

# Package ‘simmer.bricks’

January 9, 2019

**Type** Package

**Title** Helper Methods for 'simmer' Trajectories

**Version** 0.2.1

**Description** Provides wrappers for common activity patterns in 'simmer' trajectories.

**License** MIT + file LICENSE

**Encoding** UTF-8

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

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

**Depends** R (>= 3.1.2), simmer (>= 3.7.0)

**Suggests** testthat, knitr, rmarkdown

**ByteCompile** yes

**RoxygenNote** 6.1.1

**VignetteBuilder** knitr

**NeedsCompilation** no

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

**Maintainer** Iñaki Ucar <[iucar@fedoraproject.org](mailto:iucar@fedoraproject.org)>

**Repository** CRAN

**Date/Publication** 2019-01-09 10:00:03 UTC

## R topics documented:

simmer.bricks-package . . . . .	2
delayed_release . . . . .	2
do_parallel . . . . .	3
interleave . . . . .	5
visit . . . . .	6
wait_n . . . . .	7

<b>Index</b>	<b>9</b>
--------------	----------

---

simmer.bricks-package **simmer.bricks**: *Helper Methods for **simmer** Trajectories*

---

### Description

Provides wrappers for common activity patterns in **simmer** trajectories.

### Author(s)

Iñaki Ucar

### See Also

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

---

delayed\_release      *Delayed Release of a Resource*

---

### Description

This brick encapsulates a delayed release: the arrival releases the resource and continues its way immediately, but the resource is busy for an additional period of time.

### Usage

```
delayed_release(.trj, resource, task, amount = 1, preemptive = FALSE,
  mon_all = FALSE)
```

```
delayed_release_selected(.trj, task, amount = 1, preemptive = FALSE,
  mon_all = FALSE)
```

### Arguments

.trj	the trajectory object.
resource	the name of the resource.
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).
amount	the amount to seize/release, accepts either a numeric or a callable object (a function) which must return a numeric.
preemptive	whether arrivals in the server can be preempted or not based on seize priorities.
mon_all	if TRUE, get_mon_arrivals will show one line per clone.

**Value**

Returns the following chain of activities: `clone` > `synchronize` (see examples below).

**Examples**

```
## These are equivalent for a non-preemptive resource:
trajectory() %>%
  delayed_release("res1", 5, 1)

trajectory() %>%
  clone(
    2,
    trajectory() %>%
      set_capacity("res1", -1, mod="+") %>%
      release("res1", 1),
    trajectory() %>%
      timeout(5) %>%
      set_capacity("res1", 1, mod="+")
  ) %>%
  synchronize(wait=FALSE)

## These are equivalent for a preemptive resource:
trajectory() %>%
  delayed_release("res2", 5, 1, preemptive=TRUE)

trajectory() %>%
  clone(
    2,
    trajectory() %>%
      release("res2", 1),
    trajectory() %>%
      set_prioritization(c(rep(.Machine$integer.max, 2), 0)) %>%
      seize("res2", 1) %>%
      timeout(5) %>%
      release("res2", 1)
  ) %>%
  synchronize(wait=FALSE)
```

---

do\_parallel

*Perform Parallel Tasks*


---

**Description**

This brick encapsulates the activity of `n` workers running parallel sub-trajectories.

**Usage**

```
do_parallel(.trj, ..., .env, wait = TRUE, mon_all = FALSE)
```

**Arguments**

.trj	the trajectory object.
...	sub-trajectories or list of sub-trajectories to parallelise.
.env	the simulation environment.
wait	if TRUE, the arrival waits until all parallel sub-trajectories are finished; if FALSE, the arrival continues as soon as the first parallel task ends.
mon_all	if TRUE, get_mon_arrivals will show one line per clone.

**Value**

Returns the following chain of activities: `clone > synchronize (> wait > untrap` if `wait=FALSE`) (see examples below).

**Examples**

```
env <- simmer()
signal <- function() get_name(env)

task.1 <- trajectory("task 1") %>%
  timeout(function() rexp(1))
task.2 <- trajectory("task 2") %>%
  timeout(function() rexp(1))

## These are equivalent:
trajectory() %>%
  do_parallel(
    task.1,
    task.2,
    .env = env, wait = TRUE
  )

trajectory() %>%
  clone(
    n = 3,
    trajectory("original") %>%
      trap(signal) %>%
      wait() %>%
      wait() %>%
      untrap(signal),
    task.1[] %>%
      send(signal),
    task.2[] %>%
      send(signal)) %>%
  synchronize(wait = TRUE)

## These are equivalent:
trajectory() %>%
  do_parallel(
    task.1,
    task.2,
```

```

        .env = env, wait = FALSE
    )

trajectory() %>%
  clone(
    n = 3,
    trajectory("original") %>%
      trap(signal),
    task.1[] %>%
      send(signal),
    task.2[] %>%
      send(signal)) %>%
  synchronize(wait = FALSE) %>%
  wait() %>%
  untrap(signal)

```

---

interleave

*Interleaved Resources*


---

### Description

This brick encapsulates a chain of interleaved resources, i.e., the current resource is not released until the next one in the chain is available. An interesting property of such a pattern is that, if one resource is blocked for some reason, the whole chain stops.

