# Package 'jobqueue'

May 24, 2025

**Type** Package

**Title** Run Interruptible Code Asynchronously

**Version** 1.7.0

**Date** 2025-05-23

**Description** Takes an R expression and returns a job object with a $stop() method which can be called to terminate the background job. Also provides timeouts and other mechanisms for automatically terminating a background job. The result of the expression is available synchronously via $result or asynchronously with callbacks or through the 'promises' package framework.

**URL** https://cmmr.github.io/jobqueue/, https://github.com/cmmr/jobqueue

**BugReports** https://github.com/cmmr/jobqueue/issues

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**Config/Needs/website** rmarkdown

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**Depends** R (>= 4.2.0)

**Imports** cli, interprocess (>= 1.2.0), later, magrittr, parallelly, promises, ps, R6, rlang, utils

**Suggests** glue, knitr, rmarkdown, testthat

**NeedsCompilation** no

**Author** Daniel P. Smith [aut, cre] (ORCID: <https://orcid.org/0000-0002-2479-2044>), Alkek Center for Metagenomics and Microbiome Research [cph, fnd]

**Maintainer** Daniel P. Smith <dansmith01@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-05-23 23:22:01 UTC

# Contents



---

`jobqueue` *Assigns Jobs to Workers*

---

## Description

Jobs go in. Results come out.

## Usage

```
jobqueue(
  globals = NULL,
  packages = NULL,
  namespace = NULL,
  init = NULL,
  max_cpus = availableCores(),
  workers = ceiling(max_cpus * 1.2),
  timeout = NULL,
  hooks = NULL,
  reformat = NULL,
  signal = FALSE,
  cpus = 1L,
  stop_id = NULL,
  copy_id = NULL
)
```

## Arguments

| | |
|---|---|
| `globals` | A named list of variables that all `<job>$exprs` will have access to. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment. |
| `packages` | Character vector of package names to load on `workers`. |
| `namespace` | The name of a package to attach to the `worker's` environment. |
| `init` | A call or R expression wrapped in curly braces to evaluate on each `worker` just once, immediately after start-up. Will have access to variables defined by `globals` and assets from `packages` and `namespace`. Returned value is ignored. |
| `max_cpus` | Total number of CPU cores that can be reserved by all running `jobs` (`sum(<job>$cpus)`). Does not enforce limits on actual CPU utilization. |

workers — How many background worker processes to start. Set to more than max_cpus to enable standby workers to quickly swap out with workers that need to restart.

timeout — A named numeric vector indicating the maximum number of seconds allowed for each state the job passes through, or 'total' to apply a single timeout from 'submitted' to 'done'. Can also limit the 'starting' state for workers. A function (job) can be used in place of a number. Example: timeout = c(total = 2.5, running = 1). See vignette('stops').

hooks — A named list of functions to run when the job state changes, of the form hooks = list(created = function (worker) {...}). Or a function (job) that returns the same. Names of worker hooks are typically 'created', 'submitted', 'queued', 'dispatched', 'starting', 'running', 'done', or '*' (duplicates okay). See vignette('hooks').

reformat — Set reformat = function (job) to define what <job>$result should return. The default, reformat = NULL passes <job>$output to <job>$result unchanged. See vignette('results').

signal — Should calling <job>$result signal on condition objects? When FALSE, <job>$result will return the object without taking additional action. Setting to TRUE or a character vector of condition classes, e.g. c('interrupt', 'error', 'warning'), will cause the equivalent of stop(<condition>) to be called when those conditions are produced. Alternatively, a function (job) that returns TRUE or FALSE. See vignette('results').

cpus — The default number of CPU cores per job. Or a function (job) that returns the number of CPU cores to reserve for a given job. Used to limit the number of jobs running simultaneously to respect <jobqueue>$max_cpus. Does not prevent a job from using more CPUs than reserved.

stop_id — If an existing job in the jobqueue has the same stop_id, that job will be stopped and return an 'interrupt' condition object as its result. stop_id can also be a function (job) that returns the stop_id to assign to a given job. A stop_id of NULL disables this feature. See vignette('stops').

copy_id — If an existing job in the jobqueue has the same copy_id, the newly submitted job will become a "proxy" for that earlier job, returning whatever result the earlier job returns. copy_id can also be a function (job) that returns the copy_id to assign to a given job. A copy_id of NULL disables this feature. See vignette('stops').

