Package ‘mrgsim.parallel’

Type Package

Title Simulate with 'mrgsolve' in Parallel

Version 0.2.1

Author Kyle Baron

Maintainer Kyle Baron <kylebtwin@imap.cc>

Description Simulation from an 'mrgsolve'

<https://cran.r-project.org/package=mrgsolve> model using a parallel backend.

Input data sets are split (chunked) and simulated in parallel using

mclapply() or future_lapply()

<https://cran.r-project.org/package=future.apply>.

License GPL (>= 2)

Imports parallel, dplyr, future, future.apply, callr, fst

Depends mrgsolve, R (>= 3.5.0)

Suggests testthat, arrow, qs, knitr, rmarkdown

Encoding UTF-8

URL https://github.com/kylebaron/mrgsim.parallel

BugReports https://github.com/kylebaron/mrgsim.parallel/issues

RoxygenNote 7.1.2

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-17 19:50:05 UTC

R topics documented:

bg_mclapply ........................................... 2
bg_mrgsim_d ........................................... 3
chunk_data_frame ........................................ 5
bg_mclapply

Multicore lapply in the background

Description

Multicore lapply in the background

Usage

bg_mclapply(X, FUN, mc.cores = 1, ..., .wait = TRUE, .seed = NULL)

Arguments

X
FUN
mc.cores
... .wait .seed
A list.
The function to be applied to each element of X.
Passed to parallel::mclapply().
Arguments passed to FUN.
If FALSE, the function returns immediately; if TRUE, then wait until the background job is finished.
A numeric value used to set the seed for the simulation; this is the only way to control the random number generation for your simulation.
Value

A list of output data.

Examples

ans <- bg_mclapply(seq(10), sqrt, mc.cores = 2)

bg_mrgsim_d

Description

This function uses `callr::r_bg()` to simulate a dataset in the background, optionally in parallel and optionally saving the results directly to disk in fst, arrow or rds format. Parallelization can be mediated by the parallel package on unix or macos or future on any os.

Usage

```r
bg_mrgsim_d(
  mod,
  data,
  nchunk = 1,
  ..., 
  .locker = NULL,
  .tag = NULL,
  .format = c("fst", "feather", "rds"),
  .wait = TRUE,
  .seed = FALSE,
  .cores = 1,
  .plan = NULL
)
```

Arguments

- `mod`: A model object.
- `data`: Data set to simulate; see `mrgsolve::data_set()`.
- `nchunk`: Number of chunks in which to split the data set.
- `...`: Arguments passed to `mrgsolve::mrgsim()`.
- `.locker`: A directory for saving simulated data; use this to collect results from several different runs in a single folder.
- `.tag`: A name to use for the current run; results are saved under `.tag` in `.path` folder.
bg_mrgsim_d

.format
The output format for saving simulations; using format `fst` will allow saved results to be read with `fst::read_fst()`; using format `arrow` will allow saved results to be read with `arrow::open_dataset()` with `format = "feather"`; note that `fst` is installed with `mrgsim.parallel` but `arrow` may need explicit installation.

.wait
If FALSE, the function returns immediately; if TRUE, then wait until the background job is finished.

.seed
A numeric value used to set the seed for the simulation; this is the only way to control the random number generation for your simulation.

.cores
The number of cores to parallelize across; pass 1 to run the simulation sequentially.

.plan
The name of a `future::plan()` strategy; if passed, the parallelization will be handled by the future package.

Details

`bg_mrgsim_d()` returns a `processx::process` object (follow that link to see a list of methods). You will have to call `process$x_get_result()` to retrieve the result. When an output .locker is not specified, simulated data are returned; when an output .locker is specified, the path to the `fst` file on disk is returned. The `fst` files should be read with `fst::read_fst()`. When the results are not saved to .locker, you will get a single data frame when `nchunk` is 1 or a list of data frames when `nchunk` is greater than 1. It is safest to call `dplyr::bind_rows()` or something equivalent on the result if you are expecting data frame.

Value

An r_process object; see `callr::r_bg()`. Call `process$x_get_result()` to get the actual result (see details). If a .locker path is supplied, the simulated data is saved to disk and a list of file names is returned.