### Usage

```
interleave(.trj, resources, task, amount = 1)
```

### Arguments

.trj	the trajectory object.
resources	character vector of resource names.
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).
amount	the amount to seize/release, accepts either a numeric or a callable object (a function) which must return a numeric.

### Details

Both task and amount accept a list of values/functions, instead of a single one, that should be of the same length as resources, so that each value/function is applied to the resource of the same index.

The transition to the second and subsequent resources is guarded by a token, an auxiliary resource whose capacity must be equal to the capacity + queue size of the guarded resource, and its queue

size must be infinite. For example, if two resources are provided, `c("A", "B")`, the auxiliary resource will be named `"B_token"`. If `capacity=2` and `queue_size=1` for B, then `capacity=3` and `queue_size=Inf` must be the values for `B_token`. But note that the user is responsible for adding such an auxiliary resource to the simulation environment with the appropriate parameters.

### Value

Returns the following chain of activities: `seize (1) > timeout > [seize (token to 2) > release (1) > seize (2) > timeout > release (2) > release (token to 2) > ... (repeat) ]` (see examples below). Thus, the total number of activities appended is  $\text{length}(\text{resources}) * 3 + (\text{length}(\text{resources}) - 1) * 2$ .

### Examples

```
## These are equivalent:
trajectory() %>%
  interleave(c("A", "B"), c(2, 10), 1)

trajectory() %>%
  seize("A", 1) %>%
  timeout(2) %>%
  seize("B_token", 1) %>%
  release("A", 1) %>%
  seize("B", 1) %>%
  timeout(10) %>%
  release("B", 1) %>%
  release("B_token", 1)
```

---

visit

*Visit a Resource*

---

### Description

These bricks encapsulate a resource visit: `seize`, spend some time and `release`.

### Usage

```
visit(.trj, resource, task, amount = 1)

visit_selected(.trj, task, amount = 1, id = 0)
```

### Arguments

<code>.trj</code>	the trajectory object.
<code>resource</code>	the name of the resource.
<code>task</code>	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).

amount	the amount to seize/release, accepts either a numeric or a callable object (a function) which must return a numeric.
id	selection identifier for nested usage.

**Value**

Returns the following chain of activities: `seize > timeout > release` (see examples below).

**Examples**

```
## These are equivalent:
trajectory() %>%
  visit("res", 5, 1)

trajectory() %>%
  seize("res", 1) %>%
  timeout(5) %>%
  release("res", 1)

## These are equivalent:
trajectory() %>%
  visit_selected(5, 1)

trajectory() %>%
  seize_selected(1) %>%
  timeout(5) %>%
  release_selected(1)
```

---

wait\_n

*Wait a Number of Signals*


---

**Description**

These bricks encapsulate n stops: wait for a sequence of n signals. `wait_until` also traps and untraps the required signals.

**Usage**

```
wait_n(.trj, n = 1)

wait_until(.trj, signals, n = 1)
```

**Arguments**

.trj	the trajectory object.
n	number of wait activities to chain.
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.

**Value**

wait\_n returns n times `wait`. `wait_until` also adds `trap` and `untrap` at the beginning and end, respectively, of the chain of waits (see examples below).

**Examples**

```
## These are equivalent:
trajectory() %>%
  wait_n(3)

trajectory() %>%
  wait() %>%
  wait() %>%
  wait()

## These are equivalent:
trajectory() %>%
  wait_until("green")

trajectory() %>%
  trap("green") %>%
  wait() %>%
  untrap("green")

## These are equivalent:
trajectory() %>%
  wait_until(c("one", "another"), 2)

trajectory() %>%
  trap(c("one", "another")) %>%
  wait() %>%
  wait() %>%
  untrap(c("one", "another"))
```



# Index

clone, [3](#), [4](#)

delayed\_release, [2](#)  
delayed\_release\_selected  
    (delayed\_release), [2](#)  
do\_parallel, [3](#)

interleave, [5](#)

release, [6](#), [7](#)

seize, [6](#), [7](#)  
simmer.bricks-package, [2](#)  
synchronize, [3](#), [4](#)

timeout, [6](#), [7](#)

trap, [8](#)

untrap, [4](#), [8](#)

visit, [6](#)  
visit\_selected (visit), [6](#)

wait, [4](#), [8](#)  
wait\_n, [7](#)  
wait\_until (wait\_n), [7](#)