Package: messaging (via r-universe)

August 21, 2024

Type Package

Title Conveniently Issue Messages, Warnings, and Errors

Version 0.1.0

Description Provides tools for creating and issuing nicely-formatted text within R diagnostic messages and those messages given during warnings and errors. The formatting of the messages can be customized using templating features. Issues with singular and plural forms can be handled through specialized syntax.

License MIT + file LICENSE

URL https://github.com/rich-iannone/messaging

BugReports https://github.com/rich-iannone/messaging/issues

Depends R (>= 3.1.2)

Imports dplyr (>= 0.7.4), glue (>= 1.2.0), magrittr, rlang (>= 0.2.0), stringr (>= 1.3.0)

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 6.0.1

Suggests testthat, covr

Repository https://rich-iannone.r-universe.dev

RemoteUrl https://github.com/rich-iannone/messaging

RemoteRef HEAD

RemoteSha 2d94d0b1f4e978ee1050a32523de3a0b6bacc87a

Contents

`emit_error` *Stop function and provide associated messages*

Description

Stop the execution of the function in the parent environment and use a message body that can be formed from multiple text objects. If a single text object is provided then the warning message text will appear on a single line. If multiple text fragments are provided then they will be separated by newlines (in the order provided). Custom formatting is possible by providing a messaging template that uses the string expression scheme used in the `glue` package.

Usage

```
emit_error(..., .format = NULL, .f_name = TRUE, .issue = TRUE)
```

Arguments

`...`	a collection of string expressions and named arguments for string interpolation. Bare strings will be pasted together and separated by newlines. Named arguments are used to provide values for string interpolation in the message.
`.format`	a format template for a message. If not provided, the default template of `"{.f_name}: {text}"` will be used.
`.f_name`	the name of the function that caused the error. If not provided, the function name will be obtained from the function call stack.
`.issue`	a logical value that indicates whether an error should be issued at all.

Examples

```
## Not run:
# Write a function that stops the function
# with a message with the requested number
# of info lines
yield_an_error <- function(msgs) {

  if (msgs > 3) msgs <- 3

  # Create some strings can serve as additional
  # info for the message
  message_components <-
    c("* message info 1",
      "* message info 2",
      "* message info 3")

  # Generate and emit a formatted message
  emit_error(
    "There (is/are) {number} thing(s) to note",
    message_components[1:msgs],
    number = msgs,
```

```
    .format = "{.f_name} info: {text}")
}

# When that function is called, a formatted
# message will appear; here are some examples:
yield_an_error(msgs = 3)
#> Error: `yield_an_error()` info: There are 3 things to note
#> * message info 1
#> * message info 2
#> * message info 3

yield_an_error(msgs = 2)
#> Error: `yield_an_error()` info: There are 2 things to note
#> * message info 1
#> * message info 2

yield_an_error(msgs = 1)
#> Error: `yield_an_error()` info: There is 1 thing to note
#> * message info 1

## End(Not run)
```

`emit_message` *Provide a message with a consistent format*

Description

Create a message using a message body that can be formed from multiple text objects. If a single text object is provided then the message will appear on a single line. If multiple text fragments are provided then they will be separated by newlines (in the order provided). Custom formatting is possible by providing a messaging template that uses the string expression scheme used in the `glue` package.

Usage

```
emit_message(..., .format = NULL, .f_name = TRUE, .issue = TRUE)
```

Arguments

`...`	a collection of string expressions and named arguments for string interpolation. Bare strings will be pasted together and separated by newlines. Named arguments are used to provide values for string interpolation in the message.
`.format`	a format template for a message. If not provided, the default template of `"{.f_name}: {text}"` will be used.
`.f_name`	the name of the function that relates to the message. If not provided, the function name will be obtained from the function call stack.
`.issue`	a logical value that indicates whether the message should be issued at all. This is to control for the verbosity of messages under certain circumstances.

Examples

```
# Write a function that yields a message with
# the requested number of info lines
yield_a_message <- function(msgs) {

  if (msgs > 3) msgs <- 3

  # Create some strings can serve as additional
  # info for the message
  message_components <-
    c("* message info 1",
      "* message info 2",
      "* message info 3")

  # Generate and emit a formatted message
  emit_message(
    "There (is/are) {number} thing(s) to note",
    message_components[1:msgs],
    number = msgs,
    .format = "{.f_name} info: {text}")
}

# When that function is called, a formatted
# message will appear; here are some examples:
yield_a_message(msgs = 3)

yield_a_message(msgs = 2)

yield_a_message(msgs = 1)
```

`emit_warning` *Provide a warning with a consistent format*

Description

Create a warning using a message body that can be formed from multiple text objects. If a single text object is provided then the warning message text will appear on a single line. If multiple text fragments are provided then they will be separated by newlines (in the order provided). Custom formatting is possible by providing a messaging template that uses the string expression scheme used in the `glue` package.

Usage

```
emit_warning(..., .format = NULL, .f_name = TRUE, .issue = TRUE)
```

Arguments

`...`	a collection of string expressions and named arguments for string interpolation. Bare strings will be pasted together and separated by newlines. Named arguments are used to provide values for string interpolation in the message.

`.format`	a format template for a message. If not provided, the default template of `"{.f_name}: {text}"` will be used.
`.f_name`	the name of the function that caused the warning. If not provided, the function name will be obtained from the function call stack.
`.issue`	a logical value that indicates whether a warning should be issued at all.

Examples

```
## Not run:
# Write a function that yields a warning with
# the requested number of info lines
yield_a_warning <- function(msgs) {

  if (msgs > 3) msgs <- 3

  # Create some strings can serve as additional
  # info for the message
  message_components <-
    c("* message info 1",
      "* message info 2",
      "* message info 3")

  # Generate and emit a formatted message
  emit_warning(
    "There (is/are) {number} thing(s) to note",
    message_components[1:msgs],
    number = msgs,
    .format = "{.f_name} info: {text}")
}

# When that function is called, a formatted
# message will appear; here are some examples:
yield_a_warning(msgs = 3)
#> Warning message:
#> `yield_a_warning()` info: There are 3 things to note
#> * message info 1
#> * message info 2
#> * message info 3

yield_a_warning(msgs = 2)
#> Warning message:
#> `yield_a_warning()` info: There are 2 things to note
#> * message info 1
#> * message info 2

yield_a_warning(msgs = 1)
#> Warning message:
#> `yield_a_warning()` info: There is 1 thing to note
#> * message info 1

## End(Not run)
```

Index