

Package ‘dotprofile’

May 8, 2026

Title Create and Manage Configuration Profiles

Version 0.0.1

Description A toolbox to create and manage metadata files and configuration profiles: files used to configure the parameters and initial settings for some computer programs.

License MIT + file LICENSE

URL <https://github.com/jeanmathieupotvin/dotprofile>

BugReports <https://github.com/jeanmathieupotvin/dotprofile/issues/new>

ByteCompile true

Encoding UTF-8

Language en

NeedsCompilation no

RoxygenNote 7.1.2

Depends R (>= 4.1.0)

Imports R6, cli (>= 3.1.0)

Suggests microbenchmark, covr, withr, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel true

Author Jean-Mathieu Potvin [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-8237-422X>>)

Maintainer Jean-Mathieu Potvin <jm@potvin.xyz>

Repository CRAN

Date/Publication 2021-11-23 19:30:04 UTC

Contents

dotprofile-package	2
dotprofile-ui	2
Index	6

dotprofile-package *dotprofile: create and manage configuration profiles*

Description

A toolbox to create and manage metadata files and configuration profiles: files used to configure the parameters and initial settings for some computer programs.

Author(s)

Maintainer: Jean-Mathieu Potvin <jm@potvin.xyz> ([ORCID](#))

See Also

Useful links:

- <https://github.com/jeanmathieupotvin/dotprofile>
- Report bugs at <https://github.com/jeanmathieupotvin/dotprofile/issues/new>

dotprofile-ui *User interface of dotprofile*

Description

These functions are used to construct the command-line interface of **dotprofile**. They are inspired from **usethis** for consistency, but their implementation is different and relies on wrapper functions to other functions of **cli**.

Usage

```
## ---  
## Print messages to the console  
## ---  
  
ui_todo(..., .envir = parent.frame())  
  
ui_info(..., .envir = parent.frame())  
  
ui_done(..., .envir = parent.frame())  
  
ui_nope(..., .envir = parent.frame())  
  
ui_ask(..., .envir = parent.frame())  
  
## ---
```

```

## Ask for inputs interactively
## ---

ui_input(..., .envir = parent.frame())

ui_menu(..., answers = c("yes", "no"), .envir = parent.frame())

## ---
## Signal conditions
## ---

ui_warn(warning, ..., .envir = parent.frame())

ui_stop(error, ..., .envir = parent.frame())

## ---
## Cheatsheet for inline markup
## ---

ui_theme()

```

Arguments

...	[any] Passed to <code>cli::format_inline()</code> , unless stated otherwise below. Use cli inline markup for inline formatting. Functions <code>ui_warn()</code> and <code>ui_stop()</code>: ... is an optional sequence of further calls executed after returning a message via <code>base::warning()</code> and <code>base::stop()</code> respectively. You should only pass calls to other user interface functions. See examples below.
.envir	[environment] Passed to <code>cli::format_inline()</code> , <code>cli::format_warning()</code> or <code>cli::format_error()</code> . This argument is not validated by dotprofile and is reserved for expert use.
answers	[atomic] Non-empty atomic vector passed to argument choices of <code>utils::menu()</code> .
warning	[character(1)] Passed to <code>cli::format_warning()</code> .
error	[character(1)] Passed to <code>cli::format_error()</code> .

Details

The user interface functions of **dotprofile** can be divided into three groups.

1. Print messages with `ui_todo()`, `ui_info()`, `ui_done()`, `ui_warn()`, `ui_ask()` and `ui_nope()`.
2. Seek *answers* from the user with `ui_input()` and `ui_menu()`. The former expects an input to be typed in the terminal, while the latter expects a value to be chosen interactively from a menu. These functions can only be used in an **interactive** session.

- Signal conditions with `ui_warn()` and `ui_stop()`. They are designed to play nicely with `ui_*` functions of the first group, and can be used to craft beautiful and meaningful error messages.

All functions support [string interpolation](#).

Value

- Functions `ui_todo()`, `ui_info()`, `ui_done()`, `ui_nope()`, `ui_ask()`, and `ui_theme()` return NULL invisibly.
- Function `ui_input()` returns a `character(1)`. It returns an empty `character(1)` in non-interactive sessions.
- Function `ui_menu()` returns the *chosen* value taken from argument answers. In non-interactive sessions, or if the user aborts the process, it returns NA. Its actual type matches the type of the value passed to argument answers.
- Function `ui_warn()` throws a warning, but **does not stop** execution of the current expression. It returns NULL invisibly.
- Function `ui_stop()` throws an error message and **stops** execution of the current expression.

Examples

```
## Use dotprofile's UI functions to convey messages to the user.
ui_examples <- function()
{
  ui_todo("This is a {.strong to-be-completed} task.")
  ui_info("This is an information.")
  ui_done("This a {.strong completed} task.")
  ui_nope("Something's {.strong wrong}, but this is not an error.")
  ui_ask("This is a question.")
}

ui_examples()

## Use cli's inline classes to easily style messages passed to `...`
ui_theme()

## Construct beautiful warnings and errors with ui_warn() and ui_stop().
## Not run:
ui_warn("this is a warning message generated with {.fn ui_warn}.")
ui_warn("this is a {.emph super custom} warning message.",
  ui_info("You can pass other {.fn ui_*} calls to it."),
  ui_info("They are printed after the warning message, like {.pkg rlang} does."),
  ui_nope("Only use functions that has side-effects here.")
)

ui_stop("this is an error message generated with {.fn ui_stop}.")
ui_stop("this is a {.emph super custom} error message.",
  ui_info("You can pass other {.fn ui_*} calls to it."),
  ui_info("They are printed after the error message, like {.pkg rlang} does."),
  ui_nope("Only use functions that has side-effects here.")
)
```

```
## End(Not run)
```

Index

atomic vector, [3](#)

base::stop(), [3](#)
base::warning(), [3](#)

cli::format_error(), [3](#)
cli::format_inline(), [3](#)
cli::format_warning(), [3](#)

dotprofile (dotprofile-package), [2](#)
dotprofile-package, [2](#)
dotprofile-ui, [2](#)

inline markup, [3](#)
interactive, [3](#)

string interpolation, [4](#)

ui_ask (dotprofile-ui), [2](#)
ui_ask(), [3](#), [4](#)
ui_done (dotprofile-ui), [2](#)
ui_done(), [3](#), [4](#)
ui_info (dotprofile-ui), [2](#)
ui_info(), [3](#), [4](#)
ui_input (dotprofile-ui), [2](#)
ui_input(), [3](#), [4](#)
ui_menu (dotprofile-ui), [2](#)
ui_menu(), [3](#), [4](#)
ui_nope (dotprofile-ui), [2](#)
ui_nope(), [3](#), [4](#)
ui_stop (dotprofile-ui), [2](#)
ui_stop(), [3](#), [4](#)
ui_theme (dotprofile-ui), [2](#)
ui_theme(), [4](#)
ui_todo (dotprofile-ui), [2](#)
ui_todo(), [3](#), [4](#)
ui_warn (dotprofile-ui), [2](#)
ui_warn(), [3](#), [4](#)
utils::menu(), [3](#)