See Also

`future_mrgsim_d()`, `internalize_fst()`, `list_fst()`, `head_fst()`, `setup_locker()`

Examples

```r
mod <- mrgsolve::house(delta = 24, end = 168)
data <- mrgsolve::expand.ev(
  amt = c(100, 300, 450),
  ID = 1:100,
  ii = 24,
  addl = 6
)
data <- dplyr::mutate(data, dose = amt)
process <- bg_mrgsim_d(
  mod,
data,
carry_out = "dose",
outvars = "CP",
)```
chunk_data_frame

\[
\text{.wait} = \text{TRUE}
\]

process$get_result()

ds <- file.path(tempdir(), "sims")
files <- bg_mrgsim_d(
  mod, data, carry_out = "dose",
  .wait = \text{TRUE},
  .locker = ds,
  .format = "fst"
)
files
sims <- internalize_fst(ds)
head(sims)

---

chunk_data_frame **Chunk a data frame**

**Description**

Use `chunk_by_id` to split up a data set by the ID column; use `chunk_by_row` split a data set by rows.

**Usage**

\[
\text{chunk_by_id}(\text{data}, \text{nchunk}, \text{id_col} = \text{"ID"}, \text{mark} = \text{NULL})
\]

\[
\text{chunk_by_cols}(\text{data}, \text{nchunk}, \text{cols}, \text{mark} = \text{NULL})
\]

\[
\text{chunk_by_row}(\text{data}, \text{nchunk}, \text{mark} = \text{NULL})
\]

**Arguments**

- `data` A data frame.
- `nchunk` The number of chunks.
- `id_col` Character name specifying the column containing the ID for chunking.
- `mark` When populated as a character label, adds a column to the chunked data frames with that name and with value the integer group number.
- `cols` A character vector of columns to use for deriving ID to use for chunking.

**Value**

A list of data frames.
Examples

```r
x <- expand.grid(ID = 1:10, B = rev(1:10))
chunk_by_id(x, nchunk = 3)
chunk_by_row(x, nchunk = 4)
```

---

**ext_stream**

*Set or change the file extension on file_stream names*

Description

Add or update the file extension for items in a `file_stream` object. If a file extension exists, it is removed first.

Usage

```r
ext_stream(x, ext)
```

Arguments

- `x` A `file_stream` object.
- `ext` The new extension.

See Also

`format_stream()`, `locate_stream()`, `new_stream()`, `file_stream()`, `file_set()`

Examples

```r
x <- new_stream(3)
x <- ext_stream(x, "feather")
x[[1]]$file
```
file_set

Generate a sequence of file objects

Description

File names have a numbered core that communicates the current file number as well as the total number of files in the set. For example, 02-20 would indicate the second file in a set of 20. Other customizations can be added.

Usage

file_set(n, where = NULL, prefix = NULL, pad = TRUE, sep = "-", ext = ")

Arguments

n The number of file names to create.
where An optional output file path.
prefix A character prefix for the file name.
pad If TRUE, numbers will be padded with zeros.
sep Separator character.
ext A file extension, including the dot.

Value

By default a list length n of lists length 2; each sublist contains the integer file number as i and the file name as file.

See Also

setup_locker()

Examples

x <- file_set(3, where = "foo/bar")
length(x)
x[2]

x <- file_set(25, ext = ".feather")
x[17]
file_stream

Create a stream of files

Description

Optionally, setup a locker storage space on disk with a specific file format (e.g. fst or feather).

Usage

file_stream(n, locker = NULL, format = NULL, where = NULL, ...)

Arguments

- **n**: The number of file names to generate; must be a single numeric value greater than or equal to 1.
- **locker**: Passed to `setup_locker()` as dir; important to note that the directory will be unlinked if it exists and is an established locker directory.
- **format**: Passed to `format_stream()`.
- **where**: An optional file path; this is replaced by locker if it is also passed.
- **...**: Additional arguments passed to `file_set()`.

Details

Pass locker to set up locker space for saving outputs; this involves clearing the locker directory (see `setup_locker()` for details). Passing locker also sets the path for output files. If you want to set up the path for output files without setting up locker space, pass where.

See Also

- `format_stream()`, `locate_stream()`, `ext_stream()`, `new_stream()`, `file_set()`

Examples

```r
x <- file_stream(3, locker = temp_ds("foo"), format = "fst")
x[[1]]
```
**format_is_set**  
*Check format status of file set item*

**Description**

This can be used to check if a file set item has been assigned an output format (e.g. `fst`, `feather`, `qs` or `rds`). If the check returns `FALSE` it would signal that data should be returned rather than calling `write_stream()`.

**Usage**

