Package 'stenographer'

January 16, 2025

Type Package

Title Flexible and Customisable Logging System

Version 1.0.0

URL https://github.com/dereckmezquita/stenographer

BugReports https://github.com/dereckmezquita/stenographer/issues

Maintainer Dereck Mezquita <dereck@mezquita.io>

Description
A comprehensive logging framework for R applications that provides hierarchical logging levels, database integration, and contextual logging capabilities. The package supports 'SQLite' storage for persistent logs, provides colour-coded console output for better readability, includes parallel processing support, and implements structured error reporting with 'JSON' formatting.

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports R6, rlang, jsonlite, fs, crayon, DBI

Suggests testthat (>= 3.0.0), diffviewer, knitr, rmarkdown, box, future, future.apply, RSQLite

Config/testthat/edition 3

NeedsCompilation no

Author Dereck Mezquita [aut, cre] (<https://orcid.org/0000-0002-9307-6762>)

Repository CRAN

Date/Publication 2025-01-16 10:50:06 UTC

Contents

`collapse` *Concatenate Vector Elements with Optional Separator*

Description

Concatenates vector elements into a single string. Unlike ‘paste0‘, it handles single-element vectors without adding a trailing separator.

Usage

```
collapse(vector, collapse = " ")
```

Arguments

`vector`	A character vector to be concatenated
`collapse`	String to use as separator between elements (default: " ")

Value

A character string containing the concatenated elements

Examples

```
# Multiple elements
collapse(c("a", "b", "c"), ", ")  # Returns "a, b, c"

# Single element - no trailing separator
collapse("a", ", ")  # Returns "a"

# With default separator
collapse(c("Hello", "World"))  # Returns "Hello World"

# Empty vector
collapse(character(0), ", ")  # Returns character(0)
```

LogLevel *Logging Level*

Description

Defines standard logging levels for controlling message output granularity. Use as a configuration for the ‘Stenographer‘ class to control which messages are logged.

A list with four integer elements:

OFF (-1) Disables all logging

ERROR (0) Logs only errors

WARNING (1) Logs errors and warnings

INFO (2) Logs all messages

Usage

```
LogLevel
```

Format

An object of class `list` of length 4.

Examples

```
# Check logging levels
LogLevel$OFF     # -1
LogLevel$ERROR   # 0
LogLevel$WARNING # 1
LogLevel$INFO    # 2
```

messageParallel *Print Messages from Parallel Processes*

Description

Enables message output from forked processes during parallel computation using the system's echo command. Primarily designed for use with ‘parallel‘ ‘future‘ and ‘future.apply‘ parallel processing.

Usage

```
messageParallel(...)
```

Arguments

...	Arguments to be concatenated into a single character string for printing

Value

Invisible NULL, called for its side effect of printing

Note

This function may have significant resource overhead when used frequently or with large amounts of output. Use sparingly in performance-critical code.

Examples

```
# Basic usage
messageParallel("Hello World")

# Multiple arguments are concatenated
messageParallel("Hello", " ", "World")


if (requireNamespace("future", quietly = TRUE)) {
  future::plan(future::multisession)
  f <- future::future({
    messageParallel("Message from parallel process")
  })
  future::value(f)
  future::plan(future::sequential)
}
```

Stenographer *R6 Class for Advanced Logging Functionality*

Description

Provides a flexible logging system with support for multiple output destinations, customisable formatting, and contextual logging. Features include:

* Multiple log levels (ERROR, WARNING, INFO) * File-based logging * Database logging support * Customisable message formatting * Contextual data attachment * Coloured console output

Methods

Public methods:

- `Stenographer$new()`
- `Stenographer$set_level()`
- `Stenographer$update_context()`
- `Stenographer$clear_context()`
- `Stenographer$get_context()`
- `Stenographer$error()`
- `Stenographer$warn()`

- Stenographer$info()
- Stenographer$clone()

Method `new()`**:** Create a new Stenographer instance

Usage:

```
Stenographer$new(
  level = LogLevel$INFO,
  file_path = NULL,
  db_conn = NULL,
  table_name = "LOGS",
  print_fn = function(x) cat(x, "\n"),
  format_fn = function(level, msg) msg,
  context = list()
)
```

Arguments:

`level` The minimum log level to output. Default is LogLevel$INFO.

`file_path` Character; the path to a file to save log entries to. Default is NULL.

`db_conn` DBI connection object; an existing database connection. Default is NULL.

