

# Package: webqueue (via r-universe)

March 14, 2025

**Type** Package

**Title** Multicore HTTP Server

**Version** 1.0.0

**Date** 2025-03-11

**Description** Distributes HTTP requests among a pool of background R processes. Supports timeouts and interrupts of requests to ensure that CPU cores are utilized effectively.

**URL** <https://cmmr.github.io/webqueue/>, <https://github.com/cmmr/webqueue>

**BugReports** <https://github.com/cmmr/webqueue/issues>

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Roxygen** list(markdown = TRUE)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Depends** R (>= 4.2.0)

**Imports** cli, httpuv, jsonlite, jobqueue, later, parallelly, promises, R6, rlang, semaphore, webutils, utils

**Suggests** httr2, knitr, RCurl, rmarkdown, testthat (>= 3.0.0), withr

**Config/pak/sysreqs** make libssl-dev zlib1g-dev

**Repository** <https://cmmr.r-universe.dev>

**RemoteUrl** <https://github.com/cmmr/webqueue>

**RemoteRef** HEAD

**RemoteSha** 2591d223a8046437c7d7ce30e34f6a7005e8df7e

## Contents

cookie . . . . .	2
header . . . . .	3
js_obj . . . . .	4
response . . . . .	4
WebQueue . . . . .	5
<b>Index</b>	<b>9</b>

---

cookie	<i>Assemble an HTTP cookie.</i>
--------	---------------------------------

---

### Description

See <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie> for a more in-depth description of each parameter's purpose.

### Usage

```

cookie(
  ...,
  max_age = NULL,
  domain = NULL,
  path = NULL,
  same_site = "Lax",
  secure = FALSE,
  http_only = FALSE,
  partitioned = FALSE,
  name = ...names(),
  value = ..1
)

```

### Arguments

...	A single key-value pair.
max_age	The number of seconds until expiration. Omit to create a session cookie. Inf is mapped to 34560000L (400 days).
domain	Send with requests to this host.
path	Send with requests to this path.
same_site	'Strict', 'Lax', or 'None'. secure required for 'None'.
secure	Only send over HTTPS.
http_only	Disallow javascript access.
partitioned	Use partitioned storage. secure required.
name	Explicitly set the name (key) in the key-value pair.
value	Explicitly set the value in the key-value pair.

**Value**

A 'header' object that can be passed to `response()`.

**Examples**

```
library(webqueue)

cookie(xyz = 123, max_age = 3600, http_only = TRUE)

token <- 'randomstring123'
cookie(token)

response(cookie(token = 'randomstring123'))
```

---

header

*Assemble an HTTP header.*

---

**Description**

See [https://developer.mozilla.org/en-US/docs/Glossary/Response\\_header](https://developer.mozilla.org/en-US/docs/Glossary/Response_header) for example response headers and their purpose.

**Usage**

```
header(..., expose = FALSE, name = ...names(), value = ..1)
```

**Arguments**

<code>...</code>	A single key-value pair.
<code>expose</code>	Allow javascript to read this header.
<code>name</code>	Explicitly set the name (key) in the key-value pair.
<code>value</code>	Explicitly set the value in the key-value pair.

**Value**

A 'header' object that can be passed to `response()`.

**Examples**

```
library(webqueue)

header(name = 'Location', value = '/index.html')

Location <- '/index.html'
header(Location)

response(307L, header(Location = '/index.html'))
```

```
# Allow javascript to access a header value
header('x-user-id' = 100, expose = TRUE)
```

---

js\_obj

*Ensure a list becomes a JSON object.*

---

### Description

This function returns a list that `jsonlite::toJSON()` will always encode as `{}`.

### Usage

```
js_obj(x = list())
```

### Arguments

`x`                    A list, or list-like object.

### Value

A list with the `names` attribute set.

### Examples

```
library(webqueue)

updates <- list()

response(list(updates = updates))

response(list(updates = js_obj(updates)))
```

---

response

*Compile an HTTP response.*

---

### Description

If your `WebQueue`'s handler function returns a list, json object, character vector, or scalar integer, `response()` will be used to transform that result into an HTTP response.

You may also call `response()` within your handler to better customize the HTTP response. Or, return a result of class `'AsIs'` to have that object passed directly on to `'httpuv'`.

**Usage**

```
response(body = NULL, status = 200L, headers = NULL, ...)
```

**Arguments**

body	The content. A list will be encoded as JSON. A scalar integer will be interpreted as a status. A character vector will be concatenated with no separator.
status	A HTTP response status code.
headers	A named character vector of HTTP headers. A list-like object is acceptable if all elements are simple strings.
...	Objects created by <code>header()</code> and/or <code>cookie()</code> . Or key-value pairs to add to headers.

**Value**

A `<response/AsIs>` object. Essentially a list with elements named `body`, `status`, and `headers` formatted as `'httpuv'` expects.