```r
format_is_set(x)
```

```r
is.stream_format(x)
```

**Arguments**

- `x`  
  An object, usually a `file_set_item`.

**Value**

Logical indicating if `x` inherits from one of the stream format classes.

---

**format_stream**  
*Set the format for a stream_file object*

**Description**

The format is set on the file objects inside the list so that the file object can be used to call a `write` method. See `write_stream()`.

**Usage**

```r
format_stream(
  x,
  type = c("fst", "feather", "qs", "rds"),
  set_ext = TRUE,
  warn = FALSE
)
```
head_fst

Arguments

- **x**: A file_stream object.
- **type**: The file format type; if feather is chosen, then a check will be made to ensure the arrow package is loaded.
- **set_ext**: If TRUE, the existing extension (if it exists) is stripped and a new extension is added based on the value of type.
- **warn**: If TRUE a warning will be issued in case the output format is set but there is no directory path associated with the file spot in x[[1]].

Value

x is returned with a new class attribute reflecting the expected output format (fst, feather (arrow), qs or rds).

See Also

- format_is_set(), locate_stream(), ext_stream(), new_stream(), file_stream(), file_set()

Examples

```r
fs <- new_stream(2)
fs <- format_stream(fs, "fst")
fs[[1]]
format_is_set(fs[[1]])
```

head_fst  Get the head of an fst file set

Description

Get the head of an fst file set

Usage

head_fst(path, n = 5, i = 1)

Arguments

- **path**: The directory to search.
- **n**: Number of rows to show.
- **i**: Which output output chunk to show.

See Also

- get_fst(), list_fst()
internalize_fst

Get the contents of an fst file set

Description
Get the contents of an fst file set

Usage
internalize_fst(path, .as_list = FALSE, ...)

get_fst(path, .as_list = FALSE, ...)

Arguments
path
  The directory to search.
.as_list
  Should the results be returned as a list (TRUE) or a tibble (FALSE).
...
  Not used.

See Also
list_fst(), head_fst()

is.file_set_item
Check if an object is a file_set_item

Description
Check if an object is a file_set_item

Usage
is.file_set_item(x)

Arguments
x
  An object.

Value
Logical value indicating if x has the file_set_item attribute set.

Examples
x <- new_stream(2)
is.file_set_item(x[[2]])
is.file_stream  
*Check if an object inherits from file_stream*

**Description**
Check if an object inherits from file_stream

**Usage**
is.file_stream(x)

**Arguments**
x  
An object.

**Value**
Logical value indicating if x inherits from file_stream.

**Examples**
x <- new_stream(2)
is.file_stream(x)

is.locker_stream  
*Check if an object inherits from locker_stream*

**Description**
Check if an object inherits from locker_stream

**Usage**
is.locker_stream(x)

**Arguments**
x  
An object.

**Value**
Logical value indicating if x inherits from locker_stream.

**Examples**
x <- new_stream(2, locker = temp_ds("locker-stream-example"))
is.locker_stream(x)
is_locker_dir

Check if a directory is dedicated locker space

Description
Check if a directory is dedicated locker space

Usage
is_locker_dir(where)

Arguments
where The locker location.

list_fst
List all output files in a fst file set

Description
Use the function to read all of the .fst files that were saved when bg_mrgsim_d was called and .path was passed along with .format = "fst".

Usage
list_fst(path)

Arguments
path The (full) directory path to search.

locate_stream
Set or change the directory for file_stream objects

Description
Add or update the directory location for items in a file_stream object. If a directory path already exists, it is removed first.

Usage
locate_stream(x, where, initialize = FALSE)
Arguments

- **x**: A file_stream object.
- **where**: The new location.
- **initialize**: If TRUE, then the where directory is passed to a call to `reset_locker()`.

Details

When `initialize` is set to TRUE, the locker space is initialized or reset. In order to initialize, where must not exist or it must have been previously set up as locker space. See `setup_locker()` for details.

See Also

- `format_stream()`, `ext_stream()`, `new_stream()`, `file_stream()`, `file_set()`

Examples

```r
x <- new_stream(5)
x <- locate_stream(x, file.path(tempdir(), "foo"))
x[[1]]$file
```

---

**mrgsim.parallel**  
*Simulate with 'mrgsolve' in Parallel*

**Description**

Simulate with 'mrgsolve' in Parallel

**Package options**

- `mrgsim.parallel.mc.able`: if TRUE, multicore will be used if appropriate.

---

**mrgsim_ms**  
*Run mrgsim after trying to load the shared object*

**Description**

Use this function when running mrgsolve while parallelizing on a multisession worker node where the model dll might not be loaded.

**Usage**

