Package: printify (via r-universe)

June 1, 2026

Title Custom Formatted Console Messages with Timing Support

Version 1.0.1

Description A lightweight message system relying purely on base R. Comes with built-in and pre styled message types and provides an easy way to create custom messages. Supports individually styled and colored text as well as timing information. Designed to make console output more informative and visually organized.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

URL https://github.com/s3rdia/printify, https://s3rdia.github.io/printify/, https://deepwiki.com/s3rdia/printify, https://s3rdia.github.io/qol_blog/

Depends R (>= 4.1.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests tinytest (>= 1.4.3)

Config/tinytest/edition 1

Config/Needs/website rmarkdown

Repository https://s3rdia.r-universe.dev

Date/Publication 2026-06-01 21:28:11 UTC

RemoteUrl https://github.com/s3rdia/printify

RemoteRef HEAD

RemoteSha f6ca3ca27263bb48ba1927715806c262e501515a

Contents

`hex_ansi` *Convert Color Codes*

Description

`hex_to_256()`: Generate a 256-color 6x6x6 color cube.

`hex_to_ansi()`: Applies hex color and font weight to a text as ansi codes.

Usage

```
hex_to_256(hex_color)

hex_to_ansi(text, hex_color = NULL, bold = FALSE)
```

Arguments

`hex_color`	The hex color code to convert.
`text`	The text which contains hex color codes to be convert into ansi style formatting.
`bold`	FALSE by default. If TRUE inserts ansi code for bold printing

Value

`hex_to_256()`: Returns 6x6x6 color cube.

`hex_to_ansi()`: Returns formatted text.

Examples

```
color_cube <- hex_to_ansi("#32CD32")

formatted_text <- hex_to_ansi("This is a text to test the coloring",
                              hex_color = "#32CD32", bold = TRUE)
```

message_helpers *Message Helper Functions*

Description

`get_message_stack()`: Retrieves the current message stack as list or data frame.

`set_no_print()`: FALSE by default. If set to TRUE the messages will be formatted and returned but not printed to the console. Can e.g. be used in unit test situations.

`set_no_color()`: TRUE by default. Suppresses the color codes so that messages can be printed clean.

`print_stack_as_messages()`: Prints the message stack as actual messages (only not suppressed messages). Can be used to trigger expect_message, expect_warning or expect_error in unit tests.

`convert_square_brackets()`: Convert different curly bracket patterns into ansi formatting or passes the context of vectors into the text.

Usage

```
get_message_stack(as_data_frame = FALSE)

set_no_print(value = FALSE)

set_no_color(value = TRUE)

print_stack_as_messages(type = NULL)

convert_square_brackets(text, ...)
```

Arguments

as_data_frame	FALSE by default. If TRUE returns message stack information as data frame.
value	Can be TRUE or FALSE.
type	The message type to filter.
text	The text in which to replace the curly brackets.
...	The actual replacement vectors.

Details

The message types in which you can enter custom texts, are capable of using different styling operators. These are:

- Insert list of elements: [vector_name]
- Adding conditional words, if list of elements has more than one element: [?word]
- Adding conditional singular/plural, depending on list of element length: [?singular/plural]
- Bold, italic and underline: [b]some text[/b], [i]some text[/i], [u]some text[/u]
- Coloring parts of the message: [#FF00FF some text]

Value

`get_message_stack()`: Returns a list of messages or a data frame.

`set_no_print()`: Returns the global no_print option.

`set_no_color()`: Returns the global no_color option.

`print_stack_as_messages()`: Returns NULL.

`convert_square_brackets()`: Returns formatted text.

See Also

Main printing functions: `print_message()`, `print_headline()`, `print_start_message()`, `print_closing()`, `print_step()`, `set_up_custom_message()`

---

messages *Print Styled Messages*

Description

Printing styled messages for different occasions. There are notifications, warnings, errors, function call headlines, progress and function closing messages. Or just a neutral one.

Usage

```
print_message(
  type,
  text,
  ...,
  always_print = FALSE,
  utf8 = .printify_messages[["format"]][["utf8"]],
  no_color = .printify_messages[["no_color"]]
)

print_headline(
  text,
  ...,
  line_char = "=",
  max_width = getOption("width"),
  always_print = FALSE,
  utf8 = .printify_messages[["format"]][["utf8"]],
  no_color = .printify_messages[["no_color"]]
)

print_start_message(
  current_time = Sys.time(),
  caller_color = "#63C2C9",
  always_print = FALSE,
```

```
  suppress = FALSE,
  utf8 = .printify_messages[["format"]][["utf8"]],
  no_color = .printify_messages[["no_color"]]
)

print_closing(
  time_threshold = 2,
  start_time = .printify_messages[["start_time"]],
  caller_color = "#63C2C9",
  always_print = FALSE,
  suppress = FALSE,
  utf8 = .printify_messages[["format"]][["utf8"]],
  no_color = .printify_messages[["no_color"]]
)

print_step(
  type,
  text,
  ...,
  in_place = FALSE,
  always_print = FALSE,
  utf8 = .printify_messages[["format"]][["utf8"]],
  no_color = .printify_messages[["no_color"]]
)

set_up_custom_message(
  type = "UNICORN",
  color = "#FF00FF",
  ansi_icon = NULL,
  text_icon = "^",
  ansi_wait_icon = NULL,
  text_wait_icon = "?",
  indent = 1,
  text_bold = FALSE,
  text_italic = FALSE,
  text_underline = FALSE,
  text_color = NULL,
  time_color = "#6B6B6B"
)
```

