Package 'progress'

June 14, 2018

Title Terminal Progress Bars

Version 1.2.0

Author Gábor Csárdi [aut, cre], Rich FitzJohn [aut]

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Description Configurable Progress bars, they may include percentage, elapsed time, and/or the estimated completion time. They work in terminals, in 'Emacs' 'ESS', 'RStudio', 'Windows' 'Rgui' and the 'macOS' 'R.app'. The package also provides a 'C++' 'API', that works with or without 'Rcpp'.

License MIT + file LICENSE

LazyData true

URL https://github.com/r-lib/progress#readme

BugReports https://github.com/r-lib/progress/issues

Imports hms, prettyunits, R6, crayon

Suggests Rcpp, testthat, withr

RoxygenNote 6.0.1.9000

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2018-06-14 12:51:38 UTC

R topics documented:

progress_bar *Progress bar in the terminal*

Description

Progress bars are configurable, may include percentage, elapsed time, and/or the estimated completion time. They work in the command line, in Emacs and in R Studio. The progress package was heavily influenced by https://github.com/tj/node-progress

Creating the progress bar

A progress bar is an R6 object, that can be created with `progress_bar$new()`. It has the following arguments:

format The format of the progress bar. A number of tokens can be used here, see them below. It defaults to `"[:bar] :percent"`, which means that the progress bar is within brackets on the left, and the percentage is printed on the right.

total Total number of ticks to complete. If it is unknown, use `NA` here. Defaults to 100.

width Width of the progress bar. Default is the current terminal width (see `options()` and `width`) minus two.

stream This argument is deprecated, and `message()` is used to print the progress bar.

complete Completion character, defaults to `=`.

incomplete Incomplete character, defaults to `-`.

current Current character, defaults to `>`.

callback Callback function to call when the progress bar finishes. The progress bar object itself is passed to it as the single parameter.

clear Whether to clear the progress bar on completion. Defaults to `TRUE`.

show_after Amount of time in seconds, after which the progress bar is shown on the screen. For very short processes, it is probably not worth showing it at all. Defaults to two tenth of a second.

force Whether to force showing the progress bar, even if the given (or default) stream does not seem to support it.

Using the progress bar

Three functions can update a progress bar. `progress_bar$tick()` increases the number of ticks by one (or another specified value). `progress_bar$update()` sets a given ratio and `progress_bar$terminate()` removes the progress bar. `progress_bar$finished` can be used to see if the progress bar has finished.

Note that the progress bar is not shown immediately, but only after `show_after` seconds. (Set this to zero, and call 'tick(0)' to force showing the progress bar.)

`progress_bar$message()` prints a message above the progress bar. It fails if the progress bar has already finished.

Tokens

They can be used in the `format` argument when creating the progress bar.

:bar The progress bar itself.

:current Current tick number.

:total Total ticks.

:elapsed Elapsed time in seconds.

:elapsedfull Elapsed time in hh:mm:ss format.

:eta Estimated completion time in seconds.

:percent Completion percentage.

:rate Download rate, bytes per second. See example below.

:tick_rate Similar to `:rate`, but we don't assume that the units are bytes, we just print the raw number of ticks per second.

:bytes Shows :current, formatted as bytes. Useful for downloads or file reads if you don't know the size of the file in advance. See example below.

:spin Shows a spinner that updates even when progress is advanced by zero.

Custom tokens are also supported, and you need to pass their values to `progress_bar$tick()` or `progress_bar$update()`, in a named list. See example below.

Options

The 'progress_enabled' option can be set to 'FALSE' to turn off the progress bar. This works for the C++ progress bar as well.

Examples

```
## We don't run the examples on CRAN, because they takes >10s
## altogether. Unfortunately it is hard to create a set of
## meaningful progress bar examples that also run quickly.
## Not run:

## Basic
pb <- progress_bar$new(total = 100)
for (i in 1:100) {
  pb$tick()
  Sys.sleep(1 / 100)
}

## ETA
pb <- progress_bar$new(
  format = "  downloading [:bar] :percent eta: :eta",
  total = 100, clear = FALSE, width= 60)
for (i in 1:100) {
  pb$tick()
  Sys.sleep(1 / 100)
}
```

```
## Elapsed time
pb <- progress_bar$new(
  format = "  downloading [:bar] :percent in :elapsed",
  total = 100, clear = FALSE, width= 60)
for (i in 1:100) {
  pb$tick()
  Sys.sleep(1 / 100)
}

## Spinner
pb <- progress_bar$new(
  format = "(:spin) [:bar] :percent",
  total = 30, clear = FALSE, width = 60)
for (i in 1:30) {
  pb$tick()
  Sys.sleep(3 / 100)
}

## Custom tokens
pb <- progress_bar$new(
  format = "  downloading :what [:bar] :percent eta: :eta",
  clear = FALSE, total = 200, width = 60)
f <- function() {
  for (i in 1:100) {
    pb$tick(tokens = list(what = "foo   "))
    Sys.sleep(2 / 100)
  }
  for (i in 1:100) {
    pb$tick(tokens = list(what = "foobar"))
    Sys.sleep(2 / 100)
  }
}
f()

## Download (or other) rates
pb <- progress_bar$new(
  format = "  downloading foobar at :rate, got :bytes in :elapsed",
  clear = FALSE, total = NA, width = 60)
f <- function() {
  for (i in 1:100) {
    pb$tick(sample(1:100 * 1000, 1))
    Sys.sleep(2/100)
  }
  pb$tick(1e7)
  invisible()
}
f()

pb <- progress_bar$new(
  format = "  got :current rows at :tick_rate/sec",
  clear = FALSE, total = NA, width = 60)
f <- function() {
```

```
  for (i in 1:100) {
    pb$tick(sample(1:100, 1))
    Sys.sleep(2/100)
  }
  pb$terminate()
  invisible()
}
f()


## End(Not run)
```

Index