

Package ‘tictoc’

April 23, 2023

Title Functions for Timing R Scripts, as Well as Implementations of ``Stack" and ``StackList" Structures

Version 1.2

Author Sergei Izrailev

Maintainer Sergei Izrailev <sizrailev@jabiruventures.com>

Description Code execution timing functions 'tic' and 'toc' that can be nested. One can record all timings while a complex script is running, and examine the values later. It is also possible to instrument the timing calls with custom callbacks. In addition, this package provides class 'Stack', implemented as a vector, and class 'StackList', which is a stack implemented as a list, both of which support operations 'push', 'pop', 'first_element', 'last_element' and 'clear'.

URL <https://github.com/jabiru/tictoc>

Depends R (>= 2.15), methods

License Apache License (== 2.0) | file LICENSE

Copyright Copyright (C) Collective, Inc. | file inst/COPYRIGHTS

Encoding UTF-8

RoxygenNote 7.2.0

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-04-23 07:10:02 UTC

R topics documented:

Stack and StackList	2
tic	3
tictoc	6
Index	8

Stack and StackList *Stack and StackList classes and methods*

Description

`push` - Append an element.

`pop` - Remove and return the last element.

`clear` - Remove all elements.

`shift` - Remove and return the first element.

`first_element` - Return the first element. We can't use `first` because it's taken by the `dplyr` package and is not an S3 method.

`last_element` - Return the last element. We can't use `last` because it's taken by the `dplyr` package and is not an S3 method.

`size` - Return the number of elements.

`as.Stack` - Creates a new `Stack` from (typically, vector) `s`.

`as.StackList` - Creates a new `StackList` from (typically, list) `s`.

`Stack()` - Creates and keeps a stack of items of the same type, implemented as an R vector. The type is determined by the first push operation.

`StackList()` - Creates and keeps a list of items of the same type, implemented as an R list. The type is determined by the first push operation.

Usage

`push(x, value)`

`pop(x)`

`clear(x)`

`shift(x)`

`first_element(x)`

`last_element(x)`

`size(x)`

`as.Stack(s)`

`as.StackList(s)`

`Stack()`

`StackList()`

Arguments

x	A Stack or StackList object.
value	Value to append.
s	A structure to be converted to a Stack or StackList.

tic	<i>Timing utilities.</i>
-----	--------------------------

Description

tic - Starts the timer and stores the start time and the message on the stack.

toc - Notes the current timer and computes elapsed time since the matching call to tic(). When quiet is FALSE, prints the associated message and the elapsed time.

toc.outmsg - Formats a message for pretty printing. Redefine this for different formatting.

tic.clearlog - Clears the tic/toc log.

tic.clear - Clears the tic/toc stack. This could be useful in cases when because of an error the closing toc() calls never get executed.

tic.log - Returns log messages from calls to tic/toc since the last call to [tic.clearlog](#).

Usage

```
tic(msg = NULL, quiet = TRUE, func.tic = NULL, ...)
```

```
toc(log = FALSE, quiet = FALSE, func.toc = toc.outmsg, ...)
```

```
toc.outmsg(tic, toc, msg)
```

```
tic.clearlog()
```

```
tic.clear()
```

```
tic.log(format = TRUE)
```

Arguments

msg	- a text string associated with the timer. It gets printed on a call to toc()
quiet	When TRUE, doesn't print any messages
func.tic	Function producing the formatted message with a signature <code>f(tic, toc, msg, ...)</code> . Here, parameters <code>tic</code> and <code>toc</code> are the elapsed process times in seconds, so the time elapsed between the <code>tic()</code> and <code>toc()</code> calls is computed by <code>toc - tic</code> . <code>msg</code> is the string passed to the <code>tic()</code> call.
...	The other parameters that are passed to <code>func.tic</code> and <code>func.toc</code> .
log	- When TRUE, pushes the timings and the message in a list of recorded timings.

<code>func.toc</code>	Function producing the formatted message with a signature <code>f(tic, toc, msg, ...)</code> . Here, parameters <code>tic</code> and <code>toc</code> are the elapsed process times in seconds, so the time elapsed between the <code>tic()</code> and <code>toc()</code> calls is computed by <code>toc - tic</code> . <code>msg</code> is the string passed to the <code>tic()</code> call.
<code>tic</code>	Time from the call to <code>tic()</code> (<code>proc.time()["elapsed"]</code>)
<code>toc</code>	Time from the call to <code>toc()</code> (<code>proc.time()["elapsed"]</code>)
<code>format</code>	When true, <code>tic.log</code> returns a list of formatted <code>toc()</code> output, otherwise, returns the raw results.

Value

`tic` returns the timestamp (invisible).

`toc` returns an (invisible) list containing the timestamps `tic`, `toc`, and the message `msg`.

`toc.outmsg` returns formatted message.

`tic.log` returns a list of formatted messages (`format = TRUE`) or a list of lists containing the timestamps and unformatted messages from prior calls to `tic/toc`.

