Package: msgr (via r-universe)

September 16, 2024

Title Extends Messages, Warnings and Errors by Adding Levels and Log Files

Version 1.1.2

Description Provides new functions info(), warn() and error(), similar to message(), warning() and stop() respectively. However, the new functions can have a 'level' associated with them, so that when executed the global level option determines whether they are shown or not. This allows debug modes, outputting more information. The can also output all messages to a log file.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Language en-GB

RoxygenNote 7.0.2

Imports rlang, purrr

Suggests testthat, covr

URL https://github.com/ChadGoymer/msgr

BugReports https://github.com/ChadGoymer/msgr/issues

NeedsCompilation no

Author Chad Goymer [aut, cre]

Maintainer Chad Goymer <chad.goymer@gmail.com>

Date/Publication 2019-12-16 11:40:02 UTC

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

RemoteUrl https://github.com/cran/msgr

RemoteRef HEAD

RemoteSha ef5c881f78f943674f308ef4128ca14627a8350a

Contents

`assert` *Display an error, and record in a log file, if a condition is false*

Description

This function calls the `error()` function to display an error if the specified condition is false. If a message is not specified then a generic message is displayed.

Usage

```
assert(
  condition,
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

condition	(boolean) The condition to check.
...	(strings) message to be displayed or written to file.
level	(integer, optional) The level of the message, from 1 to 10. Default: 1.
msg_level	(integer, optional) The maximum level of messages to output. Default: set in the option "msgr.level".
msg_types	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option "msgr.types".
log_path	(string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option "msgr.log_path".

Value

A string is return invisibly containing the error

Examples

```
## Not run:
  # Use assert() to create conditional timed errors
  x <- 1
  assert(x > 0, "Condition is true so this error is not shown")
  assert(x < 0, "Condition is false so this error is shown")

  # As with error() a level can be set
  assert(x < 0, "This is a level 2 error, so not shown by default", level = 2)

  # Set default level in options to determine what is shown
  options(msgr.level = 2)
  assert(x < 0, "This is a level 2 error, so is shown now", level = 2)

## End(Not run)
```

---

error *Display an error, and record it in a log file.*

Description

error() is similar to stop(), but it also writes the error to a log file. Whether it is shown, or written to the log, depends on the level and type of the error. See details below for more information.

Usage

```
error(
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

`...`	(strings) error to be displayed or written to file.
`level`	(integer, optional) The level of the error, from 1 to 10. Default: 1.
`msg_level`	(integer, optional) The maximum level of messages to output. Default: set in the option `"msgr.level"`.
`msg_types`	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option `"msgr.types"`.
`log_path`	(string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option `"msgr.log_path"`.

Details

Whether an error is shown, or written to the log, depends on two options:

1. **Level**: This allows control over the depth of messages. Each message can be assigned a `level` and if it is below the `msg_level` (set in the package option `msgr.level` by default) the message is displayed and written to the log.
2. **Type**: The type is refers to whether the message is "INFO", "WARNING" or "ERROR", as produced by the functions `info()`, `warn()` and `error()` respectively. If the message type is in the `msg_types` (set in the package option `msgr.types` by default) the message is displayed and written to the log. This allows you to for instance, just display errors and warnings and ignore messages.

The location of the log file is set in the package option `msgr.log_path`, or as an argument to this function. messages are added with a time stamp. If the `log_path` is equal to "" then no log is produced.

Value

A string is return invisibly containing the error

Examples

```
## Not run:
  # Use error() to create timed errors
  error("This is a simple error")
  error("This is a level 2 error, so not shown by default", level = 2)
```

```
  # Set default level in options to determine what is shown
  options(msgr.level = 2)
  error("This is a level 2 error, so is shown now", level = 2)

## End(Not run)
```

`error_if` *Display an error, and record in a log file, if a condition is true.*

Description

This function calls the `error()` function to display an error if the specified condition is true. If a message is not specified then a generic message is displayed.