```r
mrgsim_ms(mod, ...)
mrgsim_worker(mod, ...)
```
**new_stream**

**Arguments**

- **mod**
  - a model object
- **...**
  - passed to `mrgsolve::mrgsim()`

**Examples**

```r
mrgsim_worker(mrgsolve::house())
```

---

**new_stream**  
Create a stream of outputs and inputs

**Description**

By stream we mean a list that pre-specifies the output file names, replicate numbers and possibly input objects for a simulation. Passing `locker` initiates a call to `setup_locker()`, which sets up or resets the output directories.

For the `data.frame` method, the data are chunked into a list by columns listed in `cols`. Ideally, this is a single column that operates as a unique ID across the data set and is used by `chunk_by_id()` to form the chunks. Alternatively, `cols` can be multiple column names which are pasted together to form a unique ID that is used for splitting via `chunk_by_cols()`.

**Usage**

```r
new_stream(x, ...)
```

- **x**
  - A list or vector to template the stream; for the `numeric` method, passing a single number will fill `x` with a sequence of that length.
- **...**
  - Additional arguments passed to `file_set()`.
- **locker**
  - Passed to `setup_locker()` as `dir`; important to note that the directory will be unlinked if it exists and is an established locker directory.
Description

This function removes the hidden locker file which designates a directory as a locker. Once the locker is modified this way, it cannot be reset again by calling setup_locker() or new_stream().

Usage

noreset_locker(where)
parallel_mrgsim_d

Arguments

where

The locker location.

Value

A logical value indicating if write ability was successfully revoked.

See Also

setup_locker(), reset_locker(), version_locker()

parallel_mrgsim_d Simulate a data set in parallel

Description

Use future_mrgsim_d() to simulate with the future package. Use mc_mrgsim_d() to simulate with parallel::mclapply.

Usage

future_mrgsim_d(
  mod,
  data,
  nchunk = 4,
  ...,
  .as_list = FALSE,
  .p = NULL,
  .dry = FALSE,
  .seed = TRUE,
  .parallel = TRUE
)

mc_mrgsim_d(
  mod,
  data,
  nchunk = 4,
  ...,
  .as_list = FALSE,
  .p = NULL,
  .dry = FALSE,
  .seed = NULL,
  .parallel = TRUE
)

fu_mrgsim_d(
  mod,
parallel_mrgsim_d

data, nchunk = 4, ...
.as_list = FALSE, .p = NULL, .dry = FALSE, .seed = TRUE, .parallel = TRUE)

fu_mrgsim_d0(..., .dry = TRUE)

Arguments

mod The mrgsolve model object see mrgsolve::mrgmod.
data Data set to simulate; see mrgsolve::data_set()
nchunk Number of chunks in which to split the data set ...
.as_list If TRUE a list is return; otherwise (default) a data frame .p Post processing function executed on the worker; arguments should be (1) the simulated output (2) the model object.
.dry If TRUE neither the simulation nor the post processing will be done.
.seed Passed to future_lapply() as future.seed.
.parallel if FALSE, the simulation will not be parallelized; this is intended for debugging and testing use only.

Value

A data frame or list of simulated data.

See Also

future_mrgsim_ei()

Examples

mod <- mrgsolve::house()
data <- mrgsolve::expand.ev(amt = seq(10))
out <- future_mrgsim_d(mod, data, nchunk = 2)
Simulate an idata set in parallel

Description

Use `future_mrgsim_ei` to simulate with the `future` package. Use `mc_mrgsim_ei` to simulate with `parallel::mclapply`.

Usage

```r
future_mrgsim_ei(
  mod,
  events,
  idata,
  nchunk = 4,
  ..., .as_list = FALSE, .p = NULL, .dry = FALSE, .seed = TRUE, .parallel = TRUE
)
```

```r
definefu_mrgsim_ei(
  mod,
  events,
  idata,
  nchunk = 4,
  ..., .as_list = FALSE, .p = NULL, .dry = FALSE, .seed = TRUE, .parallel = TRUE
)
```

