

# Package ‘simmer’

March 3, 2018

**Type** Package

**Title** Discrete-Event Simulation for R

**Version** 3.7.0

**Description** A process-oriented and trajectory-based Discrete-Event Simulation (DES) package for R. It is designed as a generic yet powerful framework. The architecture encloses a robust and fast simulation core written in 'C++' with automatic monitoring capabilities. It provides a rich and flexible R API that revolves around the concept of trajectory, a common path in the simulation model for entities of the same type.

**License** MIT + file LICENSE

**Encoding** UTF-8

**URL** <http://r-simmer.org>, <https://github.com/r-simmer/simmer>

**BugReports** <https://github.com/r-simmer/simmer/issues>

**Depends** R (>= 3.1.2)

**Imports** Rcpp, R6, magrittr, codetools

**Suggests** simmer.plot, parallel, dplyr, tidyr, testthat, knitr, rmarkdown, rticles

**LinkingTo** Rcpp (>= 0.12.9), BH (>= 1.62.0-1)

**ByteCompile** yes

**RoxygenNote** 6.0.1

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Iñaki Ucar [aut, cph, cre] (<<https://orcid.org/0000-0001-6403-5550>>),  
Bart Smeets [aut, cph]

**Maintainer** Iñaki Ucar <[i.ucar86@gmail.com](mailto:i.ucar86@gmail.com)>

**Repository** CRAN

**Date/Publication** 2018-03-03 14:03:20 UTC

**R topics documented:**

simmer-package . . . . .	2
activate . . . . .	3
add_generator . . . . .	4
add_resource . . . . .	5
at . . . . .	6
batch . . . . .	7
branch . . . . .	8
clone . . . . .	9
Extract.trajectory . . . . .	9
get_capacity . . . . .	10
get_mon . . . . .	11
get_n_generated . . . . .	12
join . . . . .	13
leave . . . . .	13
length.trajectory . . . . .	14
log_ . . . . .	15
now . . . . .	15
peek . . . . .	16
renege_in . . . . .	16
reset . . . . .	17
rollback . . . . .	18
run . . . . .	18
schedule . . . . .	19
seize . . . . .	20
select . . . . .	21
send . . . . .	21
set_attribute . . . . .	22
set_capacity . . . . .	23
set_prioritization . . . . .	24
set_trajectory . . . . .	25
simmer . . . . .	25
timeout . . . . .	26
trajectory . . . . .	27
wrap . . . . .	28
<b>Index</b>	<b>30</b>

## Description

A process-oriented and trajectory-based Discrete-Event Simulation (DES) package for R. Designed to be a generic framework like **SimPy** or **SimJulia**, it leverages the power of **Rcpp** to boost the performance and turning DES in R feasible. As a noteworthy characteristic, **simmer** exploits the concept of trajectory: a common path in the simulation model for entities of the same type. It is pretty flexible and simple to use, and leverages the chaining/piping workflow introduced by the **magrittr** package.

## Author(s)

Iñaki Ucar and Bart Smeets

## References

Iñaki Ucar, Bart Smeets and Arturo Azcorra (2018). **simmer**: Discrete-Event Simulation for R. *Journal of Statistical Software*, accepted for publication.

## See Also

**simmer**'s homepage <http://r-simmer.org> and GitHub repository <https://github.com/r-simmer/simmer>.

## Examples

```
## Not run:
# introduction to simmer
vignette("simmer-01-introduction")

# JSS paper available as vignette
vignette("simmer-02-jss")

# more vignettes
vignette(package = "simmer")

## End(Not run)
```

---

activate

*Activate/Deactivate Generators*

---

## Description

Activities for activating or deactivating the generation of arrivals by name.

## Usage

```
activate(.trj, generator)

deactivate(.trj, generator)
```

**Arguments**

.trj	the trajectory object.
generator	the name of the generator or a function returning a name.

**Value**

Returns the trajectory object.

**See Also**

[set\\_trajectory](#), [set\\_distribution](#).

---

add_generator	<i>Add a Generator</i>
---------------	------------------------

---

**Description**

Define a new generator of arrivals in a simulation environment.

**Usage**

```
add_generator(.env, name_prefix, trajectory, distribution, mon = 1,
             priority = 0, preemptible = priority, restart = FALSE)
```

**Arguments**

.env	the simulation environment.
name_prefix	the name prefix of the generated arrivals.
trajectory	the trajectory that the generated arrivals will follow (see <a href="#">trajectory</a> ).
distribution	a function modelling the interarrival times (returning a negative value stops the generator).
mon	whether the simulator must monitor the generated arrivals or not (0 = no monitoring, 1 = simple arrival monitoring, 2 = level 1 + arrival attribute monitoring)
priority	the priority of each arrival (a higher integer equals higher priority; defaults to the minimum priority, which is 0).
preemptible	if a seize occurs in a preemptive resource, this parameter establishes the minimum incoming priority that can preempt these arrivals (an arrival with a priority greater than preemptible gains the resource). In any case, preemptible must be equal or greater than priority, and thus only higher priority arrivals can trigger preemption.
restart	whether the activity must be restarted after being preempted.