Arguments

type	If displayed as a normal note, then this is the text displayed in front of the message. This also appears as type in the message stack.
text	The message text to display.
...	Additional information to display like variable names. To use these write [NAME YOU PUT IN] in the text.
always_print	FALSE by default. If TRUE, prints headlines even in deeper nested situations.

utf8	Whether to display complex characters or just plain text.
no_color	Whether to display color codes or suppress them.
line_char	The character that that forms the line.
max_width	The maximum number of characters drawn, which determines the line length of the headline.
current_time	The current time to create a time stamp.
caller_color	Hex color code for the displayed caller function.
suppress	FALSE by default. If TRUE triggers all the message procedures to create a message stack but doesn't print the message to the console.
time_threshold	The total time spent is displayed in different colors from green over yellow to red, depending on the threshold specified (in seconds).
start_time	The time at which the function call started to calculate the time difference and output the total time spent.
in_place	Prints the step message on the same line as before, instead of in the next line. This can e.g. be used inside loops.
color	The color of the message type.
ansi_icon	The icon used when message is displayed in utf8 mode.
text_icon	The icon used when message is displayed in text only mode.
ansi_wait_icon	The icon used when a timed message is waiting for the end of execution displayed in utf8 mode.
text_wait_icon	The icon used when a timed message is waiting for the end of execution displayed in text only mode.
indent	How many spaces to indent the message.
text_bold	FALSE by default. If TRUE prints the message text in bold letters.
text_italic	FALSE by default. If TRUE prints the message text in italic letters.
text_underline	FALSE by default. If TRUE prints the message text underlined.
text_color	The color of the actual message text.
time_color	The color used for the time stamps.

Details

The message types in which you can enter custom texts, are capable of using different styling operators. These are:

- Insert list of elements: [vector_name]
- Adding conditional words, if list of elements has more than one element: [?word]
- Adding conditional singular/plural, depending on list of element length: [?singular/plural]
- Bold, italic and underline: [b]some text[/b], [i]some text[/i], [u]some text[/u]
- Coloring parts of the message: [#FF00FF some text]

Value

Return text without styling or total running time.

`set_up_custom_message()`: Returns the global list of custom messages.

See Also

Also have a look at the small helpers: `get_message_stack()`, `set_no_print()`, `set_no_color()`, `print_stack_as_messages()`, `convert_square_brackets()`

Examples

```
# Example messages
print_message("NOTE", c("Just a quick note that you can also insert e.g.[? a / ]variable",
                        "name[?s] like this: [listing].",
                      "Depending on the number of variables you can also alter the text."),
             listing = c("age", "state", "NUTS3"))

print_message("WARNING", "Just a quick [#FF00FF colored warning]!")

print_message("ERROR", "Or a [b]bold[/b], [i]italic[/i] and [u]underlined[/u] error.")

print_message("NEUTRAL", c("You can also just output [u]plain text[/u] if you like and use",
                       "[#FFFF00 [b]all the different[/b] [i]formatting options.[/i]]"))

# Different headlines
print_headline("This is a headline")

print_headline("[#00FFFF This is a different headline] with some color",
               line_char = "-")

print_headline("[b]This is a very small[/b] and bold headline",
               line_char = ".",
               max_width = 60)

# Messages with time stamps
test_func <- function(){
    print_start_message()
    print_step("GREY", "Probably not so important")
    print_step("MAJOR", "This is a major step...")
    print_step("MINOR", "Sub step1")
    print_step("MINOR", "Sub step2")
    print_step("MINOR", "Sub step3")
    print_step("MAJOR", "[b]Finishing... [/b][#00FFFF with some color again!]")
    print_closing()
}

test_func()

# See what is going on in the message stack
message_stack <- get_message_stack()
```

```
# Print messages on the same line instead of below each other
in_place_steps <- function(){
    print_start_message()

    print_step("MAJOR", "Let's get started...")

    for (i in seq_len(10)){
        print_step("Minor", "This is in place step [i] of 10", i = i, in_place = TRUE)
        Sys.sleep(0.25)
    }

    print_step("MAJOR", "Loop has ended")

    print_closing()
}

in_place_steps()

# Set up a custom message
set_up_custom_message(type           = "HOTDOG",
                      color          = "#B27A01",
                      ansi_icon      = "\U0001f32d",
                      text_icon      = "IOI",
                      ansi_wait_icon = "\U00023f1",
                      text_wait_icon = "/",
                      indent         = 1)

hotdog_print <- function(){
    print_start_message()
    print_message("HOTDOG", c("This is the first hotdog message! Hurray!",
                              "And it is also multiline in this version."))
    print_step("HOTDOG", "Or use as single line message with time stamps.")
    Sys.sleep(0.5)
    print_step("HOTDOG", "Or use as single line message with time stamps.")
    Sys.sleep(0.5)
    print_step("HOTDOG", "Or use as single line message with time stamps.")
    Sys.sleep(0.5)
    print_closing()
}

hotdog_print()

# See new message in the message stack
hotdog_stack <- get_message_stack()
```

Index