```r
definefu_mrgsim_ei0(..., .dry = TRUE)
```

```r
mc_mrgsim_ei(
  mod,
  events,
  idata,
  nchunk = 4,
  ..., .as_list = FALSE, .p = NULL, .dry = FALSE,
)
```
Arguments

- **mod**: The mrgsolve model object see `mrgsolve::mrgmod`.
- **events**: An event object from mrgsolve; see `mrgsolve::ev()`.
- **idata**: An idata set of parameters, one per simulation unit (individual); see `mrgsolve::idata_set()`.
- **nchunk**: Number of chunks in which to split the data set.
- **...**: Passed to `mrgsim_d()`.
- **.as_list**: If TRUE a list is return; otherwise (default) a data frame.
- **.p**: Post processing function executed on the worker; arguments should be (1) the simulated output (2) the model object.
- **.dry**: If TRUE neither the simulation nor the post processing will be done.
- **.seed**: Passed to `future_lapply()` as `future.seed`.
- **.parallel**: if FALSE, the simulation will not be parallelized; this is intended for debugging and testing use only.

Value

A data frame or list of simulated data.

See Also

- `future_mrgsim_ei`

Examples

```r
mod <- mrgsolve::house()

events <- mrgsolve::ev(amt = 100)

idata <- data.frame(CL = runif(10, 0.5, 1.5))

out <- future_mrgsim_ei(mod, events, idata)
```
reset_locker

Initialize the locker directory

Description

This function is called by setup_locker() to initialize and re-initialize a locker directory. We call it reset_locker because it is expected that the locker space is created once and then repeatedly reset and simulations are run and re-run.

Usage

reset_locker(where, pattern = NULL)

Arguments

where
The full path to the locker.

pattern
A regular expression for finding files to clear from the locker directory.

Details

For the locker space to be initialized, the where directory must not exist; if it exists, there will be an error. It is also an error for where to exist and not contain a particular hidden locker file name that marks the directory as established locker space.

NOTE: when the locker is reset, all contents are cleared according to the files matched by pattern. If any un-matched files exist after clearing the directory, a warning will be issued.

See Also

setup_locker(), noreset_locker(), version_locker()

setup_locker

Set up a data storage locker

Description

A locker is a directory structure where an enclosing folder contains subfolders that in turn contain the results of different simulation runs. When the number of simulation result sets is known, a stream of file names is returned. This function is mainly called by other functions; an exported function and documentation is provided in order to better communicate how the locker works.

Usage

setup_locker(where, tag = locker_tag(where))
temp_ds

Arguments

where
   The directory that contains tagged directories of run results.
tag
   The name of a folder under where; this directory must not exist the first time the locker is set up and **will be deleted** and re-created each time it is used to store output from a new simulation run.

Details

where must exist when setting up the locker. The directory tag will be created under where and must not exist except if it had previously been set up using setup_locker. Existing tag directories will have a hidden file in them indicating that they are established simulation output folders.

When recreating the tag directory, it will be unlinked and created new. To not try to set up a locker directory that already contains outputs that need to be preserved. You can call noreset_locker() on that directory to prevent future resets.

Value

The locker location.

See Also

reset_locker(), noreset_locker(), version_locker()

Examples