**Value**

Returns the simulation environment.

**See Also**

Convenience functions: [at](#), [from](#), [to](#), [from\\_to](#).

---

add_resource	<i>Add a Resource</i>
--------------	-----------------------

---

**Description**

Define a new resource in a simulation environment.

**Usage**

```
add_resource(.env, name, capacity = 1, queue_size = Inf, mon = TRUE,
             preemptive = FALSE, preempt_order = c("fifo", "lifo"),
             queue_size_strict = FALSE)
```

**Arguments**

.env	the simulation environment.
name	the name of the resource.
capacity	the capacity of the server.
queue_size	the size of the queue.
mon	whether the simulator must monitor this resource or not.
preemptive	whether arrivals in the server can be preempted or not based on seize priorities.
preempt_order	if the resource is preemptive and preemption occurs with more than one arrival in the server, this parameter defines which arrival should be preempted first. It must be <code>fifo</code> (First In First Out: older preemptible tasks are preempted first) or <code>lifo</code> (Last In First Out: newer preemptible tasks are preempted first).
queue_size_strict	if the resource is preemptive and preemption occurs, this parameter controls whether the <code>queue_size</code> is a hard limit. By default, preempted arrivals go to a dedicated queue, so that <code>queue_size</code> may be exceeded. If this option is <code>TRUE</code> , preempted arrivals go to the standard queue, and the maximum <code>queue_size</code> is guaranteed (rejection may occur).

**Value**

Returns the simulation environment.

**See Also**

Convenience functions: [schedule](#).

---

at

*Convenience Functions for Generators*

---

### Description

These convenience functions facilitate the definition of generators of arrivals for some common cases.

### Usage

```
at(...)
```

```
from(start_time, dist, arrive = TRUE)
```

```
to(stop_time, dist)
```

```
from_to(start_time, stop_time, dist, arrive = TRUE, every = NULL)
```

### Arguments

...	a vector or multiple parameters of times at which to initiate an arrival.
start_time	the time at which to launch the initial arrival.
dist	a function modelling the interarrival times.
arrive	if set to TRUE (default) the first arrival will be generated at start_time and will follow dist from then on. If set to FALSE, will initiate dist at start_time (and the first arrival will most likely start at a time later than start_time).
stop_time	the time at which to stop the generator.
every	repeat with this time cycle.

### Details

[at](#) generates arrivals at specific absolute times. [from](#) generates inter-arrivals following a given distribution with a specified start time. [to](#) generates inter-arrivals following a given distribution with a specified stop time. [from\\_to](#) is the union of the last two.

### Value

Returns a generator function.

### See Also

[add\\_generator](#).

**Examples**

```

t0 <- trajectory() %>%
  timeout(0)

simmer() %>%
  add_generator("dummy", t0, at(0, c(1,10,30), 40, 43)) %>%
  run(100) %>%
  get_mon_arrivals()
t0 <- trajectory() %>%
  timeout(0)

simmer() %>%
  add_generator("dummy", t0, from(5, function() runif(1, 1, 2))) %>%
  run(10) %>%
  get_mon_arrivals()
t0 <- trajectory() %>%
  timeout(0)

simmer() %>%
  add_generator("dummy", t0, to(5, function() runif(1, 1, 2))) %>%
  run(10) %>%
  get_mon_arrivals()
t0 <- trajectory() %>%
  timeout(0)

# from 8 to 16 h every 24 h:
simmer() %>%
  add_generator("dummy", t0, from_to(8, 16, function() runif(1, 1, 2), every=24)) %>%
  run(48) %>%
  get_mon_arrivals()

```

---

batch

*Batch/Separate Arrivals*


---

**Description**

Activities for collecting a number of arrivals before they can continue processing and splitting a previously established batch.

**Usage**

```

batch(.trj, n, timeout = 0, permanent = FALSE, name = "", rule = NULL)

separate(.trj)

```

**Arguments**

```

.trj          the trajectory object.
n             batch size, accepts a numeric.

```

timeout	set an optional timer which triggers batches every timeout time units even if the batch size has not been fulfilled, accepts a numeric or a callable object (a function) which must return a numeric (0 = disabled).
permanent	if TRUE, batches cannot be split.
name	optional string. Unnamed batches from different batch activities are independent. However, if you want to feed arrivals from different trajectories into a same batch, you need to specify a common name across all your batch activities.
rule	an optional callable object (a function) which will be applied to every arrival to determine whether it should be included into the batch, thus

**Value**

Returns the trajectory object.

