Package: PEcAn.logger (via r-universe)

September 18, 2024

Title Logger Functions for 'PEcAn'

Version 1.8.2.9000

Description Convenience functions for logging outputs from 'PEcAn', the Predictive Ecosystem Analyzer (LeBauer et al. 2017) <doi:10.1890/12-0137.1>. Enables the user to set what level of messages are printed, as well as whether these messages are written to the console, a file, or both. It also allows control over whether severe errors should stop execution of the 'PEcAn' workflow; this allows strictness when debugging and lenience when running large batches of simulations that should not be terminated by errors in individual models. It is loosely based on the 'log4j' package.

BugReports https://github.com/PecanProject/pecan/issues

URL https://pecanproject.github.io/

Imports utils, stringi

Suggests testthat, withr

License BSD_3_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/PecanProject/pecan

RemoteRef HEAD

RemoteSha f22a7c4bbc532e4551f7bc9624cef649da317ac1

Contents

`logger.debug` *Prints a debug message.*

Description

This function will print a debug message.

Usage

```
logger.debug(msg, ...)
```

Arguments

`msg`	the message that should be printed.
`...`	any additional text that should be printed.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.debug('variable', 5)

## End(Not run)
```

logger.error *Prints an error message.*

Description

This function will print an error message.

Usage

```
logger.error(msg, ...)
```

Arguments

msg	the message that should be printed.
...	any additional text that should be printed.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.error('system did not converge')

## End(Not run)
```

logger.getLevel *Get configured logging level.*

Description

This will return the current level configured of the logging messages

Usage

```
logger.getLevel()
```

Value

level the level of the message (ALL, DEBUG, INFO, WARN, ERROR, OFF)

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.getLevel()

## End(Not run)
```

`logger.info` *Prints an informational message.*

Description

This function will print an informational message.

Usage

```
logger.info(msg, ...)
```

Arguments

`msg`	the message that should be printed.
`...`	any additional text that should be printed.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.info('PEcAn version 1.2')

## End(Not run)
```

`logger.message` *Prints a message at a certain log level.*

Description

This function will print a message. This is the function that is responsible for the actual printing of the message.

Usage

```
logger.message(level, msg, ..., wrap = TRUE)
```

Arguments

level	the level of the message (DEBUG, INFO, WARN, ERROR)
msg	the message that should be printed.
...	any additional text that should be printed.
wrap	Whether or not to wrap long messages (default = TRUE). If FALSE, preserve format of original string. Useful for specifically formatted error messages.

Details

This is a place holder and will be later filled in with a more complex logging set

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.message('DEBUG', 'variable', 5)

## End(Not run)
```

---

logger.setLevel *Configure logging level.*

Description

This will configure the logger level. This allows to turn DEBUG, INFO, WARN and ERROR messages on and off.

Usage

```
logger.setLevel(level)
```

Arguments

level	the level of the message (ALL, DEBUG, INFO, WARN, ERROR, OFF)

Value

When logger level is set, the previous level is returned invisibly. This can be passed to `logger.setLevel()` to restore the previous level.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.setLevel('DEBUG')

## End(Not run)
```

`logger.setOutputFile` *Configure logging output filename.*

Description

The name of the file where the logging information should be written to.

Usage

```
logger.setOutputFile(filename)
```

Arguments

`filename`	the file to send the log messages to (or NA to not write to file)

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.setOutputFile('pecan.log')

## End(Not run)
```

`logger.setQuitOnSevere`
Configure whether severe should quit.

Description

The default is for a non-interactive session to quit. Setting this to false is especially useful for running tests when placed in `inst/tests/test.<fn>.R`, but is not passed from `tests/run.all.R`.

Usage

```
logger.setQuitOnSevere(severeQuits)
```

Arguments

severeQuits	should R quit on a severe error.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.setQuitOnSevere(FALSE)

## End(Not run)
```

---

logger.setUseConsole *Configure logging to console.*

Description

Should the logging to be printed to the console or not.

Usage

```
logger.setUseConsole(console, stderr = TRUE)
```

Arguments

console	set to true to print logging to console.
stderr	set to true (default) to use stderr instead of stdout for logging

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.setUseConsole(TRUE)

## End(Not run)
```

logger.setWidth *Configure the number of chars per line*

Description

The default is for 60 chars per line. Setting this to any value will wrap the line when printing a message at that many chars.

Usage

```
logger.setWidth(width)
```

Arguments

width	number of chars to print before wrapping to next line.

Author(s)

David LeBauer

Examples

```
## Not run:
logger.setWidth(70)

## End(Not run)
```

logger.severe *Prints an severe message and stops execution.*

Description

This function will print a message and stop execution of the code. This should only be used if the application should terminate.

Usage

```
logger.severe(msg, ..., wrap = TRUE)
```

Arguments

msg	the message that should be printed.
...	any additional text that should be printed.
wrap	Whether or not to wrap long messages (default = TRUE). If FALSE, preserve format of original string. Useful for specifically formatted error messages.

Details

set `logger.setQuitOnSevere(FALSE)` to avoid terminating the session. This is set by default to TRUE if interactive or running inside Rstudio.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.severe('missing parameters')

## End(Not run)
```

`logger.warn` *Prints a warning message.*

Description

This function will print a warning message.

Usage

```
logger.warn(msg, ...)
```

Arguments

`msg`	the message that should be printed.
`...`	any additional text that should be printed.

Author(s)

Rob Kooper

Examples

```
## Not run:
logger.warn('detected NA values')

## End(Not run)
```

`print2string` *Convert the printed output of an object to a string. This is particularly useful for passing complex objects (e.g. matrices, data.frames) to simple printing functions like* `logger.info()` *or* `base::message()`.

Description

Note that for this to work properly in the `PEcAn.logger` functions, you should always add the `wrap = FALSE` argument, and probably add a newline (`"\n"`) before the output of this function.

Usage

```
print2string(x, ...)
```

Arguments

`x`	Object to print
`...`	Additional arguments to `base::print()`.

Value

Output of `print(x, ...)`, captured as string

Author(s)

Alexey Shiklomanov

Examples

```
logger.info("First few rows of Iris:\n", print2string(iris[1:10, -5]), wrap = FALSE)
df <- data.frame(test = c("download", "process", "plot"), status = c(TRUE, TRUE, FALSE))
logger.debug("Current status:\n", print2string(df, row.names = FALSE), wrap = FALSE)
```

`severeifnot` *Logger message if conditions are not met*

Description

Similar to base::stopifnot, but allows you to use a custom message and logger level. If all conditions are `TRUE`, silently exit.

Usage

```
severeifnot(msg, ...)

errorifnot(msg, ...)

warnifnot(msg, ...)

infoifnot(msg, ...)

debugifnot(msg, ...)
```

Arguments

msg	Logger message to write, as a single character string.
...	Conditions to evaluate

Details

Conditions can be vectorized, or can return non-logical values.The underlying function automatically applies `isTRUE(all(.))` to the conditions.

Value

Invisibly, `TRUE` if conditions are met, `FALSE` otherwise

Examples

```
a <- 1:5
b <- list(6, 7, 8)
debugifnot("By the way, something is not a list.", is.list(a), is.list(b))
infoifnot("Something is not a list.", is.list(a), is.list(b))
warnifnot("I would prefer it if you used lists.", is.list(a), is.list(b))
errorifnot("You should definitely use lists.", is.list(a), is.list(b))
try({
  severeifnot("I cannot deal with the fact that something is not a list.",
    is.list(a),
    is.list(b))
})
```

Index