

# Package ‘tictoc’

December 28, 2017

**Title** Functions for timing R scripts, as well as implementations of Stack and List structures.

**Version** 1.1

**Author** Sergei Izrailev

**Maintainer** Sergei Izrailev <sizrailev@jabiruventures.com>

**Description** This package provides the 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 'List', implemented as a list, both of which support operations 'push', 'pop', 'first', 'last' and 'clear'.

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

**Depends** R (>= 2.15),  
methods

**License** Apache License (== 2.0) | file LICENSE

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

**LazyData** true

**RoxygenNote** 6.0.1

## R topics documented:

Stack	1
tic	3
tictoc	6
<b>Index</b>	<b>8</b>

---

Stack

*Stack and List 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` - Return the first element.

`last` - Return the last element.

`size` - Return the number of elements.

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

`as.List` - Creates a new List 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.

`List()` - 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**

```
## Default S3 method:  
push(x, value)
```

```
## Default S3 method:  
pop(x)
```

```
## Default S3 method:  
clear(x)
```

```
## Default S3 method:  
shift(x)
```

```
## Default S3 method:  
first(x)
```

```
## Default S3 method:  
last(x)
```

```
## Default S3 method:  
size(x)
```

```
## Default S3 method:  
as.Stack(s)
```

```
## Default S3 method:  
as.List(s)
```

```
Stack()
```

```
List()
```

**Arguments**

<code>x</code>	A Stack or List object.
<code>value</code>	Value to append.
<code>s</code>	A structure to be converted to a Stack or List.

---

<code>tic</code>	<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**

<code>msg</code>	- a text string associated with the timer. It gets printed on a call to <code>toc()</code>
<code>quiet</code>	When <code>TRUE</code> , doesn't print any messages
<code>func.tic</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>...</code>	The other parameters that are passed to <code>func.tic</code> and <code>func.toc</code> .
<code>log</code>	- When <code>TRUE</code> , 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()</code> ["elapsed"])
<code>toc</code>	Time from the call to <code>toc()</code> ( <code>proc.time()</code> ["elapsed"])
<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()
  # 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 <- paste(round(toc - tic, 3), " seconds elapsed", sep="")
  }
  else
  {
    outmsg <- paste("Starting ", msg, "...", sep="")
  }
}

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

```

```

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 List 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 `List` (implemented as a list), both of which support operations `push`, `pop`, `first`, `last`, `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

- \*Topic **list**
  - tictoc, [6](#)
- \*Topic **profiling**
  - tictoc, [6](#)
- \*Topic **stack**
  - tictoc, [6](#)
- \*Topic **timing**
  - tictoc, [6](#)
  
- as.List (*Stack*), [1](#)
- as.Stack (*Stack*), [1](#)
  
- clear (*Stack*), [1](#)
  
- first (*Stack*), [1](#)
  
- last (*Stack*), [1](#)
- List, [6](#)
- List (*Stack*), [1](#)
  
- pop (*Stack*), [1](#)
- push (*Stack*), [1](#)
  
- shift (*Stack*), [1](#)
- size (*Stack*), [1](#)
- Stack, [1](#), [4](#), [6](#), [7](#)
  
- tic, [3](#), [7](#)
- tic.clearlog, [3](#)
- tic.log, [6](#)
- tictoc, [4](#), [6](#)
- tictoc-package (*tictoc*), [6](#)
- toc (*tic*), [3](#)