## Value

A jobqueue object.

## Examples

```
jq <- jobqueue(globals = list(N = 42), workers = 2)
print(jq)

job <- jq$run({ paste("N is", N) })
job$result
```

```
jq$stop()
```

---

`jobqueue_class` *Assigns Jobs to Workers (R6 Class)*

---

## Description

Jobs go in. Results come out.

## Active bindings

`hooks` A named list of currently registered callback hooks.

`jobs` Get or set - List of jobs currently managed by this `jobqueue`.

`state` The `jobqueue's` state: `'starting'`, `'idle'`, `'busy'`, `'stopped'`, or `'error.'`

`uid` A short string, e.g. `'Q1'`, that uniquely identifies this `jobqueue`.

`tmp` The `jobqueue`'s temporary directory.

`workers` Get or set - List of `workers` used for processing `jobs`.

`cnd` The error that caused the `jobqueue` to stop.

## Methods

### Public methods:

- `jobqueue_class$new()`
- `jobqueue_class$print()`
- `jobqueue_class$run()`
- `jobqueue_class$submit()`
- `jobqueue_class$wait()`
- `jobqueue_class$on()`
- `jobqueue_class$stop()`

**Method** `new()`**:** Creates a pool of background processes for handling `$run()` and `$submit()` calls. These `workers` are initialized according to the `globals`, `packages`, and `init` arguments.

*Usage:*

```
jobqueue_class$new(
  globals = NULL,
  packages = NULL,
  namespace = NULL,
  init = NULL,
  max_cpus = availableCores(),
  workers = ceiling(max_cpus * 1.2),
  timeout = NULL,
  hooks = NULL,
```

```
  reformat = NULL,
  signal = FALSE,
  cpus = 1L,
  stop_id = NULL,
  copy_id = NULL
)
```

*Arguments:*

`globals` A named list of variables that all `<job>$exprs` will have access to. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment.

`packages` Character vector of package names to load on `workers`.

`namespace` The name of a package to attach to the `worker's` environment.

`init` A call or R expression wrapped in curly braces to evaluate on each `worker` just once, immediately after start-up. Will have access to variables defined by `globals` and assets from `packages` and `namespace`. Returned value is ignored.

`max_cpus` Total number of CPU cores that can be reserved by all running `jobs` (`sum(<job>$cpus)`). Does not enforce limits on actual CPU utilization.

`workers` How many background `worker` processes to start. Set to more than `max_cpus` to enable standby `workers` to quickly swap out with `workers` that need to restart.

`timeout, hooks, reformat, signal, cpus, stop_id, copy_id` Defaults for this `jobqueue's` `$run()` method. Here only, `stop_id` and `copy_id` must be either a `function (job)` or `NULL`. `hooks` can set `jobqueue`, `worker`, and/or `job` hooks - see the "Attaching" section in `vignette('hooks')`.

*Returns:* A `jobqueue` object.

**Method** `print()`**:** Print method for a `jobqueue`.

*Usage:*

```
jobqueue_class$print(...)
```

*Arguments:*

`...` Arguments are not used currently.

**Method** `run()`**:** Creates a `job` object and submits it to the `jobqueue` for running. Any `NA` arguments will be replaced with their value from `jobqueue_class$new()`.

*Usage:*

```
jobqueue_class$run(
  expr,
  vars = list(),
  timeout = NA,
  hooks = NA,
  reformat = NA,
  signal = NA,
  cpus = NA,
  stop_id = NA,
  copy_id = NA,
  ...
)
```

*Arguments:*

`expr` A call or R expression wrapped in curly braces to evaluate on a `worker`. Will have access to any variables defined by `vars`, as well as the `jobqueue's` `globals`, `packages`, and `init` configuration. See `vignette('eval')`.

`vars` A named list of variables to make available to `expr` during evaluation. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment. Or a `function (job)` that returns such an object.

