

Package: memtoc (via r-universe)

September 30, 2026

Title Tictoc-style memory usage tracking

Version 0.1.0

Description Simple start/stop memory tracking functions 'tic_mem()' and 'toc_mem()' that can be nested, inspired by the 'tictoc' package. Track RAM usage during code execution with support for logging, custom messages, nested tracking blocks, and parallel worker monitoring. Features continuous background polling to capture true peak memory usage across main process and workers. Integrates with the 'future' package ecosystem for automatic worker detection. Designed for monitoring memory consumption in parallel workflows.

License MIT + file LICENSE

URL <https://github.com/jcoa05/memtoc>

BugReports <https://github.com/jcoa05/memtoc>

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports ps (>= 1.7.0), cli (>= 3.0.0), callr (>= 3.7.0)

Suggests testthat (>= 3.0.0), withr, future, parallelly, knitr, rmarkdown

Config/testthat/edition 3

OS_type windows

Language en-US

VignetteBuilder knitr

Repository <https://jcoa05.r-universe.dev>

Date/Publication 2025-12-17 20:21:48 UTC

RemoteUrl <https://github.com/jcoa05/memtoc>

RemoteRef HEAD

RemoteSha a7503e849834b9704e6fc0d3f18670c8410c1dad

Contents

memtoc-package	2
mem_capabilities	4
mem_clear	4
mem_clearlog	5
mem_diagnose	5
mem_log	6
mem_parallel_info	7
mem_print_log	8
mem_recover	9
tic_mem	10
toc_mem	11
Index	14

memtoc-package	<i>Tictoc-Style Memory Usage Tracking</i>
----------------	---

Description

The memtoc package provides simple start/stop memory tracking functions that can be nested, inspired by the tictoc package for timing. Track RAM usage during code execution with support for logging, custom messages, background polling for true peak detection, and parallel worker monitoring.

Main Functions

- `tic_mem`: Start memory tracking
- `toc_mem`: Stop memory tracking and report
- `mem_log`: Retrieve logged memory measurements
- `mem_clearlog`: Clear the memory log
- `mem_clear`: Clear the tracking stack
- `mem_recover`: Recover data from crashed sessions
- `mem_capabilities`: Check available features
- `mem_diagnose`: Detailed diagnostics
- `mem_parallel_info`: Check parallel backend status

Basic Usage

```
tic_mem("data loading")
data <- read.csv("large_file.csv")
toc_mem()
#> data loading: 142.3 MB peak | 89.1 MB current | 2.34 sec | 3 samples
```

Background Polling

By default, memtoc spawns a background process to continuously sample memory, capturing the true peak even for short-lived allocations:

```
tic_mem("computation", interval = 0.5) # Sample every 0.5 seconds
x <- matrix(rnorm(1e8), ncol = 1000)    # Large temporary allocation
result <- toc_mem()
result$trajectory # Full memory trajectory
```

Parallel Worker Monitoring

When using the future package, memtoc can monitor memory across workers:

```
library(future)
plan(multisession, workers = 4)
tic_mem("parallel job", workers = "auto")
# ... parallel computation ...
result <- toc_mem()
result$worker_stats # Per-worker memory usage
```

Nested Tracking

```
tic_mem("full pipeline")
  tic_mem("preprocessing")
  # ... code ...
  toc_mem()

  tic_mem("modeling")
  # ... code ...
  toc_mem()
toc_mem()
```

Author(s)

Maintainer: Your Name <your.email@example.com>

See Also

Useful links:

- <https://github.com/yourusername/memtoc>
- Report bugs at <https://github.com/yourusername/memtoc/issues>

mem_capabilities	<i>Check memtoc capabilities</i>
------------------	----------------------------------

Description

Tests which features are available on the current system. This is useful for understanding why certain features might be disabled.

Usage

```
mem_capabilities()
```

Value

A named logical vector with elements:

memory_queries Can query process memory via ps package

background_polling Can spawn background processes via callr

Examples

```
mem_capabilities()
```

mem_clear	<i>Clear the memtoc stack</i>
-----------	-------------------------------

Description

Removes all entries from the tracking stack. This is useful if an error occurred before `toc_mem()` could be called, leaving orphaned entries on the stack.

Usage

```
mem_clear()
```

Value

Invisibly returns NULL.

Examples

```
tic_mem("will be cleared")
mem_clear()
# Stack is now empty
```

mem_clearlog	<i>Clear the memory tracking log</i>
--------------	--------------------------------------

Description

Removes all entries from the memtoc log. This does not affect the tracking stack (active tic_mem/toc_mem blocks).

Usage

```
mem_clearlog()
```

Value

Invisibly returns NULL.

See Also

[mem_log](#) to retrieve the log before clearing

Examples

```
tic_mem("test")
toc_mem(log = TRUE, quiet = TRUE)

nrow(mem_log())
#> 1

mem_clearlog()
nrow(mem_log())
#> 0
```

mem_diagnose	<i>Diagnose memtoc capabilities</i>
--------------	-------------------------------------

Description

Runs detailed diagnostics to identify issues with background process spawning. Useful for troubleshooting when mem_capabilities() shows background_polling = FALSE.