---

branch	<i>Fork the Trajectory Path</i>
--------	---------------------------------

---

**Description**

Activity for defining a fork with N alternative sub-trajectories.

**Usage**

```
branch(.trj, option, continue, ...)
```

**Arguments**

.trj	the trajectory object.
option	a callable object (a function) which must return an integer between 0 and N. A return value equal to 0 skips the branch and continues to the next activity. A returning value between 1 to N makes the arrival to follow the corresponding sub-trajectory.
continue	a vector of N booleans that indicate whether the arrival must continue to the main trajectory after each sub-trajectory or not.
...	N trajectory objects describing each sub-trajectory.

**Value**

Returns the trajectory object.



---

clone	<i>Clone/Synchronize Arrivals</i>
-------	-----------------------------------

---

**Description**

Activities for defining a parallel fork and removing the copies. clone replicates an arrival n times (the original one + n-1 copies). synchronize removes all but one clone for each set of clones.

**Usage**

```
clone(.trj, n, ...)
```

```
synchronize(.trj, wait = TRUE, mon_all = FALSE)
```

**Arguments**

.trj	the trajectory object.
n	number of clones, accepts either a numeric or a callable object (a function) which must return a numeric.
...	optional parallel sub-trajectories. Each clone will follow a different sub-trajectory if available.
wait	if FALSE, all clones but the first to arrive are removed. if TRUE (default), all clones but the last to arrive are removed.
mon_all	if TRUE, get_mon_arrivals will show one line per clone.

**Value**

Returns the trajectory object.

---

Extract.trajectory	<i>Extract or Replace Parts of a Trajectory</i>
--------------------	---

---

**Description**

Operators acting on trajectories to extract or replace parts.

**Usage**

```
## S3 method for class 'trajectory'
x[i]
```

```
## S3 method for class 'trajectory'
x[[i]]
```

```
## S3 replacement method for class 'trajectory'
x[i] <- value
```

```
## S3 replacement method for class 'trajectory'
x[[i]] <- value
```

### Arguments

x	the trajectory object.
i	indices specifying elements to extract. Indices are numeric or character or logical vectors or empty (missing) or NULL. Numeric values are coerced to integer as by <a href="#">as.integer</a> (and hence truncated towards zero). Negative integers indicate elements/slices to leave out the selection. Character vectors will be matched to the names of the activities in the trajectory as by <a href="#">%in%</a> . Logical vectors indicate elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. An empty index will return the whole trajectory. An index value of NULL is treated as if it were <code>integer(0)</code> .
value	another trajectory object.

### Value

Returns a new trajectory object.

### See Also

[length.trajectory](#), [get\\_n\\_activities](#), [join](#).

---

get\_capacity

*Get Resource Parameters*

---

### Description

Getters for resources: server capacity/count and queue size/count.

### Usage

```
get_capacity(.env, resource)
```

```
get_capacity_selected(.env, id = 0)
```

```
get_queue_size(.env, resource)
```

```
get_queue_size_selected(.env, id = 0)
```

```

get_server_count(.env, resource)

get_server_count_selected(.env, id = 0)

get_queue_count(.env, resource)

get_queue_count_selected(.env, id = 0)

```

**Arguments**

.env	the simulation environment.
resource	the name of the resource.
id	selection identifier for nested usage.

**Value**

Return a numeric value.

**See Also**

[set\\_capacity](#), [set\\_queue\\_size](#).

---

get\_mon

*Monitoring Statistics*


---

**Description**

Getters for obtaining monitored data (if any) about arrivals, attributes and resources.

**Usage**

```

get_mon_arrivals(.envs, per_resource = FALSE, ongoing = FALSE)

get_mon_attributes(.envs)

get_mon_resources(.envs)

```

**Arguments**

.envs	the simulation environment (or a list of environments).
per_resource	if TRUE, statistics will be reported on a per-resource basis.
ongoing	if TRUE, ongoing arrivals will be reported. The columns end_time and finished of these arrivals are reported as NAs.

**Value**

Returns a data frame.

---

<code>get_n_generated</code>	<i>Get Process Parameters</i>
------------------------------	-------------------------------

---

**Description**

Getters for processes (generators and arrivals) number of arrivals generated by a generator, the name of the active arrival, an attribute from the active arrival or a global one, and prioritization values.

**Usage**

```
get_n_generated(.env, generator)

get_name(.env)

get_attribute(.env, keys, global = FALSE)

get_global(.env, keys)

get_prioritization(.env)
```

**Arguments**

<code>.env</code>	the simulation environment.
<code>generator</code>	the name of the generator.
<code>keys</code>	the attribute name(s).
<code>global</code>	if TRUE, the attribute will be global instead of per-arrival.

