

Package: civetwebR (via r-universe)

October 14, 2026

Type Package

Title Embedded HTTP and WebSocket Server Through 'CivetWeb'

Version 0.1.1.1

Copyright See inst/COPYRIGHTS for the bundled CivetWeb and Mbed TLS.

SystemRequirements GNU make

Description An embedded HTTP and WebSocket server. The bundled 'CivetWeb' <<https://github.com/civetweb/civetweb>> C library does the socket work on its own threads, and R handles one request or WebSocket event at a time on its main thread, so handlers need no locking. Static files are served from disk, TLS (Transport Layer Security) is terminated by the bundled 'Mbed TLS' <<https://github.com/Mbed-TLS/mbedtls>>, and the event loop is exposed so an application can drive it from its own.

License MIT + file LICENSE

URL <https://github.com/cornball-ai/civetwebR>

BugReports <https://github.com/cornball-ai/civetwebR/issues>

Encoding UTF-8

NeedsCompilation yes

Suggests callr, tinytest

Config/pak/sysreqs make

Repository <https://cornball-ai.r-universe.dev>

Date/Publication 2026-10-14 21:09:34 UTC

RemoteUrl <https://github.com/cornball-ai/civetwebR>

RemoteRef HEAD

RemoteSha 184c860c512bdf78f4116ab752216fe829af4d0a

Contents

clear_handlers 2

group 3

handle 3

has_tls 4

next_event 4

on_ws 6

req_header 7

req_query 7

run_server 8

send_response 8

serve 9

server_port 10

start_server 11

static_files 12

stop_server 13

unhandle 14

ws_close 14

ws_send 15

Index 16

clear_handlers	<i>Clear all registered handlers</i>
----------------	--------------------------------------

Description

Removes every HTTP handler and every WebSocket event handler.

Usage

```
clear_handlers()
```

Value

Invisibly returns TRUE.

Examples

```
handle("GET", "/a", function(req) "a")
on_ws("message", function(req) ws_send(req$id, req$body))
clear_handlers()
```

group	<i>Register handlers under a common path prefix</i>
-------	---

Description

Register handlers under a common path prefix

Usage

```
group(prefix, expr)
```

Arguments

prefix	Character(1). Path prefix (e.g. "/api/v1").
expr	Code that registers handlers.

Value

Invisibly returns TRUE.

Examples

```
group("/api", {  
  handle("GET", "/users", function(req) "")  
  handle("GET", "/status", function(req) "ok")  
})  
# registered as /api/users and /api/status  
clear_handlers()
```

handle	<i>Register an HTTP handler</i>
--------	---------------------------------

Description

Registers a function for a given HTTP method and path. The path is normalized and combined with the current route prefix.

Usage

```
handle(method, path, fun, overwrite = TRUE)
```

Arguments

method	Character string. HTTP method (e.g. "GET", "POST").
path	Character string. Route path starting with "/".
fun	Function. Handler of form function(req).
overwrite	Logical. If TRUE (default), overwrite existing handler.

Value

TRUE (invisibly) on success.

Examples

```
# a string is a 200 text/plain response
handle("GET", "/hello", function(req) "hello world")
# a list sets the status, headers and body
handle("POST", "/echo", function(req) {
  list(status = 200L, headers = list("Content-Type" = "text/plain"),
    body = req$body)
})
clear_handlers()
```

has_tls	<i>Was TLS compiled in?</i>
---------	-----------------------------

Description

TRUE when the package was built with TLS support, which the bundled Mbed TLS sources provide on every platform.

Usage

```
has_tls()
```

Value

Logical.

Examples

```
has_tls()
```

next_event	<i>Wait for the next event</i>
------------	--------------------------------

Description

The low-level driver: one call returns one event from a server, or NULL after timeout_ms with nothing to do. run_server() and serve() are loops over this; an application with its own event loop calls it directly.

Usage

```
next_event(timeout_ms = 100L, server = NULL)
```

Arguments

timeout_ms	Milliseconds to wait for an event. Default 100.
server	A server handle from start_server(), or NULL for the default server.