Usage

```
mem_diagnose()
```

Details

This function tests each component of memtoc’s functionality and provides detailed error messages and suggestions for fixing issues, particularly on Windows where Rtools may be required for background process spawning.

Value

Invisibly returns a list with diagnostic results including:

- ps_ok** Whether ps memory queries work
- pids_ok** Whether process listing works
- sys_mem_ok** Whether system memory queries work
- callr_loaded** Whether callr package is installed
- callr_works** Whether background processes can be spawned
- all_ok** Whether all checks passed

See Also

[mem_capabilities](#) for a quick capability check

Examples

```
mem_diagnose()
```

mem_log	<i>Retrieve the memory tracking log</i>
---------	---

Description

Returns a data frame containing all logged memory tracking results. Results are added to the log when `toc_mem(log = TRUE)` is called.

Usage

```
mem_log(format = c("data.frame", "list"))
```

Arguments

- format** Character string specifying the output format.
 "data.frame" (Default) Returns a data frame with one row per logged result
 "list" Returns the raw list of memtoc_result objects

Value

A data frame (default) or list containing logged results. The data frame has columns:

msg Label from tic_mem(), or NA if none provided

mem_start Memory (RSS) in bytes at start

mem_end Memory (RSS) in bytes at end

mem_peak Peak memory in bytes

mem_change Change in memory (bytes)

elapsed Elapsed time in seconds

tic_timestamp When tic_mem() was called

toc_timestamp When toc_mem() was called

See Also

[mem_clearlog](#) to clear the log, [toc_mem](#) with log = TRUE

Examples

```
mem_clearlog()

tic_mem("step 1")
Sys.sleep(0.1)
toc_mem(log = TRUE, quiet = TRUE)

tic_mem("step 2")
Sys.sleep(0.1)
toc_mem(log = TRUE, quiet = TRUE)

mem_log()
```

mem_parallel_info	<i>Get information about current parallel setup</i>
-------------------	---

Description

Returns diagnostic information about the detected parallel backend and any workers that can be monitored. Useful for verifying that memtoc can detect your parallel workers before running a job.

Usage

```
mem_parallel_info()
```

Value

Invisibly returns a list with parallel backend information including:

future_available Whether the future package is installed

parallelly_available Whether the parallelly package is installed

plan Current future plan (e.g., "multisession", "sequential")

n_workers Number of configured workers

worker_pids Integer vector of detected worker PIDs

child_pids Integer vector of child process PIDs

main_pid PID of the main R process

See Also

[tic_mem](#) for the workers parameter, [mem_capabilities](#) for general capability check

Examples

```
mem_parallel_info()

# With future workers active:
# library(future)
# plan(multisession, workers = 2)
# mem_parallel_info()
```

mem_print_log	<i>Print formatted log output</i>
---------------	-----------------------------------

Description

Displays the memory tracking log in a human-readable format, similar to how tictoc's `tic.log()` output can be printed with `writeLines()`.

Usage

```
mem_print_log()
```

Value

Invisibly returns a character vector of formatted log lines.

Examples

```

mem_clearlog()

tic_mem("step 1")
Sys.sleep(0.1)
toc_mem(log = TRUE, quiet = TRUE)

tic_mem("step 2")
Sys.sleep(0.2)
toc_mem(log = TRUE, quiet = TRUE)

mem_print_log()

```

mem_recover

*Recover data from a crashed monitoring session***Description**

Attempts to recover memory samples from a previous session that crashed or was interrupted before `toc_mem()` was called. The monitor periodically saves checkpoints to disk, so some data may be recoverable.

Usage

```
mem_recover(pid = NULL, path = NULL)
```

Arguments

<code>pid</code>	Process ID of the crashed R session. If <code>NULL</code> , lists all available recovery files.
<code>path</code>	Direct path to a recovery file. Overrides <code>pid</code> if provided.

Value

If `pid` or `path` is provided, returns a data frame of recovered samples or `NULL` if not found. If both are `NULL`, returns a list of available recovery files with their metadata.

See Also

[tic_mem](#) for starting memory tracking with persistence enabled

Examples

```

# List available recovery files
mem_recover()

# Recover from a specific PID
# mem_recover(pid = 12345)

# Recover from a specific file
# mem_recover(path = "/path/to/memtoc_12345.rds")

```

tic_mem	<i>Start memory tracking</i>
---------	------------------------------

Description

Begins a memory tracking block. Call `toc_mem` to end the block and see the results. Multiple calls to `tic_mem()` can be nested, and each `toc_mem()` will match with the most recent unmatched `tic_mem()`.