Usage

```
error_if(
  condition,
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

`condition`	(boolean) The condition to check.
`...`	(strings) message to be displayed or written to file.
`level`	(integer, optional) The level of the message, from 1 to 10. Default: 1.
`msg_level`	(integer, optional) The maximum level of messages to output. Default: set in the option `"msgr.level"`.
`msg_types`	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option `"msgr.types"`.
`log_path`	(string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option `"msgr.log_path"`.

Value

A string is return invisibly containing the error

Examples

```
## Not run:
  # Use error_if() to create conditional timed errors
  error_if(2 > 1, "Condition is true so this error is shown")
  error_if(1 > 2, "Condition is false so this error is not shown")

  # As with error() a level can be set
  error_if(2 > 1, "This is a level 2 error, so not shown by default", level = 2)

  # Set default level in options to determine what is shown
  options(msgr.level = 2)
  error_if(2 > 1, "This is a level 2 error, so is shown now", level = 2)

## End(Not run)
```

---

has_names *Checks whether the variable has names*

Description

Checks whether the variable has names

Usage

```
has_names(x, nm)
```

Arguments

x	(any) The object to test
nm	(character, optional) The names to check for. If not specified then the function checks for any names.

Value

TRUE if x has any names, FALSE otherwise

Examples

```
x <- list(a = 1, b = 2)

has_names(x, "a")
has_names(x, c("a", "b"))

has_names(x, "c")
```

`info` *Display a message, and record it in a log file.*

Description

`info()` is similar to `message()`, but it also writes the message to a log file. Whether it is shown, or written to the log, depends on the level and type of the message. See details below for more information.

Usage

```
info(
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

`...`	(strings) message to be displayed or written to file.
`level`	(integer, optional) The level of the message, from 1 to 10. Default: 1.
`msg_level`	(integer, optional) The maximum level of messages to output. Default: set in the option `"msgr.level"`.
`msg_types`	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option `"msgr.types"`.
`log_path`	(string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option `"msgr.log_path"`.

Details

Whether a message is shown, or written to the log, depends on two options:

1. **Level**: This allows control over the depth of messages. Each message can be assigned a `level` and if it is below the `msg_level` (set in the package option `msgr.level` by default) the message is displayed and written to the log.
2. **Type**: The type is refers to whether the message is "INFO", "WARNING" or "ERROR", as produced by the functions `info()`, `warn()` and `error()` respectively. If the message type is in the `msg_types` (set in the package option `msgr.types` by default) the message is displayed and written to the log. This allows you to for instance, just display errors and warnings and ignore messages.

The location of the log file is set in the package option `msgr.log_path`, or as an argument to this function. messages are added with a time stamp. If the `log_path` is equal to "" then no log is produced.

Value

A string is return invisibly containing the message.

Examples

```
# Use info() to create timed messages
info("This is a simple message")
info("This is a level 2 message, so not shown by default", level = 2)

# Set default level in options to determine what is shown
options(msgr.level = 2)
info("This is a level 2 message, so is shown now", level = 2)

# Set message types in options to determine what is shown
options(msgr.types = c("WARNING", "ERROR"))
info("This is message, so will not be shown now")
```

`info_if` *Display a message, and record in a log file, if a condition is true.*

Description

This function calls the `info()` function to display a message if the specified condition is true. If a message is not specified then a generic message is displayed.

Usage

```
info_if(
  condition,
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

`condition`	(boolean) The condition to check.
`...`	(strings) message to be displayed or written to file.
`level`	(integer, optional) The level of the message, from 1 to 10. Default: 1.
`msg_level`	(integer, optional) The maximum level of messages to output. Default: set in the option `"msgr.level"`.
`msg_types`	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option `"msgr.types"`.

log_path (string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option "msgr.log_path".

Value

A string is return invisibly containing the message.

Examples

```
# Use info_if() to create conditional timed messages
info_if(2 > 1, "Condition is true so this message is shown")
info_if(1 > 2, "Condition is false so this message is not shown")

# As with info() a level can be set
info_if(2 > 1, "This is a level 2 message, so not shown by default", level = 2)

# Set default level in options to determine what is shown
options(msgr.level = 2)
info_if(2 > 1, "This is a level 2 message, so is shown now", level = 2)