**Details**

`get_n_generated` returns the number of arrivals generated by a given generator.

`get_name` returns the number of the running arrival. `get_attribute` returns a running arrival's attributes or global ones. If a provided key was not previously set, it returns a missing value. `get_global` is a shortcut for `get_attribute(global=TRUE)`. `get_prioritization` returns a running arrival's prioritization values. `get_name`, `get_attribute` and `get_prioritization` are meant to be used inside a trajectory; otherwise, there will be no arrival running and these functions will throw an error.

**See Also**

[set\\_attribute](#), [set\\_prioritization](#).

---

join	<i>Join Trajectories</i>
------	--------------------------

---

**Description**

Concatenate any number of trajectories in the specified order.

**Usage**

```
join(...)
```

**Arguments**

... trajectory objects.

**Value**

Returns a new trajectory object.

**See Also**

[\[.trajectory](#), [\[\[.trajectory](#), [length.trajectory](#), [get\\_n\\_activities](#).

**Examples**

```
t1 <- trajectory() %>% seize("dummy", 1)
t2 <- trajectory() %>% timeout(1)
t3 <- trajectory() %>% release("dummy", 1)

join(t1, t2, t3)

trajectory() %>%
  join(t1) %>%
  timeout(1) %>%
  join(t3)
```

---

leave	<i>Leave the Trajectory</i>
-------	-----------------------------

---

**Description**

Activity for leaving the trajectory with some probability.

**Usage**

```
leave(.trj, prob)
```

**Arguments**

.trj            the trajectory object.  
prob            a probability or a function returning a probability.

**Value**

Returns the trajectory object.

---

length.trajectory      *Number of Activities in a Trajectory*

---

**Description**

Get the number of activities in a trajectory. length returns the number of first-level activities (sub-trajectories not included). get\_n\_activities returns the total number of activities (sub-trajectories included).

**Usage**

```
## S3 method for class 'trajectory'  
length(x)  
  
get_n_activities(x)
```

**Arguments**

x            the trajectory object.

**Value**

Returns a non-negative integer of length 1.

**See Also**

[\[.trajectory](#), [\[\[.trajectory](#), [join](#).

---

log_	<i>Logging</i>
------	----------------

---

**Description**

Activity for displaying messages preceded by the simulation time and the name of the arrival.

**Usage**

```
log_(.trj, message)
```

**Arguments**

.trj	the trajectory object.
message	the message to display, accepts either a string or a callable object (a function) which must return a string.

**Value**

Returns the trajectory object.

---

now	<i>Simulation Time</i>
-----	------------------------

---

**Description**

Get the current simulation time.

**Usage**

```
now(.env)
```

**Arguments**

.env	the simulation environment.
------	-----------------------------

**Value**

Returns a numeric value.

**See Also**

[peek](#).

---

peek	<i>Peek Next Events</i>
------	-------------------------

---

**Description**

Look for future events in the event queue and (optionally) obtain info about them.

**Usage**

```
peek(.env, steps = 1, verbose = FALSE)
```

**Arguments**

.env	the simulation environment.
steps	number of steps to peek.
verbose	show additional information (i.e., the name of the process) about future events.

**Value**

Returns numeric values if verbose=F and a data frame otherwise.

**See Also**

[now](#).

---

renege_in	<i>Reneg on some Condition</i>
-----------	--------------------------------

---

**Description**

Activities for setting or unsetting a timer or a signal after which the arrival will abandon.

**Usage**

```
renege_in(.trj, t, out = NULL)
```

```
renege_if(.trj, signal, out = NULL)
```

```
renege_abort(.trj)
```



**Arguments**

<code>.trj</code>	the trajectory object.
<code>t</code>	timeout to trigger reneing, accepts either a numeric or a callable object (a function) which must return a numeric.
<code>out</code>	optional sub-trajectory in case of reneing.
<code>signal</code>	signal to trigger reneing, accepts either a string or a callable object (a function) which must return a string.

**Value**

Returns the trajectory object.

**See Also**

[send](#)

---

reset

*Reset a Simulator*

---

**Description**

Reset the following components of a simulation environment: time, event queue, resources, generators and statistics.

**Usage**

```
reset(.env)
```

**Arguments**

<code>.env</code>	the simulation environment.
-------------------	-----------------------------

**Value**

Returns the simulation environment.

**See Also**

[stepn](#), [run](#).

---

rollback	<i>Rollback a Number of Activities</i>
----------	--

---

**Description**

Activity for going backwards to a previous point in the trajectory. Useful to implement loops.

**Usage**

```
rollback(.trj, amount, times = Inf, check = NULL)
```

**Arguments**

.trj	the trajectory object.
amount	the amount of activities (of the same or parent trajectories) to roll back.
times	the number of repetitions until an arrival may continue.
check	a callable object (a function) which must return a boolean. If present, the times parameter is ignored, and the activity uses this function to check whether the rollback must be done or not.