`timeout` A named numeric vector indicating the maximum number of seconds allowed for each state the `job` passes through, or 'total' to apply a single timeout from 'submitted' to 'done'. Can also limit the 'starting' state for `workers`. A `function (job)` can be used in place of a number. Example: `timeout = c(total = 2.5, running = 1)`. See `vignette('stops')`.

`hooks` A named list of functions to run when the `job` state changes, of the form `hooks = list(created = function (worker) {...})`. Or a `function (job)` that returns the same. Names of `worker` hooks are typically `'created'`, `'submitted'`, `'queued'`, `'dispatched'`, `'starting'`, `'running'`, `'done'`, or `'*'` (duplicates okay). See `vignette('hooks')`.

`reformat` Set `reformat = function (job)` to define what `<job>$result` should return. The default, `reformat = NULL` passes `<job>$output` to `<job>$result` unchanged. See `vignette('results')`.

`signal` Should calling `<job>$result` signal on condition objects? When `FALSE`, `<job>$result` will return the object without taking additional action. Setting to `TRUE` or a character vector of condition classes, e.g. `c('interrupt', 'error', 'warning')`, will cause the equivalent of `stop(<condition>)` to be called when those conditions are produced. Alternatively, a `function (job)` that returns `TRUE` or `FALSE`. See `vignette('results')`.

`cpus` How many CPU cores to reserve for this `job`. Or a `function (job)` that returns the same. Used to limit the number of `jobs` running simultaneously to respect `<jobqueue>$max_cpus`. Does not prevent a `job` from using more CPUs than reserved.

`stop_id` If an existing `job` in the `jobqueue` has the same `stop_id`, that `job` will be stopped and return an 'interrupt' condition object as its result. `stop_id` can also be a `function (job)` that returns the `stop_id` to assign to a given `job`. A `stop_id` of `NULL` disables this feature. See `vignette('stops')`.

`copy_id` If an existing `job` in the `jobqueue` has the same `copy_id`, the newly submitted `job` will become a "proxy" for that earlier `job`, returning whatever result the earlier `job` returns. `copy_id` can also be a `function (job)` that returns the `copy_id` to assign to a given `job`. A `copy_id` of `NULL` disables this feature. See `vignette('stops')`.

`...` Arbitrary named values to add to the returned `job` object.

*Returns:* The new `job` object.

**Method** `submit()`**:** Adds a `job` to the `jobqueue` for running on a background process.

*Usage:*

```
jobqueue_class$submit(job)
```

*Arguments:*

`job` A `job` object, as created by `job_class$new()`.

*Returns:* This `jobqueue`, invisibly.

**Method** `wait()`**:** Blocks until the `jobqueue` enters the given state.

*Usage:*

```
jobqueue_class$wait(state = "idle", timeout = NULL, signal = TRUE)
```

*Arguments:*

`state` The name of a `jobqueue` state. Typically one of:

- `'*'` - Every time the state changes.
- `'.next'` - Only one time, the next time the state changes.
- `'starting'` - `workers` are starting.
- `'idle'` - All `workers` are ready/idle.
- `'busy'` - At least one `worker` is busy.
- `'stopped'` - Shutdown is complete.

`timeout` Stop the `jobqueue` if it takes longer than this number of seconds, or `NULL`.

`signal` Raise an error if encountered (will also be recorded in `<jobqueue>$cnd`).

*Returns:* This `jobqueue`, invisibly.

**Method** `on()`**:** Attach a callback function to execute when the `jobqueue` enters `state`.

*Usage:*

```
jobqueue_class$on(state, func)
```

*Arguments:*

`state` The name of a `jobqueue` state. Typically one of:

- `'*'` - Every time the state changes.
- `'.next'` - Only one time, the next time the state changes.
- `'starting'` - `workers` are starting.
- `'idle'` - All `workers` are ready/idle.
- `'busy'` - At least one `worker` is busy.
- `'stopped'` - Shutdown is complete.

`func` A function that accepts a `jobqueue` object as input. Return value is ignored.

*Returns:* A function that when called removes this callback from the `jobqueue`.

**Method** `stop()`**:** Stop all `jobs` and `workers`.

*Usage:*

```
jobqueue_class$stop(reason = "jobqueue shut down by user", cls = NULL)
```

*Arguments:*

`reason` Passed to `<job>$stop()` for any `jobs` currently managed by this `jobqueue`.