# Set message types in options to determine what is shown
options(msgr.types = c("WARNING", "ERROR"))
info_if(2 > 1, "This is message, so will not be shown now")
```

---

is_dir *Checks whether the variable is a path to an existing directory*

Description

Checks whether the variable is a path to an existing directory

Usage

```
is_dir(x)
```

Arguments

x (any) The object to test

Value

TRUE if x is a path to an existing directory, FALSE otherwise

Examples

```
is_dir(tempdir())
is_dir("/does/not/exist")
is_dir(1)
```

is_file *Checks whether the variable is a path to an existing file*

Description

Checks whether the variable is a path to an existing file

Usage

```
is_file(x)
```

Arguments

x	(any) The object to test

Value

TRUE if x is a path to an existing file, FALSE otherwise

Examples

```
tmpfile <- tempfile()
file.create(tmpfile)

is_file(tmpfile)

is_file("/does/not/exist.txt")
is_file(1)
```

is_in *Checks whether all elements of one variable are in another*

Description

Checks whether all elements of one variable are in another

Usage

```
is_in(x, y)
```

Arguments

x	(any) The object with elements to test
y	(any) The object with elements to test against

Value

TRUE if all elements in x are in y, FALSE otherwise

Examples

```
is_in("a", letters)
is_in(c("a", "b", "c"), letters)

is_in(1, LETTERS)
is_in(1:2, LETTERS)
```

---

is_na *Checks whether the variable is NA*

Description

Checks whether the variable is NA

Usage

```
is_na(x)
```

Arguments

x	(any) The object to test

Value

TRUE if x is NA, FALSE otherwise

Examples

```
is_na(1)
is_na("foo")
is_na(NA)

is_na(c(1, NA))
is_na(c(NA, NA))
```

is_readable — *Checks whether the variable is a path to an existing, readable file or directory*

Description

Checks whether the variable is a path to an existing, readable file or directory

Usage

```
is_readable(x)
```

Arguments

x	(any) The object to test

Value

TRUE if x is a path to an existing, readable file or directory, FALSE otherwise

Examples

```
tmpfile <- tempfile()
file.create(tmpfile)

is_readable(tmpfile)

is_readable("/does/not/exist.txt")
is_readable(1)
```

is_url — *Checks whether the variable is a valid URL*

Description

Checks whether the variable is a valid URL

Usage

```
is_url(x)
```

Arguments

x	(any) The object to test

Value

TRUE if x is a valid URL, FALSE otherwise

Examples

```
is_url("http://something.com")
is_url("https://google.com")

is_url(1)
is_url("foo")
is_url(NA)
```

`is_writeable` *Checks whether the variable is a path to an existing, writeable file or directory*

Description

Checks whether the variable is a path to an existing, writeable file or directory

Usage

```
is_writeable(x)
```

Arguments

x	(any) The object to test

Value

TRUE if x is a path to an existing, writeable file or directory, FALSE otherwise

Examples

```
tmpfile <- tempfile()
file.create(tmpfile)

is_writeable(tmpfile)

is_writeable("/does/not/exist.txt")
is_writeable(1)
```

msgr *Extends messages, warnings and errors by adding levels and log files*

Description

Provides new functions `info()`, `warn()` and `error()`, similar to `message()`, `warning()` and `stop()` respectively. However, the new functions can have a `level` associated with them, so that when executed the global level option determines whether they are shown or not. This allows debug modes, outputting more information. The can also output all messages to a log file.

Author(s)

Chad Goymer <chad.goymer@gmail.com>

See Also

Useful links:

- https://github.com/ChadGoymer/msgr
- Report bugs at https://github.com/ChadGoymer/msgr/issues

try_catch *Try to evaluate an expressions and capture any messages, warnings or errors*

Description

This function is similar to `tryCatch()`, except that, by default, errors are captured and presented using `error()`. Messages and warnings are not captured by this function. In addition, a "finally" expression can be specified which is evaluated at the end of the call no matter the result.

Usage