**Examples**

```
library(webqueue)

response(list(name = unbox('Peter'), pi = pi))

response(307L, Location = '/new/file.html')

# The `body` and `status` slots also handle header objects.
response(cookie(id = 123, http_only = TRUE))

# Allow javascript to access custom headers.
uid <- header('x-user-id' = 100, expose = TRUE)
sid <- header('x-session-id' = 303, expose = TRUE)
response(uid, sid)
```

**Description**

Connects the `'httpuv'` and `'jobqueue'` R packages.

**Active bindings**

`ur1` URL where the server is available.

## Methods

### Public methods:

- [WebQueue\\$new\(\)](#)
- [WebQueue\\$print\(\)](#)
- [WebQueue\\$stop\(\)](#)

**Method** `new()`: Creates an `httpuv::WebServer` with requests handled by a `jobqueue::Queue`.

#### *Usage:*

```
WebQueue$new(
  handler,
  host = "0.0.0.0",
  port = 8080L,
  parse = NULL,
  globals = list(),
  packages = NULL,
  namespace = NULL,
  init = NULL,
  max_cpus = availableCores(),
  workers = ceiling(max_cpus * 1.2),
  timeout = NULL,
  hooks = NULL,
  reformat = NULL,
  stop_id = NULL,
  copy_id = NULL,
  bg = TRUE,
  quiet = FALSE,
  onHeaders = NULL,
  staticPaths = NULL,
  staticPathOptions = NULL
)
```

#### *Arguments:*

`handler` A function (request) that will be run on a background worker process. The returned value will be passed through `reformat`, then sent as the server's response to the web client.

`host` A string that is a valid IPv4 address that is owned by this server, or '0.0.0.0' to listen on all IP addresses.

`port` A number or integer that indicates the server port that should be listened on. Note that on most Unix-like systems including Linux and macOS, port numbers smaller than 1024 require root privileges.

`parse` A function (req) that is run on the foreground process to transform the HTTP request prior to passing it to handler. req is the environment object provided by 'httpuv', amended with \$ARGS and \$COOKIES. Return value is used as req going forward.

`globals` A list of variables to add to handler's evaluation 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(cpus)`). Does not enforce limits on actual CPU utilization.
- workers** How many background `jobqueue::Worker` processes to start. Set to more than `max_cpus` to enable interrupted workers to be quickly swapped out with standby Workers while a replacement Worker boots up.
- 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'. Example: `timeout = c(total = 2.5, running = 1)`.
- hooks** A list of functions to run when the Job state changes, of the form `hooks = list(created = function (job) {...}, done = ~{...})`. See `vignette('hooks')`.
- reformat** A function (`job`) that is run in the foreground process to transform the output from handler. The default, `reformat = NULL`, is essentially `function (job) { job$output }`.
- stop\_id** A function (`job`). If two Jobs generate the same value from this function, then the earlier Job will be aborted. If the returned value is `NULL`, no Jobs will be stopped.
- copy\_id** A function (`job`). If two Jobs generate the same value from this function, then the later Job will clone its output from the earlier Job. If the returned value is `NULL`, no Jobs will be cloned.
- bg** Where/how to run the server. `TRUE`: on a separate R process. `FALSE`: blocking on the current R process. `NULL`: non-blocking on the current R process.
- quiet** If `TRUE`, suppress error messages from starting the 'httpuv' server.
- onHeaders** A function (`request`) triggered when headers are received by 'httpuv'. Return `NULL` to continue normal processing of the request, or a Rook response to send that response, stop processing the request, and ask the client to close the connection. (This can be used to implement upload size limits, for example.)
- staticPaths** A named list of paths that will be served without invoking `handler()` or `onHeaders()`. The name of each one is the URL path, and the value is either a string referring to a local path, or an object created by the `httpuv::staticPath()` function.
- staticPathOptions** A set of default options to use when serving static paths. If not set or `NULL`, then it will use the result from calling `httpuv::staticPathOptions()` with no arguments.

*Returns:* A `WebQueue` object.

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

*Usage:*

```
WebQueue$print(...)
```

*Arguments:*

... Arguments are not used currently.

**Method** `stop()`: Shuts down the `WebQueue` and all associated subprocesses. Stopped Jobs will have their `$output` set to a object of class `<interrupt/condition>`

*Usage:*

```
WebQueue$stop(reason = "server stopped")
```

*Arguments:*

*reason* A brief message for the condition object.

*Returns:* This WebQueue, invisibly.

**Examples**

```
library(webqueue)
```

```
wq <- WebQueue$new(function (req) 'Hello World!\n')  
readLines(wq$url)  
wq$stop()
```

# Index

cookie, [2](#)

header, [3](#)

jobqueue::Worker, [7](#)

js\_obj, [4](#)

response, [4](#)

WebQueue, [5](#)