`cls` Passed to `<job>$stop()` for any `jobs` currently managed by this `jobqueue`.

*Returns:* This `jobqueue`, invisibly.

job_class *Define an R Expression (R6 Class)*

## Description

The job object encapsulates an expression and its evaluation parameters. It also provides a way to check for and retrieve the result.

## Active bindings

`expr` R expression that will be run by this job.

`vars` Get or set - List of variables that will be placed into the expression's environment before evaluation.

`reformat` Get or set - `function (job)` for defining `<job>$result`.

`signal` Get or set - Conditions to signal.

`cpus` Get or set - Number of CPUs to reserve for evaluating `expr`.

`timeout` Get or set - Time limits to apply to this job.

`proxy` Get or set - job to proxy in place of running `expr`.

`state` Get or set - The job's state: `'created'`, `'submitted'`, `'queued'`, `'dispatched'`, `'starting'`, `'running'`, or `'done'`. *Assigning to* `<job>$state` *will trigger callback hooks.*

`output` Get or set - job's raw output. *Assigning to* `<job>$output` *will change the job's state to* `'done'`.

`jobqueue` The jobqueue that this job belongs to.

`worker` The worker that this job belongs to.

`result` Result of `expr`. Will block until job is finished.

`hooks` Currently registered callback hooks as a named list of functions. Set new hooks with `<job>$on()`.

`is_done` `TRUE` or `FALSE` depending on if the job's result is ready.

`uid` A short string, e.g. `'J16'`, that uniquely identifies this job.

## Methods

### Public methods:

- job_class$new()
- job_class$print()
- job_class$on()
- job_class$wait()
- job_class$stop()

**Method** `new()`**:** Creates a job object defining how to run an expression on a background worker process.
*Typically you won't need to call* `job_class$new()`. *Instead, create a* jobqueue *and use* `<jobqueue>$run()` *to generate* job *objects.*

*Usage:*

```
job_class$new(
  expr,
  vars = NULL,
  timeout = NULL,
  hooks = NULL,
  reformat = NULL,
  signal = FALSE,
  cpus = 1L,
  ...
)
```

*Arguments:*

`expr` A call or R expression wrapped in curly braces to evaluate on a `worker`. Will have access to any variables defined by `vars`, as well as the `worker's` `globals`, `packages`, and `init` configuration. See `vignette('eval')`.

`vars` A named list of variables to make available to `expr` during evaluation. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment. Or a `function (job)` that returns such an object.

`timeout` A named numeric vector indicating the maximum number of seconds allowed for each state the `job` passes through, or 'total' to apply a single timeout from 'submitted' to 'done'. Or a `function (job)` that returns the same. Example: `timeout = c(total = 2.5, running = 1)`. See `vignette('stops')`.

`hooks` A named list of functions to run when the `job` state changes, of the form `hooks = list(created = function (worker) {...})`. Or a `function (job)` that returns the same. Names of `worker` hooks are typically `'created'`, `'submitted'`, `'queued'`, `'dispatched'`, `'starting'`, `'running'`, `'done'`, or `'*'` (duplicates okay). See `vignette('hooks')`.

`reformat` Set `reformat = function (job)` to define what `<job>$result` should return. The default, `reformat = NULL` passes `<job>$output` to `<job>$result` unchanged. See `vignette('results')`.

`signal` Should calling `<job>$result` signal on condition objects? When `FALSE`, `<job>$result` will return the object without taking additional action. Setting to `TRUE` or a character vector of condition classes, e.g. `c('interrupt', 'error', 'warning')`, will cause the equivalent of `stop(<condition>)` to be called when those conditions are produced. Alternatively, a `function (job)` that returns `TRUE` or `FALSE`. See `vignette('results')`.

`cpus` How many CPU cores to reserve for this `job`. Or a `function (job)` that returns the same. Used to limit the number of `jobs` running simultaneously to respect `<jobqueue>$max_cpus`. Does not prevent a `job` from using more CPUs than reserved.

`...` Arbitrary named values to add to the returned `job` object.

