

Package: log4r (via r-universe)

August 27, 2024

Type Package

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

Version 0.4.2

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>

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

Suggests futile.logger, httr, jsonlite, knitr, lgr, logger, logging, loggit, microbenchmark, rlog, rmarkdown, rsyslog, testthat

Encoding UTF-8

LazyLoad yes

Roxygen list(old_usage = TRUE)

RoxygenNote 7.1.1

VignetteBuilder knitr

Repository <https://johnmyleswhite.r-universe.dev>

RemoteUrl <https://github.com/johnmyleswhite/log4r>

RemoteRef HEAD

RemoteSha b9f4881cd5f21540ffe5b9dae33eb5bea884f680

Contents

log4r-package	2
appenders	3
create.logger	4
http_appender	4
layouts	5
level	6
levellog	7

logfile	8
logformat	9
logger	9
loglevel	10
syslog_appender	12
tcp_appender	12

Index	14
--------------	-----------

log4r-package	<i>A simple logging system for R, based on log4j.</i>
---------------	---

Description

logr4 provides an object-oriented logging system that uses an API roughly equivalent to log4j and its related variants.

Details

Package:	log4r
Type:	Package
Version:	0.2
Date:	2014-09-29
License:	Artistic-2.0
LazyLoad:	yes

Maintainer: Kirill Müller <krmlr+r@mailbox.org>

URL: <https://github.com/johnmyleswhite/log4r>

Issue tracker: <https://github.com/johnmyleswhite/log4r/issues>

References

See the log4j documentation or the documentation for its many derivatives to understand the origins of this logging system.

Examples

```
# Import the log4r package.
library('log4r')

# Create a new logger object with create.logger().
logger <- create.logger()

# Set the logger's file output.
logfile(logger) <- 'base.log'

# Set the current level of the logger.
```

```

level(logger) <- 'INFO'

# Try logging messages with different priorities.
# At priority level INFO, a call to debug() won't print anything.
debug(logger, 'A Debugging Message')
info(logger, 'An Info Message')
warn(logger, 'A Warning Message')
error(logger, 'An Error Message')
fatal(logger, 'A Fatal Error Message')

```

appenders

Appenders

Description

In [log4j](#) etymology, **Appenders** are destinations where messages are written. Depending on the nature of the destination, the format of the messages may be controlled using a [Layout](#).

The most basic appenders log messages 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, ".")

```

<code>create.logger</code>	<i>Creates a logger object.</i>
----------------------------	---------------------------------

Description

Creates a logger object.

Usage

```
create.logger(logfile = "logfile.log", level = "FATAL", logformat = NULL)
```

Arguments

<code>logfile</code>	The full pathname of the file you want log messages to be written to.
<code>level</code>	The level at which the logger is initialized. Will be coerced using as.loglevel .
<code>logformat</code>	The format string used when writing messages to the log file.

See Also

[loglevel](#), [level.logger](#)

Examples

```
library('log4r')

logger <- create.logger(logfile = 'debugging.log', level = "DEBUG")
```

<code>http_appender</code>	<i>Log Messages via HTTP</i>
----------------------------	------------------------------

Description

Send messages 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

<code>url</code>	The URL to submit messages to.
<code>method</code>	The HTTP method to use, usually "POST" or "GET".
<code>layout</code>	A layout function taking a <code>level</code> parameter and additional arguments corresponding to the message.
<code>...</code>	Further arguments passed on to 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

Layouts

Description

In [log4j](#) etymology, **Layouts** are how [Appenders](#) control the format of messages.

Some general-purpose layouts are described below.

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 [strptime](#). For some layouts this can be NA to elide the timestamp.

Details

Layouts are implemented as functions with the interface `function(level, ...)` and returning a single string.

`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, ".")
```

level	<i>Set or get the priority level for a logger object.</i>
-------	---

Description

The priority level can be an integer from the set 1..5 (otherwise it will be modified sensibly to fit in that range), or a named logging level (one of "DEBUG", "INFO", "WARN", "ERROR", or "FATAL"). An object of class `loglevel` is also accepted; other input will be coerced using [as.loglevel](#).

Usage

```
level(x)

level(x) <- value

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

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

Arguments

x	An object of class <code>logger</code> .
value	A <code>loglevel</code> .

See Also

[loglevel](#)

Examples

```
library('log4r')

logger <- create.logger(logfile = 'debugging.log', level = 1)
level(logger)
level(logger) <- "FATAL"
```

levellog	<i>Write messages to logs at a given priority level.</i>
----------	--

Description

Write messages to logs at a given priority level.

Usage

```
levellog(logger, level, ...)

debug(logger, ...)

info(logger, ...)

warn(logger, ...)

error(logger, ...)

fatal(logger, ...)
```

Arguments

logger	An object of class 'logger'.
level	The desired priority level: a number, a character, or an object of class 'loglevel'. Will be coerced using as.loglevel .
...	One or more items to be written to the log at the corresponding priority level.

See Also

[loglevel](#)

Examples

```
library('log4r')

logger <- create.logger(logfile = 'debugging.log', level = "WARN")

levellog(logger, 'WARN', 'First warning from our code')
debug(logger, 'Debugging our code')
```

```
info(logger, 'Information about our code')
warn(logger, 'Another warning from our code')
error(logger, 'An error from our code')
fatal(logger, "I'm outta here")
```

logfile

Get or set the logfile for a logger object.

Description

Get or set the logfile for a logger object.

Usage

```
logfile(x)

logfile(x) <- value

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

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

Arguments

x	An object of class logger.
value	The path name of a file to be used for logging. Must be a valid path in an already existing directory

Examples

```
library('log4r')

logger <- create.logger()
print(logfile(logger))
logfile(logger) <- 'debug.log'
debug(logger, 'A Debugging Message')
```

logformat	<i>Get or set the format string for a logger object.</i>
-----------	--

Description

Get or set the format string for a logger object.