Details

Events are lists of class `cw_request` with an event field:

- "request": an HTTP request. method, path, query, headers (named character), body (raw), body_too_large, remote_addr, remote_port. Must be answered with `send_response()`; the worker thread waits until then.
- "ws_connect": a client asks to upgrade on the WebSocket path. Same fields as a request. Must be answered with `send_response()`: TRUE accepts, FALSE refuses with 403, a list with a status refuses with that status. Nothing is sent to the client until R decides, so this is where an Origin or cookie check belongs.
- "ws_open": the handshake completed for the connection with this id.
- "ws_message": a data frame arrived. body (raw), binary, fin and opcode (1 text, 2 binary, 0 continuation). A message split across frames arrives as several events: the first with the text or binary opcode, the rest as continuations, the last with fin = TRUE; the receiver concatenates the bodies.
- "ws_close": the connection with this id is gone.

A WebSocket connection keeps the id of its connect event for its whole life; `ws_send()` and `ws_close()` address it by that id.

Control frames never reach R: CivetWeb answers a ping with a pong, and a close frame from the client is echoed back with its status code before the `ws_close` event (RFC 6455 section 5.5.1).

On Ctrl-C an interrupt condition is signalled.

Value

A `cw_request`, or NULL on timeout.

Examples

```

srv <- start_server(port = 0L, register = FALSE)
# one turn of an event loop: nothing arrives within 10 ms, so NULL
ev <- next_event(10L, srv)
is.null(ev)
stop_server(srv)

## Not run:
# the loop an application runs, until interrupted
repeat {
  ev <- next_event(100L, srv)
  if (is.null(ev)) next
  switch(ev$event,
    request    = send_response(ev$id, "ok", srv),
    ws_connect = send_response(ev$id, TRUE, srv),
  )
}
```

```

        ws_message = ws_send(ev$id, ev$body, srv))
    }

    ## End(Not run)

```

on_ws

Register WebSocket event handlers

Description

A "connect" handler runs before the handshake and decides whether the client may upgrade: it returns TRUE to accept, FALSE to refuse with 403, or a list with a status to refuse with that status. It sees the upgrade request's headers (so Origin or a cookie can be checked) and remote_addr. Without a connect handler every upgrade is accepted.

Usage

```
on_ws(event, fun)
```

Arguments

event	Character string: "connect", "open", "message", or "close".
fun	Function taking a cw_request object.

Value

Invisibly returns TRUE.

Examples

```

# accept upgrades from one origin only, and echo every message
on_ws("connect", function(req) {
    identical(req_header(req, "Origin"), "http://localhost:8080")
})
on_ws("message", function(req) ws_send(req$id, req$body))
clear_handlers()

```

req_header	<i>Helper to extract a header from a request</i>
------------	--

Description

Helper to extract a header from a request

Usage

```
req_header(req, key)
```

Arguments

req	A cw_request object.
key	Header name, matched without regard to case.

Value

The header's value as a string, or NULL when the request has no such header.

Examples

```
req <- structure(list(headers = c("Content-Type" = "application/json")),  
                  class = "cw_request")  
req_header(req, "content-type")  
req_header(req, "Accept")
```

req_query	<i>Helper to extract a query parameter from a request</i>
-----------	---

Description

Helper to extract a query parameter from a request

Usage

```
req_query(req, key)
```

Arguments

req	A cw_request object.
key	Parameter name.

Value

The parameter's value as a string, or NULL when the query has no such parameter.

Examples

```
req <- structure(list(query_params = list(page = "2")),
                 class = "cw_request")
req_query(req, "page")
req_query(req, "missing")
```

run_server

Run the CivetWeb driver loop (single-thread safe mode)

Description

This loop pulls requests from C, dispatches in R, and sends responses back. It runs until interrupted.

Usage

```
run_server(timeout_ms = 100L)
```

Arguments

timeout_ms Polling timeout in milliseconds. Default is 100ms.