*Returns:* A `job` object.

**Method** `print()`**:** Print method for a `job`.

*Usage:*

```
job_class$print(...)
```

*Arguments:*

`...` Arguments are not used currently.

*Returns:* This job, invisibly.

**Method** `on()`**:** Attach a callback function to execute when the job enters `state`.

*Usage:*

```
job_class$on(state, func)
```

*Arguments:*

`state` The name of a job state. Typically one of:

- `'*'` - Every time the state changes.
- `'.next'` - Only one time, the next time the state changes.
- `'created'` - After `job_class$new()` initialization.
- `'submitted'` - After `<job>$jobqueue` is assigned.
- `'queued'` - After `stop_id` and `copy_id` are resolved.
- `'dispatched'` - After `<job>$worker` is assigned.
- `'starting'` - Before evaluation begins.
- `'running'` - After evaluation begins.
- `'done'` - After `<job>$output` is assigned.

Custom states can also be specified.

`func` A function that accepts a job object as input. You can call `<job>$stop()` or edit `<job>$` values and the changes will be persisted (since jobs are reference class objects). You can also edit/stop other queued jobs by modifying the jobs in `<job>$jobqueue$jobs`. Return value is ignored.

*Returns:* A function that when called removes this callback from the job.

**Method** `wait()`**:** Blocks until the job enters the given state.

*Usage:*

```
job_class$wait(state = "done", timeout = NULL)
```

*Arguments:*

`state` The name of a job state. Typically one of:

- `'*'` - Every time the state changes.
- `'.next'` - Only one time, the next time the state changes.
- `'created'` - After `job_class$new()` initialization.
- `'submitted'` - After `<job>$jobqueue` is assigned.
- `'queued'` - After `stop_id` and `copy_id` are resolved.
- `'dispatched'` - After `<job>$worker` is assigned.
- `'starting'` - Before evaluation begins.
- `'running'` - After evaluation begins.
- `'done'` - After `<job>$output` is assigned.

Custom states can also be specified.

`timeout` Stop the job if it takes longer than this number of seconds, or `NULL`.

*Returns:* This job, invisibly.

**Method** `stop()`**:** Stop this job. If the job is running, its worker will be restarted.

*Usage:*

```
job_class$stop(reason = "job stopped by user", cls = NULL)
```

*Arguments:*

`reason` A message to include in the 'interrupt' condition object that will be returned as the `job's` result. Or a condition object.

`cls` Character vector of additional classes to prepend to `c('interrupt', 'condition')`.

*Returns:* This `job`, invisibly.

---

`worker_class` *A Background Process (R6 Class)*

---

## Description

Where job expressions are evaluated.

## Active bindings

`hooks` A named list of currently registered callback hooks.

`job` The currently running `job`.

`ps` The `ps::ps_handle()` object for the background process.

`state` The `worker's` state: `'starting'`, `'idle'`, `'busy'`, or `'stopped'`.

`uid` A short string, e.g. `'W11'`, that uniquely identifies this `worker`.

`tmp` The `worker's` temporary directory.

`cnd` The error that caused the `worker` to stop.

## Methods

### Public methods:

- `worker_class$new()`
- `worker_class$print()`
- `worker_class$start()`
- `worker_class$stop()`
- `worker_class$restart()`
- `worker_class$on()`
- `worker_class$wait()`
- `worker_class$run()`

**Method** `new()`**:** Creates a background R process for running `jobs`.

*Usage:*

```
worker_class$new(
  globals = NULL,
  packages = NULL,
  namespace = NULL,
  init = NULL,
  hooks = NULL,
  wait = TRUE,
  timeout = Inf
)
```

*Arguments:*

`globals` A named list of variables that all `<job>$exprs` will have access to. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment.

`packages` Character vector of package names to load on `workers`.

`namespace` The name of a package to attach to the `worker's` environment.

`init` A call or R expression wrapped in curly braces to evaluate on each `worker` just once, immediately after start-up. Will have access to variables defined by `globals` and assets from `packages` and `namespace`. Returned value is ignored.

`hooks` A named list of functions to run when the `worker` state changes, of the form `hooks = list(idle = function (worker) {...})`. Names of `worker` hooks are typically `starting`, `idle`, `busy`, `stopped`, or `'*'` (duplicates okay). See `vignette('hooks')`.