Usage

```
logformat(x)

logformat(x) <- value

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

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

Arguments

x	An object of class logger.
value	A string containing a proper format string.

Examples

```
library('log4r')

logger <- create.logger(logfile = 'debugging.log', level = 'DEBUG')
print(logformat(logger))
logformat(logger) <- 'FORMAT STRING'
```

logger	<i>Create Logger Objects</i>
--------	------------------------------

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 messages are written, and the **Layout** is the format of the messages.

Usage

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

Arguments

threshold	The logging threshold level. Messages with a lower priority level will be discarded. See loglevel .
appenders	The logging appenders; both single appenders and a <code>list()</code> 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, messages are logged to the console at the
# "INFO" threshold.
logger <- logger()

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

loglevel

Logging levels

Description

Functions for handling logging levels. With each log entry, a logging level is associated that indicate its severity – debugging output, informational output, warning message, error message or fatal error. Each logger only prints log entries where the log level is equal or above its threshold.

Usage

```
loglevel(i)

is.loglevel(x, ...)

as.loglevel(i)

## S3 method for class 'loglevel'
print(x, ...)

## S3 method for class 'loglevel'
as.numeric(x, ...)
```

```
## S3 method for class 'loglevel'
as.character(x, ...)

available.loglevels()

verbosity(v)
```

Arguments

i	An integer from the set 1..5. Otherwise it will be modified sensibly to fit in that range. Alternatively, a named logging level (one of "DEBUG", "INFO", "WARN", "ERROR", or "FATAL").
x	An object of class "loglevel"
...	Unused
v	A verbosity level from the set 5..1. For historical reasons, they do not match the log levels; a verbosity level of 1 corresponds to a logging level of 5, 2 corresponds to 4, etc.

Details

To specify a logging level, use a character value, e.g. "WARN", or an integer between 1 and 5. The function `available.levels` lists all possible logging levels.

Value

An object of class "loglevel"

Examples

```
loglevel(2) == loglevel("INFO")
loglevel("WARN") < loglevel("ERROR")
loglevel(-1)
try(loglevel("UNDEFINED"))
is.loglevel("DEBUG")
is.loglevel(loglevel("DEBUG"))
as.numeric(loglevel("FATAL"))
available.loglevels()

## Not run:
library(optparse)
library(log4r)

optlist <- list(make_option(c('-v', '--verbosity-level'),
  type = "integer",
  dest = "verbosity",
  default = 1,
  help = "Verbosity threshold (5=DEBUG, 4=INFO 3=WARN, 2=ERROR, 1=FATAL)"))

optparser <- OptionParser(option_list=optlist)
```

```

opt <- parse_args(optparser)

my.logger <- create.logger(logfile = "", level = verbosity(opt$verbosity))

fatal(my.logger, "Fatal message")
error(my.logger, "Error message")
warn(my.logger, "Warning message")
info(my.logger, "Informational message")
debug(my.logger, "Debugging message")

## End(Not run)

```

syslog_appender	<i>Log Messages to the Local Syslog</i>
-----------------	---

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 open_syslog .

See Also

[appenders](#) for more information on Appenders.

tcp_appender	<i>Log Messages via TCP</i>
--------------	-----------------------------

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 [socketConnection](#) for the underlying connection object used by *tcp_appender*.

Index

- * **package**
 - log4r-package, 2
- Appenders, 5, 9, 10
- appenders, 3, 5, 10, 12, 13
- as.character.loglevel (loglevel), 10
- as.loglevel, 4, 6, 7
- as.loglevel (loglevel), 10
- as.numeric.loglevel (loglevel), 10
- available.loglevels (loglevel), 10

- bare_log_layout (layouts), 5

- console_appender (appenders), 3
- create.logger, 4

- debug (levellog), 7
- default_log_layout (layouts), 5

- error (levellog), 7

- fatal (levellog), 7
- file_appender (appenders), 3

- http_appender, 3, 4

- info (levellog), 7
- is.loglevel (loglevel), 10

- json_log_layout (layouts), 5

- Layout, 3, 9
- Layouts, 10
- layouts, 3, 5
- level, 6
- level.logger, 4
- level<- (level), 6
- levellog, 7
- log4r (log4r-package), 2
- log4r-package, 2
- logfile, 8

- logfile<- (logfile), 8
- logfmt_log_layout (layouts), 5
- logformat, 9
- logformat<- (logformat), 9
- logger, 9
- loglevel, 4, 6, 7, 10, 10

- open_syslog, 12

- POST, 4
- print.loglevel (loglevel), 10

- simple_log_layout (layouts), 5
- socketConnection, 13
- strptime, 5
- syslog_appender, 3, 12

- tcp_appender, 3, 12

- verbosity (loglevel), 10

- warn (levellog), 7