```
try_catch(expr, on_error, finally)
```

Arguments

`expr`	(expression) The expression to evaluate
`on_error`	(function, optional) A function describing what to do in the event of a error in the above expression. The function must take a single argument, which is the `simpleError()`. If missing or `NULL`, errors are not caught. Default: `error()` called with the error's message prefixed by the calling function name.
`finally`	(expression, optional) An expression to evaluate at the end of the call. If missing or `NULL`, nothing is actioned.

Value

The result of the evaluated expression

Examples

```
## Not run:
  try_catch(x <- "foo")
  try_catch(stop("This is an error"))

## End(Not run)
```

`try_map` *Apply a function over a vector or list, capturing any errors to display at the end*

Description

This function is similar to `purrr::map()` except that instead of stopping at the first error it captures them and continues. If there are any errors it collects them together and displays them at the end. You have the option to specify a prefix to the error message using the `msg_prefix` argument.

Usage

```
try_map(
  x,
  f,
  ...,
  msg_prefix,
  warn_level = 2,
  error_level = 1,
  on_error = "error",
  simplify = FALSE,
  use_names = TRUE
)
```

Arguments

`x`	(vector or list) The vector or list to map the function to.
`f`	(function) The function to map to the elements of `x`.
`...`	(optional) Extra arguments to supply to f.
`msg_prefix`	(string, optional) A message to prefix any resulting error message.
`warn_level`	(integer, optional) The level of any warnings about errors encountered. If 0 then no warnings are shown. Default: 2.
`error_level`	(integer, optional) The level of any resulting errors. Default: 1.

on_error	(string) The kind of message to produce if there is an error. Either "info", "warn", or "error". Default: "error".
simplify	(boolean, optional) Whether to try to simplify the result of the mapping into an atomic vector. Default: FALSE.
use_names	(boolean, optional) Whether to use 'x' as names in the resulting list. 'x' must be a character vector for this to work. Default: TRUE.

Details

If the mapped function is a long running process `try_map()` can output a warning at the time an error occurs, but specifying the `warn_level` argument to be greater than 0 (see `warn()` for more details about message levels. Similarly `error_level` argument specifies the level of any reported error, as detailed in `error()`.

If you do not want the function to stop with an error, you can instead return a warning or info message using the `on_error` argument.

Finally, `simplify` and `use_names` allow the user to specify whether to simplify the output to an atomic vector, if possible, and whether to use the vector input `x` as names to the resulting list.

Value

If `simplify = FALSE` a list is returned. Otherwise, the function attempts to simplify the result to an atomic vector or array.

Examples

```
## Not run:
  test_try_map <- function(x, y) if (x > y) stop("x > y") else x
  try_map(1:3, test_try_map, y = 2)
  try_map(1:3, test_try_map, y = 5)

## End(Not run)
```

---

try_pmap *Apply a function over a list of vectors, capturing any errors to display at the end*

Description

This function is similar to `purrr::pmap()` except that instead of stopping at the first error it captures them and continues. If there are any errors it collects them together and displays them at the end. You have the option to specify a prefix to the error message using the `msg_prefix` argument.

Usage

```
try_pmap(
  l,
  f,
  ...,
  msg_prefix,
  warn_level = 2,
  error_level = 1,
  on_error = "error",
  simplify = FALSE,
  use_names = TRUE
)
```

Arguments

l	(list) A list of vectors the same length to apply the function to.
f	(function) The function to map to the elements of the vectors in l.
...	(optional) Extra arguments to supply to f.
msg_prefix	(string, optional) A message to prefix any resulting error message.
warn_level	(integer, optional) The level of any warnings about errors encountered. If 0 then no warnings are shown. Default: 2.
error_level	(integer, optional) The level of any resulting errors. Default: 1.
on_error	(string) The kind of message to produce if there is an error. Either "info", "warn", or "error". Default: "error".
simplify	(boolean, optional) Whether to try to simplify the result of the mapping into an atomic vector. Default: FALSE.
use_names	(boolean, optional) Whether to use 'x' as names in the resulting list. 'x' must be a character vector for this to work. Default: TRUE.

Details

If the mapped function is a long running process try_pmap can output a warning at the time an error occurs, but specifying the warn_level argument to be greater than 0 (see warn() for more details about message levels. Similarly error_level argument specifies the level of any reported error, as detailed in error().

If you do not want the function to stop with an error, you can instead return a warning or info message using the on_error argument.

Finally, simplify and use_names allow the user to specify whether to simplify the output to an atomic vector, if possible, and whether to use the vector input x as names to the resulting list.

Value

If simplify = FALSE a list is returned. Otherwise, the function attempts to simplify the result to an atomic vector.

Examples