**Value**

Returns the trajectory object.

---

run	<i>Run a Simulation</i>
-----	-------------------------

---

**Description**

Execute steps until a given criterion.

**Usage**

```
run(.env, until = 1000, progress = NULL, steps = 10)
```

```
stepn(.env, n = 1)
```

**Arguments**

.env	the simulation environment.
until	stop time.
progress	optional callback to show the progress of the simulation. The completed ratio is periodically passed as argument to the callback.
steps	number of steps to show as progress (it takes effect only if progress is provided).
n	number of events to simulate.

**Value**

Returns the simulation environment.

**See Also**

[reset.](#)

---

schedule	<i>Generate a Scheduling Object</i>
----------	-------------------------------------

---

**Description**

Resource convenience function to generate a scheduling object from a timetable specification.

**Usage**

```
schedule(timetable, values, period = Inf)
```

**Arguments**

timetable	absolute points in time in which the desired value changes.
values	one value for each point in time.
period	period of repetition.

**Value**

Returns a schedule object.

**See Also**

[add\\_resource.](#)

**Examples**

```
# Schedule 3 units from 8 to 16 h
#           2 units from 16 to 24 h
#           1 units from 24 to 8 h
capacity_schedule <- schedule(c(8, 16, 24), c(3, 2, 1), period=24)

env <- simmer() %>%
  add_resource("dummy", capacity_schedule)
```

---

 seize
 

---



---

*Seize/Release Resources*


---

**Description**

Activities for seizing/releasing a resource, by name or a previously selected one.

**Usage**

```
seize(.trj, resource, amount = 1, continue = NULL, post.seize = NULL,
      reject = NULL)
```

```
seize_selected(.trj, amount = 1, id = 0, continue = NULL,
               post.seize = NULL, reject = NULL)
```

```
release(.trj, resource, amount = 1)
```

```
release_selected(.trj, amount = 1, id = 0)
```

**Arguments**

<code>.trj</code>	the trajectory object.
<code>resource</code>	the name of the resource.
<code>amount</code>	the amount to seize/release, accepts either a numeric or a callable object (a function) which must return a numeric.
<code>continue</code>	a boolean (if <code>post.seize</code> OR <code>reject</code> is defined) or a pair of booleans (if <code>post.seize</code> AND <code>reject</code> are defined) to indicate whether these subtrajectories should continue to the next activity in the main trajectory.
<code>post.seize</code>	an optional trajectory object which will be followed after a successful seize.
<code>reject</code>	an optional trajectory object which will be followed if the arrival is rejected.
<code>id</code>	selection identifier for nested usage.

**Value**

Returns the trajectory object.

**See Also**

[select](#), [set\\_capacity](#), [set\\_queue\\_size](#), [set\\_capacity\\_selected](#), [set\\_queue\\_size\\_selected](#).

---

select	<i>Select Resources</i>
--------	-------------------------

---

**Description**

Activity for selecting a resource for a subsequent seize/release or setting its parameters (capacity or queue size).

**Usage**

```
select(.trj, resources, policy = c("shortest-queue", "round-robin",
    "first-available", "random"), id = 0)
```

**Arguments**

.trj	the trajectory object.
resources	one or more resource names, or a callable object (a function) which must return one or more resource names.
policy	if resources is a character vector, this parameter determines the criteria for selecting a resource among the set of policies available: 'shortest-queue' selects the least busy resource, 'round-robin' selects the resources in order cyclically, 'first-available' selects the first resource available, and 'random' selects one at random.
id	selection identifier for nested usage.

**Value**

Returns the trajectory object.

**See Also**

[seize\\_selected](#), [release\\_selected](#), [set\\_capacity\\_selected](#), [set\\_queue\\_size\\_selected](#).

---

send	<i>Inter-arrival Communication</i>
------	------------------------------------

---

**Description**

These activities enable asynchronous programming. `send()` broadcasts a signal or a list of signals. Arrivals can subscribe to signals and (optionally) assign a handler with `trap()`. Note that, while inside a batch, all the signals subscribed before entering the batch are ignored. Upon a signal reception, the arrival stops the current activity and executes the handler (if provided). Then, the execution returns to the activity following the point of the interruption. `untrap()` can be used to unsubscribe from signals. `wait()` blocks until a signal is received.

**Usage**

```

send(.trj, signals, delay = 0)

trap(.trj, signals, handler = NULL, interruptible = TRUE)

untrap(.trj, signals)

wait(.trj)

```

**Arguments**

.trj	the trajectory object.
signals	signal or list of signals, accepts either a string, a list of strings or a callable object (a function) which must return a string or a list of strings.
delay	optional timeout to trigger the signals, accepts either a numeric or a callable object (a function) which must return a numeric.
handler	optional trajectory object to handle a signal received.
interruptible	whether the handler can be interrupted by signals.