```r
x <- setup_locker(tempdir(), tag = "my-sims")
x
```

---

temp_ds  
*Create a path to a dataset in tempdir*

Description

Create a path to a dataset in tempdir

Usage

temp_ds(tag)

Arguments

tag
   The dataset subdirectory.
version_locker

Description
Version locker contents

Usage
version_locker(where, version = "save", overwrite = FALSE, noreset = FALSE)

Arguments
where
The locker location.
version
A tag to be appended to where for creating a backup of the locker contents.
overwrite
If TRUE, the new location will be removed with unlink() if it exists.
noreset
If TRUE, noreset_locker() is called on the new version.

Value
A logical value indicating whether or not all files were successfully copied to the backup, invisibly.

See Also
reset_locker(), noreset_locker(), setup_locker()

Examples
locker <- file.path(tempdir(), "version-locker-example")
if(dir.exists(locker)) unlink(locker, recursive = TRUE)
x <- new_stream(1, locker = locker)
cat("test", file = file.path(locker, "1-1"))
dir.exists(locker)
list.files(locker, all.files = TRUE)
y <- version_locker(locker, version = "y")
y
list.files(y, all.files = TRUE)
write_stream

Writer functions for stream_file objects

Description
This function will write out objects that have been assigned a format with either format_stream() or the format argument to new_stream(). See examples.

Usage
write_stream(x, ...)

## Default S3 method:
write_stream(x, data, ...)

## S3 method for class 'stream_format_fst'
write_stream(x, data, dir = NULL, ...)

## S3 method for class 'stream_format_feather'
write_stream(x, data, dir = NULL, ...)

## S3 method for class 'stream_format_qs'
write_stream(x, data, dir = NULL, ...)

## S3 method for class 'stream_format_rds'
write_stream(x, data, dir = NULL, ...)

Arguments
x A file_stream object.
... Not used.
data An object to write.
dir An optional directory location to be used if not already in the file spot in x.

Details
The default method always returns FALSE; other methods which get invoked if a format was set will return TRUE. So, the user can always call write_stream() and check the return value: if TRUE, the file was written to disk and the data to not need to be returned; a FALSE return value indicates that no format was set and the data should be returned.

Note the write methods can be invoked directly for a specific format if no format was set (see examples).

Value
A logical value indicating if the output was written or not.
write_stream

See Also

format_stream(), ext_stream(), locate_stream(), new_stream(), file_stream()

Examples

d <- temp_ds("example")

fs <- new_stream(2, locker = ds, format = "fst")

data <- data.frame(x = rnorm(10))

x <- lapply(fs, write_stream, data = data)

list.files(ds)

reset_locker(ds)

fs <- format_stream(fs, "rds")

x <- lapply(fs, write_stream, data = data)

list.files(ds)
Index

arrow::open_dataset(), 4
bg_mclapply, 2
bg_mrgsim_d, 3
bg_mrgsim_d(), 4
callr::r_bg(), 3, 4
chunk_by_cols (chunk_data_frame), 5
chunk_by_cols(), 15
chunk_by_id, 5
chunk_by_id (chunk_data_frame), 5
chunk_by_id(), 15
chunk_by_row, 5
chunk_by_row (chunk_data_frame), 5
chunk_data_frame, 5
dplyr::bind_rows(), 4
ext_stream, 6
ext_stream(), 8, 10, 14, 16, 25
file_set, 7
file_set(), 6, 8, 10, 14–16
file_stream, 8
file_stream(), 6, 10, 14, 16, 25
format_is_set, 9
format_is_set(), 10
format_stream, 9
format_stream(), 6, 8, 14, 16, 24, 25
fst::read_fst(), 4
fu_mrgsim_d (parallel_mrgsim_d), 17
fu_mrgsim_d0 (parallel_mrgsim_d), 17
fu_mrgsim_ei (parallel_mrgsim_ei), 19
fu_mrgsim_ei0 (parallel_mrgsim_ei), 19
future::plan(), 4
future_lapply(), 18, 20
future_mclapply (parallel_mrgsim_d), 17
future_mclapply(), 4, 17
future_mrgsim_ei, 19, 20
future_mrgsim_ei (parallel_mrgsim_ei), 19
future_mrgsim_ei(), 18
get_fst (initialize_fst), 11
get_fst(), 10
head_fst, 10
head_fst(), 4, 11
initialize_fst, 11
initialize_fst(), 4
is.file_set_item, 11
is.file_stream, 12
is.locker_stream, 12
is.stream_format (format_is_set), 9
is_locker_dir, 13
list_fst, 13
list_fst(), 4, 10, 11
locate_stream, 13
locate_stream(), 6, 8, 10, 16, 25
mc_mrgsim_d (parallel_mrgsim_d), 17
mc_mrgsim_d(), 17
mc_mrgsim_ei, 19
mc_mrgsim_ei (parallel_mrgsim_ei), 19
mrgsim.parallel, 14
mrgsim_d(), 18, 20
mrgsim_ms, 14
mrgsim_worker (mrgsim_ms), 14
mrgsolve::data_set(), 3, 18
mrgsolve::ev(), 20
mrgsolve::idata_set(), 20
mrgsolve::mrgmod, 18, 20
mrgsolve::mrgsim(), 3, 15
new_stream, 15
new_stream(), 6, 8, 10, 14, 16, 24, 25
noreset_locker, 16
noreset_locker(), 21–23
parallel::mclapply(), 2

26
INDEX

parallel_mrgsim_d, 17
parallel_mrgsim_ei, 19
processx::process, 4

reset_locker, 21
reset_locker(), 14, 17, 22, 23

setup_locker, 21
setup_locker(), 4, 7, 8, 14–17, 21, 23

temp_ds, 22

unlink(), 23

version_locker, 23
version_locker(), 17, 21, 22

write_stream, 24
write_stream(), 9