`wait` If `TRUE`, blocks until the `worker` is 'idle'. If `FALSE`, the `worker` object is returned in the 'starting' state.

`timeout` How long to wait for the `worker` to finish starting (in seconds). If `NA`, defaults to the `worker_class$new()` argument.

*Returns:* A `worker` object.

**Method** `print()`**:** Print method for a `worker`.

*Usage:*

```
worker_class$print(...)
```

*Arguments:*

`...` Arguments are not used currently.

*Returns:* The `worker`, invisibly.

**Method** `start()`**:** Restarts a stopped `worker`.

*Usage:*

```
worker_class$start(wait = TRUE, timeout = NA)
```

*Arguments:*

`wait` If `TRUE`, blocks until the `worker` is 'idle'. If `FALSE`, the `worker` object is returned in the 'starting' state.

`timeout` How long to wait for the `worker` to finish starting (in seconds). If `NA`, defaults to the `worker_class$new()` argument.

*Returns:* The `worker`, invisibly.

**Method** `stop()`**:** Stops a `worker` by terminating the background process and calling `<job>$stop(reason)` on any `jobs` currently assigned to this `worker`.

*Usage:*

```
worker_class$stop(reason = "worker stopped by user", cls = NULL)
```

*Arguments:*

`reason` Passed to `<job>$stop()` for any `jobs` currently managed by this `worker`.

`cls` Passed to `<job>$stop()` for any `jobs` currently managed by this `worker`.

*Returns:* The `worker`, invisibly.

**Method** `restart()`**:** Restarts a `worker` by calling `<worker>$stop(reason)` and `<worker>$start()` in succession.

*Usage:*

```
worker_class$restart(
  wait = TRUE,
  timeout = NA,
  reason = "restarting worker",
  cls = NULL
)
```

*Arguments:*

`wait` If `TRUE`, blocks until the `worker` is 'idle'. If `FALSE`, the `worker` object is returned in the 'starting' state.

`timeout` How long to wait for the `worker` to finish starting (in seconds). If `NA`, defaults to the `worker_class$new()` argument.

`reason` Passed to `<job>$stop()` for any `jobs` currently managed by this `worker`.

`cls` Passed to `<job>$stop()` for any `jobs` currently managed by this `worker`.

*Returns:* The `worker`, invisibly.

**Method** `on()`**:** Attach a callback function to execute when the `worker` enters `state`.

*Usage:*

```
worker_class$on(state, func)
```

*Arguments:*

`state` The name of a `worker` state. Typically one of:

- `'*'` - Every time the state changes.
- `'.next'` - Only one time, the next time the state changes.
- `'starting'` - Waiting for the background process to load.
- `'idle'` - Waiting for `jobs` to be `$run()`.
- `'busy'` - While a `job` is running.
- `'stopped'` - After `<worker>$stop()` is called.

`func` A function that accepts a `worker` object as input. You can call `<worker>$stop()` and other `<worker>$` methods.

*Returns:* A function that when called removes this callback from the `worker`.

**Method** `wait()`**:** Blocks until the `worker` enters the given state.

*Usage:*

```
worker_class$wait(state = "idle", timeout = Inf, signal = TRUE)
```

*Arguments:*

`state` The name of a `worker` state. Typically one of:

- `'*'` - Every time the state changes.
- `'.next'` - Only one time, the next time the state changes.
- `'starting'` - Waiting for the background process to load.
- `'idle'` - Waiting for `jobs` to be `$run()`.
- `'busy'` - While a `job` is running.
- `'stopped'` - After `<worker>$stop()` is called.

`timeout` Stop the `worker` if it takes longer than this number of seconds.

`signal` Raise an error if encountered (will also be recorded in `<worker>$cnd`).

*Returns:* This `worker`, invisibly.

**Method** `run()`**:** Assigns a `job` to this `worker` for evaluation on the background process.

*Usage:*

```
worker_class$run(job)
```

*Arguments:*

`job` A `job` object, as created by `job_class$new()`.

*Returns:* This `worker`, invisibly.

# Index