```
## Not run:
  test_try_pmap <- function(x, y) if (x > y) stop("x > y") else x
  try_pmap(list(1:3, 3:1), test_try_pmap)
  try_pmap(list(1:3, 2:4), test_try_pmap)

## End(Not run)
```

`warn` *Display a warning, and record it in a log file.*

Description

`warn()` is similar to `warning()`, but it also writes the warning to a log file. Whether it is shown, or written to the log, depends on the level and type of the warning. See details below for more information.

Usage

```
warn(
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

`...`	(strings) warning to be displayed or written to file.
`level`	(integer, optional) The level of the warning, from 1 to 10. Default: 1.
`msg_level`	(integer, optional) The maximum level of messages to output. Default: set in the option `"msgr.level"`.
`msg_types`	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option `"msgr.types"`.
`log_path`	(string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option `"msgr.log_path"`.

Details

Whether a warning is shown, or written to the log, depends on two options:

1. **Level**: This allows control over the depth of messages. Each message can be assigned a `level` and if it is below the `msg_level` (set in the package option `msgr.level` by default) the message is displayed and written to the log.

2. **Type**: The type is refers to whether the message is "INFO", "WARNING" or "ERROR", as produced by the functions `info()`, `warn()` and `error()` respectively. If the message type is in the `msg_types` (set in the package option `msgr.types` by default) the message is displayed and written to the log. This allows you to for instance, just display errors and warnings and ignore messages.

The location of the log file is set in the package option `msgr.log_path`, or as an argument to this function. messages are added with a time stamp. If the `log_path` is equal to "" then no log is produced.

Value

A string is return invisibly containing the warning

Examples

```
# Use warn() to create timed warnings
warn("This is a simple warning")
warn("This is a level 2 warning, so not shown by default", level = 2)

# Set default level in options to determine what is shown
options(msgr.level = 2)
warn("This is a level 2 warning, so is shown now", level = 2)

# Set message types in options to determine what is shown
options(msgr.types = "ERROR")
warn("This is warning, so will not be shown now")
```

---

warn_if *Display a warning, and record in a log file, if a condition is true.*

Description

This function calls the `warn()` function to display a warning if the specified condition is true. If a message is not specified then a generic message is displayed.

Usage

```
warn_if(
  condition,
  ...,
  level = 1,
  msg_level = getOption("msgr.level"),
  msg_types = getOption("msgr.types"),
  log_path = getOption("msgr.log_path")
)
```

Arguments

condition	(boolean) The condition to check.
...	(strings) message to be displayed or written to file.
level	(integer, optional) The level of the message, from 1 to 10. Default: 1.
msg_level	(integer, optional) The maximum level of messages to output. Default: set in the option "msgr.level".
msg_types	(character, optional) The type to write or display. Must either NULL or one or more from "INFO", "WARNING" or "ERROR". Default: set in the option "msgr.types".
log_path	(string, optional) The file path to the text log file. If set to "", then no logs are written. Default: set in the option "msgr.log_path".

Value

A string is return invisibly containing the warning.

Examples

```
# Use warn_if() to create conditional timed warnings
warn_if(2 > 1, "Condition is true so this warning is shown")
warn_if(1 > 2, "Condition is false so this warning is not shown")

# As with warn() a level can be set
warn_if(2 > 1, "This is a level 2 warning, so not shown by default", level = 2)

# Set default level in options to determine what is shown
options(msgr.level = 2)
warn_if(2 > 1, "This is a level 2 warning, so is shown now", level = 2)

# Set message types in options to determine what is shown
options(msgr.types = "ERROR")
warn_if(2 > 1, "This is warning, so will not be shown now")
```

Index