**Value**

Returns the trajectory object.

**See Also**

[renege\\_if](#)

---

set_attribute	<i>Set Attributes</i>
---------------	-----------------------

---

**Description**

Activity for modifying an arrival's attributes.

**Usage**

```

set_attribute(.trj, keys, values, global = FALSE, mod = c(NA, "+", "*"))

set_global(.trj, keys, values, mod = c(NA, "+", "*"))

```

**Arguments**

.trj	the trajectory object.
keys	the attribute name(s), or a callable object (a function) which must return attribute name(s).
values	numeric value(s) to set, or a callable object (a function) which must return numeric value(s).
global	if TRUE, the attribute will be global instead of per-arrival.
mod	if set, values modify the attributes rather than substituting them.

**Details**

set\_global is a shortcut for set\_attribute(global=TRUE).

**Value**

Returns the trajectory object.

**See Also**

[get\\_attribute](#).

---

set\_capacity

*Set Resource Parameters*

---

**Description**

Activities for modifying a resource's server capacity or queue size, by name or a previously selected one.

**Usage**

```
set_capacity(.trj, resource, value, mod = c(NA, "+", "*"))
```

```
set_capacity_selected(.trj, value, id = 0, mod = c(NA, "+", "*"))
```

```
set_queue_size(.trj, resource, value, mod = c(NA, "+", "*"))
```

```
set_queue_size_selected(.trj, value, id = 0, mod = c(NA, "+", "*"))
```

**Arguments**

.trj	the trajectory object.
resource	the name of the resource.
value	new value to set.
mod	if set, values modify the attributes rather than substituting them.
id	selection identifier for nested usage.

**Value**

Returns the trajectory object.

**See Also**

[select](#), [seize](#), [release](#), [seize\\_selected](#), [release\\_selected](#), [get\\_capacity](#), [get\\_queue\\_size](#).

---

set\_prioritization      *Set Prioritization Values*

---

**Description**

Activity for modifying an arrival's prioritization values.

**Usage**

```
set_prioritization(.trj, values, mod = c(NA, "+", "*"))
```

**Arguments**

.trj	the trajectory object.
values	expects either a vector/list or a callable object (a function) returning a vector/list of three values <code>c(priority, preemptible, restart)</code> . A negative value leaves the corresponding parameter unchanged. See <a href="#">add_generator</a> for more information about these parameters.
mod	if set, values modify the attributes rather than substituting them.

**Value**

Returns the trajectory object.

**See Also**

[get\\_prioritization](#).



---

set_trajectory	<i>Set Generator Parameters</i>
----------------	---------------------------------

---

**Description**

Activities for modifying a generator's trajectory or distribution by name.

**Usage**

```
set_trajectory(.trj, generator, trajectory)
```

```
set_distribution(.trj, generator, distribution)
```

**Arguments**

.trj	the trajectory object.
generator	the name of the generator or a function returning a name.
trajectory	the trajectory that the generated arrivals will follow.
distribution	a function modelling the interarrival times (returning a negative value stops the generator).

**Value**

Returns the trajectory object.

**See Also**

[activate](#), [deactivate](#).

---

simmer	<i>Create a Simulator</i>
--------	---------------------------

---

**Description**

This method initialises a simulation environment.

**Usage**

```
simmer(name = "anonymous", verbose = FALSE)
```

**Arguments**

name	the name of the simulator.
verbose	enable showing activity information.

**Value**

Returns a simulation environment.

**See Also**

Methods for dealing with a simulation environment: [reset](#), [now](#), [peek](#), [stepn](#), [run](#), [add\\_resource](#), [add\\_generator](#), [get\\_mon\\_arrivals](#), [get\\_mon\\_attributes](#), [get\\_mon\\_resources](#), [get\\_n\\_generated](#), [get\\_capacity](#), [get\\_queue\\_size](#), [get\\_server\\_count](#), [get\\_queue\\_count](#).

**Examples**

```
t0 <- trajectory("my trajectory") %>%
  ## add an intake activity
  seize("nurse", 1) %>%
  timeout(function() rnorm(1, 15)) %>%
  release("nurse", 1) %>%
  ## add a consultation activity
  seize("doctor", 1) %>%
  timeout(function() rnorm(1, 20)) %>%
  release("doctor", 1) %>%
  ## add a planning activity
  seize("administration", 1) %>%
  timeout(function() rnorm(1, 5)) %>%
  release("administration", 1)

env <- simmer("SuperDuperSim") %>%
  add_resource("nurse", 1) %>%
  add_resource("doctor", 2) %>%
  add_resource("administration", 1) %>%
  add_generator("patient", t0, function() rnorm(1, 10, 2)) %>%
  run(until=80)
```

---

timeout

*Delay*

---

**Description**

Activity for inserting delays and execute user-defined tasks.