`table_name` Character; the name of the table to log to in the database. Default is "LOGS".

`print_fn` Function; custom print function to use for console output. Should accept a single character string as input. Default uses cat with a newline.

`format_fn` Function; custom format function to modify the log message. Should accept level and msg as inputs and return a formatted string.

`context` List; initial context for the logger. Default is an empty list.

Returns: A new 'Stenographer' object.

Method `set_level()`**:** Update the minimum logging level

Usage:

```
Stenographer$set_level(level)
```

Arguments:

`level` New log level (see 'LogLevel')

Method `update_context()`**:** Add or update contextual data

Usage:

```
Stenographer$update_context(new_context)
```

Arguments:

`new_context` List of context key-value pairs

Method `clear_context()`**:** Remove all contextual data

Usage:

```
Stenographer$clear_context()
```

Method `get_context()`**:** Retrieve current context data

Usage:

```
Stenographer$get_context()
```

Returns: List of current context

Method `error()`**:** Log an error message

Usage:

```
Stenographer$error(msg, data = NULL, error = NULL)
```

Arguments:

`msg` Error message text

`data` Optional data to attach

`error` Optional error object

Method `warn()`**:** Log a warning message

Usage:

```
Stenographer$warn(msg, data = NULL)
```

Arguments:

`msg` Warning message text

`data` Optional data to attach

Method `info()`**:** Log an informational message

Usage:

```
Stenographer$info(msg, data = NULL)
```

Arguments:

`msg` Info message text

`data` Optional data to attach

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Stenographer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a basic Stenographer
steno <- Stenographer$new()
steno$info("This is an info message")
steno$warn("This is a warning")
steno$error("This is an error")

# Disable all logging
steno$set_level(LogLevel$OFF)
steno$info("This won't be logged")
steno$warn("This won't be logged either")
steno$error("This also won't be logged")
```

```
# Create a logger with custom settings, message formatting, and context
custom_steno <- Stenographer$new(
  level = LogLevel$WARNING,
  file_path = tempfile("log_"),
  print_fn = function(x) message(paste0("Custom: ", x)),
  format_fn = function(level, msg) paste0("Hello prefix: ", msg),
  context = list(program = "MyApp")
)
custom_steno$info("This won't be logged")
custom_steno$warn("This will be logged with a custom prefix")

# Change log level and update context
custom_steno$set_level(LogLevel$INFO)
custom_steno$update_context(list(user = "John"))
custom_steno$info("Now this will be logged with a custom prefix and context")
```

`tableToString` *Convert a Data Frame or R Object to a String Representation*

Description

Captures the printed output of a data.frame or an R object (coerced to a data.frame) as a single string with preserved formatting. Useful for error messages, logging, and string-based output.

Usage

```
tableToString(obj)
```

Arguments

`obj`	An R object that can be coerced to a data.frame

Value

A character string containing the formatted table output with newlines

Examples

```
# Basic usage with a data.frame
df <- data.frame(
  numbers = 1:3,
  letters = c("a", "b", "c")
)
str_output <- tableToString(df)
cat(str_output)

# Using in error messages
df <- data.frame(value = c(10, 20, 30))
if (any(df$value > 25)) {
```

```
  msg <- sprintf(
    "Values exceed threshold:\n%s",
    tableToString(df)
  )
  message(msg)
}
```

`valueCoordinates` *Locate Specific Values in a Data Frame*

Description

Finds the positions (row and column indices) of values in a data.frame that match specified criteria. This function is useful for locating particular values within large datasets.

Usage

```
valueCoordinates(df, value = NA, eq_fun = value_check)
```

Arguments

`df`	A data.frame to search
`value`	The target value to find (default: NA)
`eq_fun`	A comparison function that takes two parameters: the current value from the data.frame and the target value. Returns TRUE for matches. Default uses internal value_check function; handles NA values.

Value

A data.frame with two columns:

column Column indices where matches were found

row Row indices where matches were found

Results are sorted by column, then by row.

Examples

```
# Sample data.frame
df <- data.frame(
  a = c(1, NA, 3),
  b = c(NA, 2, NA),
  c = c(3, 2, 1)
)

# Find NA positions
valueCoordinates(df)
```

```
# Find positions of value 2
valueCoordinates(df, 2)

# Find positions where values exceed 2
valueCoordinates(df, 2, function(x, y) x > y)

# Find positions of values in range [1,3]
valueCoordinates(df, c(1, 3), function(x, y) x >= y[1] & x <= y[2])
```

Index