Usage

```
tic_mem(msg = NULL, quiet = TRUE, interval = 1, persist = TRUE, workers = "auto")
```

Arguments

msg	Optional character string label for this tracking block. This label is displayed in the output from <code>toc_mem()</code> and stored in the log if <code>log = TRUE</code> is passed to <code>toc_mem()</code> .
quiet	Logical. If <code>TRUE</code> (default), no message is printed when tracking starts. Set to <code>FALSE</code> to see a startup message.
interval	Numeric. Polling interval in seconds for background monitoring. Set to <code>NULL</code> or <code>0</code> to disable background polling and use snapshot-only mode (faster but only captures start/end memory). Default is 1 second.
persist	Logical. If <code>TRUE</code> (default), periodically saves samples to disk for crash recovery. Set to <code>FALSE</code> to disable persistence.
workers	Worker specification for parallel monitoring: • "auto" (default): Automatically detect future workers and child processes • "none": Only monitor the main R process • "children": Monitor main process and all child processes • Integer vector: Explicit list of PIDs to monitor

Details

When `interval` is set (default), a background R process is spawned to continuously sample memory usage. This captures the true peak memory even for short-lived allocations. The background process adds minimal overhead (typically less than 1% CPU).

For very short operations (under 1 second), consider using `interval = NULL` to avoid the background process startup overhead (~200ms).

Parallel Worker Monitoring: When using `future` for parallel processing, `memtoc` can automatically detect and monitor worker processes. Set `workers = "auto"` to enable this feature. The trajectory will include memory samples from all workers, and the result will show aggregate statistics.

Value

Invisibly returns the timestamp when tracking started.

See Also

[tic_mem](#) to stop tracking, [mem_log](#) to retrieve logged results, [mem_recover](#) to recover data after a crash, [mem_parallel_info](#) to check parallel backend status

Examples

```
# Basic usage with background polling (default)
tic_mem("my operation")
Sys.sleep(0.5)
x <- numeric(1e6)
toc_mem()

# Snapshot-only mode (no background process)
tic_mem("quick op", interval = NULL)
y <- 1:100
toc_mem()

# Custom polling interval
tic_mem("long operation", interval = 0.5)
Sys.sleep(2)
toc_mem()

# Monitor with parallel workers (if future is set up)
# future::plan(future::multisession, workers = 2)
# tic_mem("parallel job", workers = "auto")
# result <- future.apply::future_lapply(1:10, function(x) rnorm(1e6))
# toc_mem()

# Nested tracking
tic_mem("outer")
  tic_mem("inner", interval = NULL)
  Sys.sleep(0.1)
  toc_mem()
toc_mem()
```

toc_mem

Stop memory tracking and report results

Description

Ends a memory tracking block started by [tic_mem](#) and reports the results. By default, prints a summary message showing peak memory, current memory, and elapsed time.

Usage

```
toc_mem(log = FALSE, quiet = FALSE)
```

Arguments

<code>log</code>	Logical. If TRUE, stores the result in an internal log that can be retrieved later with <code>mem_log</code> . Default is FALSE.
<code>quiet</code>	Logical. If TRUE, suppresses the output message. Default is FALSE (message is shown).

Value

Invisibly returns a `memtoc_result` object (a list) containing:

msg The label passed to `tic_mem()`, or NULL
mem_start Memory (RSS) in bytes at start (main process)
mem_end Memory (RSS) in bytes at end (main process)
mem_peak Peak memory during the interval (total across all processes)
mem_change Change in memory (end - start) in bytes
elapsed Elapsed time in seconds
tic_timestamp POSIXct timestamp when `tic_mem()` was called
toc_timestamp POSIXct timestamp when `toc_mem()` was called
trajectory Data frame of memory samples (if background polling was used)
n_samples Number of samples collected
n_workers Number of worker processes monitored (0 if main process only)
worker_stats Data frame with per-worker peak memory (if workers monitored)
sys_peak_percent Peak system RAM usage percentage
sys_total Total system RAM in bytes
cpu_user User CPU time consumed (seconds)
cpu_system System CPU time consumed (seconds)

See Also

`tic_mem` to start tracking, `mem_log` to retrieve logged results

Examples

```
tic_mem("example")
x <- rnorm(1e6)
Sys.sleep(1)
result <- toc_mem()

# Access result components
result$elapsed
result$mem_peak
result$n_samples

# View the full trajectory (if available)
if (!is.null(result$trajectory)) {
```

```
      head(result$trajectory)
    }

    # View per-worker stats (if workers were monitored)
    if (!is.null(result$worker_stats)) {
      result$worker_stats
    }
  }
}
```

Index

mem_capabilities, [2](#), [4](#), [6](#), [8](#)
mem_clear, [2](#), [4](#)
mem_clearlog, [2](#), [5](#), [7](#)
mem_diagnose, [2](#), [5](#)
mem_log, [2](#), [5](#), [6](#), [11](#), [12](#)
mem_parallel_info, [2](#), [7](#), [11](#)
mem_print_log, [8](#)
mem_recover, [2](#), [9](#), [11](#)
mentoc (mentoc-package), [2](#)
mentoc-package, [2](#)

tic_mem, [2](#), [8](#), [9](#), [10](#), [11](#), [12](#)
toc_mem, [2](#), [7](#), [10](#), [11](#), [11](#)