Value

Invisibly returns TRUE when the loop exits.

Examples

```
## Not run:
# runs until Ctrl-C
handle("GET", "/", function(req) "hello")
start_server(port = 8080L)
run_server()
stop_server()

## End(Not run)
```

send_response

Answer a request or a WebSocket connect

Description

Answer a request or a WebSocket connect

Usage

```
send_response(id, res, server = NULL)
```

Arguments

id	The event's id.
res	For a request: a string (200, text/plain) or a list with status, headers and body (character or raw). For a WebSocket connect: TRUE to accept, FALSE to refuse with 403, or a list with a status to refuse with that status.
server	A server handle from start_server(), or NULL for the default server.

Value

NULL, invisibly.

Examples

```
srv <- start_server(port = 0L, register = FALSE)
ev <- next_event(10L, srv)
if (!is.null(ev) && ev$event == "request") {
  send_response(ev$id, list(status = 200L, body = "ok"), srv)
}
stop_server(srv)
```

 serve

Start and run the HTTP server

Description

Start and run the HTTP server

Usage

```
serve(
  port = 8080L,
  host = "127.0.0.1",
  quiet = FALSE,
  log = NULL,
  static = NULL,
  timeout_ms = 100L,
  num_threads = 50L,
  max_body_size = 8 * 1024 * 1024,
  request_timeout_ms = 30000L,
  ws_path = "/ws",
  keep_alive = FALSE,
  tls_cert = NULL
)
```

Arguments

port	Integer. Port to listen on.
host	Character(1). Bind address.
quiet	Logical. Suppress startup message.
log	Function or NULL. Optional request logger.
static	NULL or list. Static serving configuration(s).
timeout_ms	Polling timeout in milliseconds. Default is 100ms.
num_threads	Integer. Number of worker threads. Default is 50.
max_body_size	Number. Maximum request body size in bytes. Default is 8MiB.
request_timeout_ms	Integer. Socket receive timeout. Default is 30s.
ws_path	String. The path WebSocket upgrades are accepted on. Default <code>"/ws"</code> .
keep_alive	Logical. Keep HTTP connections open between requests. Default <code>FALSE</code> .
tls_cert	NULL, or the path of a PEM file holding the server's certificate and private key; see start_server .

Value

Invisibly returns `TRUE` when the loop exits.

Examples

```
## Not run:
# runs until Ctrl-C
handle("GET", "/", function(req) "hello")
serve(port = 8080L)

## End(Not run)
```

server_port	<i>The port a server is listening on</i>
-------------	--

Description

Useful after `start_server(port = 0)`.

Usage

```
server_port(server = NULL)
```

Arguments

server A server handle from `start_server()`, or `NULL` for the default server.

Value

Integer port.

Examples

```
srv <- start_server(port = 0L, register = FALSE)
server_port(srv)
stop_server(srv)
```

start_server	<i>Start the CivetWeb server</i>
--------------	----------------------------------

Description

Starts a server and, by default, registers it as the package's default server, the one `run_server()`, `serve()`, `ws_send()` and friends use when no server is given. With `register = FALSE` the server is only returned, so one R process can run several.

Usage

```
start_server(
  port = 8080L,
  host = "127.0.0.1",
  num_threads = 50L,
  max_body_size = 8 * 1024 * 1024,
  request_timeout_ms = 30000L,
  ws_path = "/ws",
  static_dirs = NULL,
  keep_alive = FALSE,
  tls_cert = NULL,
  register = TRUE
)
```

Arguments

port	Integer. Port to listen on; 0 lets the OS choose (see <code>server_port()</code>).
host	String. Host to expose to.
num_threads	Integer. Number of worker threads. Default is 50.
max_body_size	Number. Maximum request body size in bytes. Default is 8MiB.
request_timeout_ms	Integer. Socket receive timeout. Default is 30s.
ws_path	String. The path WebSocket upgrades are accepted on. Default <code>"/ws"</code> .
static_dirs	NULL or a named character vector mapping URL prefixes to directories, e.g. <code>c("/assets/" = "www")</code> . Requests under these prefixes are served by CivetWeb itself, with byte ranges, and never reach R. Distinct from the R-side <code>static</code> argument of <code>serve()</code> .