**Usage**

```
timeout(.trj, task)
```

```
timeout_from_attribute(.trj, key, global = FALSE)
```

```
timeout_from_global(.trj, key)
```

**Arguments**

.trj	the trajectory object.
task	the timeout duration supplied by either passing a numeric or a callable object (a function) which must return a numeric (negative values are automatically coerced to positive).
key	the attribute name, or a callable object (a function) which must return the attribute name.
global	if TRUE, the attribute will be global instead of per-arrival.

**Details**

timeout\_from\_global is a shortcut for timeout\_from\_attribute(global=TRUE).

**Value**

Returns the trajectory object.

---

trajectory	<i>Create a Trajectory</i>
------------	----------------------------

---

**Description**

This method initialises a trajectory object, which comprises a chain of activities that can be attached to a generator.

**Usage**

```
trajectory(name = "anonymous", verbose = FALSE)
```

**Arguments**

name	the name of the trajectory.
verbose	enable showing additional information.

**Value**

Returns an environment that represents the trajectory.

**See Also**

Methods for dealing with trajectories: [\[.trajectory](#), [\[\[.trajectory](#), [length.trajectory](#), [get\\_n\\_activities](#), [join](#), [seize](#), [release](#), [seize\\_selected](#), [release\\_selected](#), [select](#), [set\\_capacity](#), [set\\_queue\\_size](#), [set\\_capacity\\_selected](#), [set\\_queue\\_size\\_selected](#), [set\\_prioritization](#), [activate](#), [deactivate](#), [set\\_trajectory](#), [set\\_distribution](#), [set\\_attribute](#), [timeout](#), [branch](#), [rollback](#), [leave](#), [renege\\_in](#), [renege\\_if](#), [renege\\_abort](#), [clone](#), [synchronize](#), [batch](#), [separate](#), [send](#), [trap](#), [untrap](#), [wait](#), [log\\_.](#)

**Examples**

```

t0 <- trajectory("my trajectory") %>%
  ## add an intake activity
  seize("nurse", 1) %>%
  timeout(function() rnorm(1, 15)) %>%
  release("nurse", 1) %>%
  ## add a consultation activity
  seize("doctor", 1) %>%
  timeout(function() rnorm(1, 20)) %>%
  release("doctor", 1) %>%
  ## add a planning activity
  seize("administration", 1) %>%
  timeout(function() rnorm(1, 5)) %>%
  release("administration", 1)

t0

t1 <- trajectory("trajectory with a branch") %>%
  seize("server", 1) %>%
  # 50-50 chance for each branch
  branch(function() sample(1:2, 1), continue=c(TRUE, FALSE),
    trajectory("branch1") %>%
      timeout(function() 1),
    trajectory("branch2") %>%
      timeout(function() rexp(1, 3)) %>%
      release("server", 1)
  ) %>%
  # only the first branch continues here
  release("server", 1) %>%
  timeout(function() 2)

t1

```

---

wrap

*Wrap a Simulation Environment*


---

**Description**

This function extracts the monitored data from a simulation environment making it accessible through the same methods. Only useful if you want to parallelize heavy replicas (see the example below), because the C++ simulation backend is destroyed when the threads exit.

**Usage**

```
wrap(.env)
```

**Arguments**

```
.env
```

the simulation environment.

**Value**

Returns a simulation wrapper.

**See Also**

Methods for dealing with a simulation wrapper: [get\\_mon\\_arrivals](#), [get\\_mon\\_attributes](#), [get\\_mon\\_resources](#), [get\\_n\\_generated](#), [get\\_capacity](#), [get\\_queue\\_size](#), [get\\_server\\_count](#), [get\\_queue\\_count](#).

**Examples**

```
## Not run:
library(parallel)

mm1 <- trajectory() %>%
  seize("server", 1) %>%
  timeout(function() rexp(1, 2)) %>%
  release("server", 1)

envs <- mclapply(1:4, function(i) {
  simmer("M/M/1 example") %>%
    add_resource("server", 1) %>%
    add_generator("customer", mm1, function() rexp(1, 1)) %>%
    run(100) %>%
    wrap()
})

## End(Not run)
```

# Index

[.trajectory, [13](#), [14](#), [27](#)  
[.trajectory (Extract.trajectory), [9](#)  
[<-.trajectory (Extract.trajectory), [9](#)  
[[.trajectory, [13](#), [14](#), [27](#)  
[[.trajectory (Extract.trajectory), [9](#)  
[[<-.trajectory (Extract.trajectory), [9](#)  
%in%, [10](#)

activate, [3](#), [25](#), [27](#)  
add\_generator, [4](#), [6](#), [24](#), [26](#)  
add\_resource, [5](#), [19](#), [26](#)  
as.integer, [10](#)  
at, [5](#), [6](#), [6](#)

batch, [7](#), [27](#)  
branch, [8](#), [27](#)

clone, [9](#), [27](#)

deactivate, [25](#), [27](#)  
deactivate (activate), [3](#)