See Also

[tictoc](#), [Stack](#)

Examples

```
## Not run:

## Basic use case
tic()
print("Do something...")
Sys.sleep(1)
toc()
# 1.034 sec elapsed

## Inline timing example, similar to system.time()
tic(); for(i in 1:1000000) { j = i / 2 }; toc()
# 0.527 sec elapsed

## Timing multiple steps
tic("step 1")
print("Do something...")
Sys.sleep(1)
toc()
# step 1: 1.005 sec elapsed

tic("step 2")
print("Do something...")
Sys.sleep(1)
toc()
# step 2: 1.004 sec elapsed
```

```
## Timing nested code
tic("outer")
  Sys.sleep(1)
  tic("middle")
    Sys.sleep(2)
    tic("inner")
      Sys.sleep(3)
    toc()
  toc()
# inner: 3.004 sec elapsed
toc()
# middle: 5.008 sec elapsed
toc()
# outer: 6.016 sec elapsed

## Timing in a loop and analyzing the results later using tic.log().
tic.clearlog()
for (x in 1:10)
{
  tic(x)
  Sys.sleep(1)
  toc(log = TRUE, quiet = TRUE)
}
log.txt <- tic.log(format = TRUE)
log.lst <- tic.log(format = FALSE)
tic.clearlog()

timings <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
mean(timings)
# [1] 1.001
writeLines(unlist(log.txt))
# 1: 1.002 sec elapsed
# 2: 1 sec elapsed
# 3: 1.002 sec elapsed
# 4: 1.001 sec elapsed
# 5: 1.001 sec elapsed
# 6: 1.001 sec elapsed
# 7: 1.001 sec elapsed
# 8: 1.001 sec elapsed
# 9: 1.001 sec elapsed
# 10: 1 sec elapsed

## Using custom callbacks in tic/toc
my.msg.tic <- function(tic, msg)
{
  if (is.null(msg) || is.na(msg) || length(msg) == 0)
  {
    outmsg <- paste0(round(toc - tic, 3), " seconds elapsed")
  }
  else
  {
    outmsg <- paste0("Starting ", msg, "...")
  }
  outmsg
}
```

```
}

my.msg.toc <- function(tic, toc, msg, info)
{
  if (is.null(msg) || is.na(msg) || length(msg) == 0)
  {
    outmsg <- paste0(round(toc - tic, 3), " seconds elapsed")
  }
  else
  {
    outmsg <- paste0(info, ": ", msg, ": ",
                     round(toc - tic, 3), " seconds elapsed")
  }
  outmsg
}

tic("outer", quiet = FALSE, func.tic = my.msg.tic)
# Starting outer...
  Sys.sleep(1)
  tic("middle", quiet = FALSE, func.tic = my.msg.tic)
# Starting middle...
    Sys.sleep(2)
    tic("inner", quiet = FALSE, func.tic = my.msg.tic)
    Sys.sleep(3)
# Starting inner...
      toc(quiet = FALSE, func.toc = my.msg.toc, info = "INFO")
# INFO: inner: 3.005 seconds elapsed
      toc(quiet = FALSE, func.toc = my.msg.toc, info = "INFO")
# INFO: middle: 5.01 seconds elapsed
    toc(quiet = FALSE, func.toc = my.msg.toc, info = "INFO")
# INFO: outer: 6.014 seconds elapsed

## End(Not run)
```

tictoc

Package tictoc.

Description

Functions for timing, as well as implementations of Stack and StackList structures.

Details

The tictoc package provides the timing functions `tic` and `toc` that can be nested. It provides an alternative to `system.time()` with a different syntax similar to that in another well-known software package. `tic` and `toc` are easy to use, and are especially useful when timing several sections in more than a few lines of code.

In general, calls to `tic` and `toc` start the timer when the `tic` call is made and stop the timer when the `toc` call is made, recording the elapsed time between the calls from `proc.time`. The default behavior is to print a simple message with the elapsed time in the `toc` call.

The features include the following:

- nesting of the `tic` and `toc` calls
- suppressing the default output with `quiet = TRUE`
- collecting the timings in user-defined variables
- collecting the timings in a log structure provided by the package (see [tic.log](#))
- providing a custom message for each `tic` call
- using custom callbacks for the `tic` and `toc` calls to redefine the default behavior and/or add other functionality (such as logging to a database)

In addition, this package provides classes [Stack](#) (implemented as a vector) and [StackList](#) (a stack implemented as a list), both of which support operations `push`, `pop`, `first_element`, `last_element`, `clear` and `size`.

Copyright

Copyright (C) Collective, Inc.; with portions Copyright (C) Jabiru Ventures LLC

License

Apache License, Version 2.0, available at <http://www.apache.org/licenses/LICENSE-2.0>

URL

<http://github.com/jabiru/tictoc>

Installation from github

```
devtools::install_github("jabiru/tictoc")
```

Author(s)

Sergei Izrailev

See Also

[tic](#), [Stack](#)

Index

- * **list**
 - tictoc, 6
- * **profiling**
 - tictoc, 6
- * **stack**
 - tictoc, 6
- * **timing**
 - tictoc, 6

as.Stack (Stack and StackList), 2
as.StackList (Stack and StackList), 2

clear (Stack and StackList), 2

first_element (Stack and StackList), 2

last_element (Stack and StackList), 2

pop (Stack and StackList), 2
push (Stack and StackList), 2

shift (Stack and StackList), 2
size (Stack and StackList), 2
Stack, 4, 7
Stack (Stack and StackList), 2
Stack and StackList, 2
StackList, 7
StackList (Stack and StackList), 2

tic, 3, 7
tic.clearlog, 3
tic.log, 7
tictoc, 4, 6
toc (tic), 3