenders` *Send logs to their final destination with Appenders*

Description

In log4j etymology, **Appenders** are destinations where logs are written. Appenders have no control over formatting; this is controlled by the **Layout**.

The most basic appenders write logs to the console or to a file; these are described below.

For implementing your own appenders, see Details.

Usage

```
console_appender(layout = default_log_layout())

file_appender(file, append = TRUE, layout = default_log_layout())
```

Arguments

`layout`	A layout function taking a `level` parameter and additional arguments corresponding to the message. See `layouts()`.
`file`	The file to write messages to.
`append`	When `TRUE`, the file is not truncated when opening for the first time.

Details

Appenders are implemented as functions with the interface `function(level, ...)`. These functions are expected to write their arguments to a destination and return `invisible(NULL)`.

See Also

`tcp_appender()`, `http_appender()`, `syslog_appender()`

Examples

```
# The behaviour of an appender can be seen by using them directly; the
# following snippet will write the message to the console.
appender <- console_appender()
appender("INFO", "Input has length ", 0, ".")
```

`http_appender` *Send logs over HTTP*

Description

Send logs in the body of HTTP requests. Responses with status code 400 or above will trigger errors.

Requires the `httr` package.

Usage

```
http_appender(url, method = "POST", layout = default_log_layout(), ...)
```

Arguments

`url`	The URL to submit messages to.
`method`	The HTTP method to use, usually `"POST"` or `"GET"`.
`layout`	A layout function taking a `level` parameter and additional arguments corresponding to the message.
`...`	Further arguments passed on to `httr::POST()`.

See Also

appenders for more information on Appenders.

Examples

```
## Not run:
# POST messages to localhost.
appender <- http_appender("localhost")
appender("INFO", "Message.")

# POST JSON-encoded messages.
appender <- http_appender(
  "localhost", method = "POST", layout = default_log_layout(),
  httr::content_type_json()
)
appender("INFO", "Message.")

## End(Not run)
```

layouts *Format logs with Layouts*

Description

In log4j etymology, **Layouts** are how **Appenders** control the format of messages. Most users will use one of the general-purpose layouts provided by the package:

- `default_log_layout()` formats messages much like the original log4j library. `simple_log_layout()` does the same, but omits the timestamp.
- `bare_log_layout()` emits only the log message, with no level or timestamp fields.
- `logfmt_log_layout()` and `json_log_layout()` format structured logs in the two most popular machine-readable formats.

For implementing your own layouts, see Details.

Usage

```
default_log_layout(time_format = "%Y-%m-%d %H:%M:%S")

simple_log_layout()

bare_log_layout()

logfmt_log_layout()

json_log_layout()
```

Arguments

`time_format`	A valid format string for timestamps. See `base::strptime()`.

Details

Layouts return a function with the signature `function(level, ...)` that itself returns a single newline-terminated string. Anything that meets this interface can be passed as a layout to one of the existing appenders.

`json_log_layout` requires the `jsonlite` package.

Examples

```
# The behaviour of a layout can be seen by using them directly:
simple <- simple_log_layout()
simple("INFO", "Input has length ", 0, ".")

with_timestamp <- default_log_layout()
with_timestamp("INFO", "Input has length ", 0, ".")
```

```
logfmt <- logfmt_log_layout()
logfmt("INFO", msg = "got input", length = 24)
```

`level` *Set the logging threshold level for a logger dynamically*

Description

It can sometimes be useful to change the logging threshold level at runtime. The `level()` accessor allows doing so.

Usage

```
level(x)

level(x) <- value

## S3 method for class 'logger'
level(x)

## S3 replacement method for class 'logger'
level(x) <- value

available.loglevels()
```

Arguments

`x`	An object of class `"logger"`.
`value`	One of `"DEBUG"`, `"INFO"`, `"WARN"`, `"ERROR"`, or `"FATAL"`.

Examples

```
lgr <- logger()
level(lgr) # Prints "INFO".
info(lgr, "This message is shown.")
level(lgr) <- "FATAL"
info(lgr, "This message is now suppressed.")
```

logger *Create a logger*

Description

This is the main interface for configuring logging behaviour. We adopt the well-known log4j etymology: **Appenders** are destinations (e.g. the console or a file) where logs are written, and the **Layout** is the format of these logs.

Usage

```
logger(threshold = "INFO", appenders = console_appender())
```

Arguments

`threshold`	The logging threshold, one of `"DEBUG"`, `"INFO"`, `"WARN"`, `"ERROR"`, or `"FATAL"`. Logs with a lower severity than the threshold will be discarded.
`appenders`	The logging appenders; both single appenders and a `list()` of them are supported. See **Appenders**.

Value

An object of class `"logger"`.

See Also

Appenders and **Layouts** for information on controlling the behaviour of the logger object.

Examples

```
# By default, logs are written to the console at the "INFO" threshold.
logger <- logger()

log_info(logger, "Located nearest gas station.")
log_warn(logger, "Ez-Gas sensor network is not available.")
log_debug(logger, "Debug messages are suppressed by default.")
```

log_at *Write logs at a given level*

Description

Write logs at a given level

Usage

```
log_at(logger, level, ...)

log_debug(logger, ...)

log_info(logger, ...)

log_warn(logger, ...)

log_error(logger, ...)

log_fatal(logger, ...)
```

Arguments

logger	An object of class "logger".
level	The desired severity, one of "DEBUG", "INFO", "WARN", "ERROR", or "FATAL". Messages with a lower severity than the logger threshold will be discarded.
...	One or more items to log.

Examples

```
logger <- logger()

log_at(logger, "WARN", "First warning from our code")
log_debug(logger, "Debugging our code")
log_info(logger, "Information about our code")
log_warn(logger, "Another warning from our code")
log_error(logger, "An error from our code")
log_fatal(logger, "I'm outta here")
```

---

syslog_appender *Send logs to the local syslog*

Description

Send messages to the local syslog. Requires the rsyslog package.

Usage

```
syslog_appender(identifier, layout = bare_log_layout(), ...)
```

Arguments

identifier	A string identifying the application.
layout	A layout function taking a level parameter and additional arguments corresponding to the message.
...	Further arguments passed on to rsyslog::open_syslog().

See Also

appenders for more information on Appenders.

`tcp_appender` *Send logs over TCP*

Description

Append messages to arbitrary TCP destinations.

Usage

```
tcp_appender(
  host,
  port,
  layout = default_log_layout(),
  timeout = getOption("timeout")
)
```

Arguments

`host`	Hostname for the socket connection.
`port`	Port number for the socket connection.
`layout`	A layout function taking a `level` parameter and additional arguments corresponding to the message.
`timeout`	Timeout for the connection.

See Also

appenders for more information on Appenders, and `base::socketConnection()` for the underlying connection object used by `tcp_appender()`.

Index