Extract.trajectory, [9](#)

from, [5](#), [6](#)  
from (at), [6](#)  
from\_to, [5](#), [6](#)  
from\_to (at), [6](#)

get\_attribute, [23](#)  
get\_attribute (get\_n\_generated), [12](#)  
get\_capacity, [10](#), [24](#), [26](#), [29](#)  
get\_capacity\_selected (get\_capacity), [10](#)  
get\_global (get\_n\_generated), [12](#)  
get\_mon, [11](#)  
get\_mon\_arrivals, [26](#), [29](#)  
get\_mon\_arrivals (get\_mon), [11](#)  
get\_mon\_attributes, [26](#), [29](#)  
get\_mon\_attributes (get\_mon), [11](#)  
get\_mon\_resources, [26](#), [29](#)  
get\_mon\_resources (get\_mon), [11](#)

get\_n\_activities, [10](#), [13](#), [27](#)  
get\_n\_activities (length.trajectory), [14](#)  
get\_n\_generated, [12](#), [26](#), [29](#)  
get\_name (get\_n\_generated), [12](#)  
get\_prioritization, [24](#)  
get\_prioritization (get\_n\_generated), [12](#)  
get\_queue\_count, [26](#), [29](#)  
get\_queue\_count (get\_capacity), [10](#)  
get\_queue\_count\_selected  
    (get\_capacity), [10](#)  
get\_queue\_size, [24](#), [26](#), [29](#)  
get\_queue\_size (get\_capacity), [10](#)  
get\_queue\_size\_selected (get\_capacity),  
    [10](#)  
get\_server\_count, [26](#), [29](#)  
get\_server\_count (get\_capacity), [10](#)  
get\_server\_count\_selected  
    (get\_capacity), [10](#)

join, [10](#), [13](#), [14](#), [27](#)

leave, [13](#), [27](#)  
length.trajectory, [10](#), [13](#), [14](#), [27](#)  
log\_, [15](#), [27](#)

now, [15](#), [16](#), [26](#)

peek, [15](#), [16](#), [26](#)

release, [24](#), [27](#)  
release (seize), [20](#)  
release\_selected, [21](#), [24](#), [27](#)  
release\_selected (seize), [20](#)  
renege\_abort, [27](#)  
renege\_abort (renege\_in), [16](#)  
renege\_if, [22](#), [27](#)  
renege\_if (renege\_in), [16](#)  
renege\_in, [16](#), [27](#)  
reset, [17](#), [19](#), [26](#)  
rollback, [18](#), [27](#)  
run, [17](#), [18](#), [26](#)

schedule, [5](#), [19](#)  
seize, [20](#), [24](#), [27](#)  
seize\_selected, [21](#), [24](#), [27](#)  
seize\_selected (seize), [20](#)  
select, [20](#), [21](#), [24](#), [27](#)  
send, [17](#), [21](#), [27](#)  
separate, [27](#)  
separate (batch), [7](#)  
set\_attribute, [12](#), [22](#), [27](#)  
set\_capacity, [11](#), [20](#), [23](#), [27](#)  
set\_capacity\_selected, [20](#), [21](#), [27](#)  
set\_capacity\_selected (set\_capacity), [23](#)  
set\_distribution, [4](#), [27](#)  
set\_distribution (set\_trajectory), [25](#)  
set\_global (set\_attribute), [22](#)  
set\_prioritization, [12](#), [24](#), [27](#)  
set\_queue\_size, [11](#), [20](#), [27](#)  
set\_queue\_size (set\_capacity), [23](#)  
set\_queue\_size\_selected, [20](#), [21](#), [27](#)  
set\_queue\_size\_selected (set\_capacity),  
[23](#)  
set\_trajectory, [4](#), [25](#), [27](#)  
simmer, [25](#)  
simmer-package, [2](#)  
stepn, [17](#), [26](#)  
stepn (run), [18](#)  
synchronize, [27](#)  
synchronize (clone), [9](#)

timeout, [26](#), [27](#)  
timeout\_from\_attribute (timeout), [26](#)  
timeout\_from\_global (timeout), [26](#)  
to, [5](#), [6](#)  
to (at), [6](#)  
trajectory, [4](#), [27](#)  
trap, [27](#)  
trap (send), [21](#)

untrap, [27](#)  
untrap (send), [21](#)

wait, [27](#)  
wait (send), [21](#)  
wrap, [28](#)