keep_alive	Logical. Keep HTTP connections open between requests. Default FALSE.
tls_cert	NULL, or the path of a PEM file holding the server's certificate (followed by its chain, if any) and its private key. When given, the port speaks TLS (https and wss) through the bundled Mbed TLS, and a plain-HTTP client on it is refused.
register	Logical. Make this the default server. Default TRUE.

Value

A server handle (class `cw_server`), invisibly.

Examples

```
# port 0: the OS picks a free port; register = FALSE: a handle only
srv <- start_server(port = 0L, register = FALSE)
srv
server_port(srv)
stop_server(srv)

# a directory served by CivetWeb itself, never reaching R
www <- tempfile("www")
dir.create(www)
writeLines("hi", file.path(www, "index.html"))
srv <- start_server(port = 0L, register = FALSE,
  static_dirs = c("/assets/" = www))
stop_server(srv)
unlink(www, recursive = TRUE)
```

static_files	<i>Static file handler (serves files from a directory)</i>
--------------	--

Description

Static file handler (serves files from a directory)

Usage

```
static_files(
  dir,
  prefix = "/",
  index = c("index.html"),
  cache_control = NULL,
  list_dirs = FALSE,
  template = NULL
)
```

Arguments

dir	Character(1). Root directory to serve from.
prefix	Character(1). URL prefix.
index	Character vector. Index files.
cache_control	NULL or character(1).
list_dirs	Logical. Enable directory listing.
template	Character(1) or NULL. Optional template path override.

Value

A handler: a function of a request that returns a response list (status, headers, raw body), as `serve(static = )` installs it.

Examples

```
www <- tempfile("www")
dir.create(www)
writeLines("<h1>hi</h1>", file.path(www, "index.html"))
h <- static_files(www, prefix = "/")
res <- h(list(path = "/index.html"))
res$status
rawToChar(res$body)
h(list(path = "/missing.html"))$status
unlink(www, recursive = TRUE)
```

stop_server

Stop a CivetWeb server

Description

Answers every request still waiting on R with 503, closes the connections and releases the server. Stopping the default server clears it.

Usage

```
stop_server(server = NULL)
```

Arguments

server	A server handle from <code>start_server()</code> , or NULL for the default server.
--------	--

Value

Invisibly returns TRUE.

Examples

```

srv <- start_server(port = 0L, register = FALSE)
stop_server(srv)

# the default server
start_server(port = 0L)
stop_server()

```

unhandle

Unregister an HTTP handler

Description

Removes a function for a given HTTP method and path.

Usage

```
unhandle(method, path)
```

Arguments

method	Character string. HTTP method (e.g. "GET").
path	Character string. Route path.

Value

TRUE (invisibly) on success.

Examples

```

handle("GET", "/temporary", function(req) "here for now")
unhandle("GET", "/temporary")

```

ws_close

Close a WebSocket connection from the server side

Description

Sends a close frame with code; the client answers and the connection ends, which arrives later as a ws_close event. Further ws_send() calls on the id return FALSE.

Usage

```
ws_close(id, code = 1000L, server = NULL)
```

Arguments

id	Connection ID.
code	Integer close code. Default 1000 (normal closure).
server	A server handle from start_server(), or NULL for the default server.

Value

TRUE if the close frame was written.

Examples

```
srv <- start_server(port = 0L, register = FALSE)
# no connection has this id, so nothing is written
ws_close(1L, 1000L, srv)
stop_server(srv)
```

ws_send	<i>Send a WebSocket message</i>
---------	---------------------------------

Description

Send a WebSocket message

Usage

```
ws_send(id, data, server = NULL)
```

Arguments

id	Connection ID.
data	Character string or raw vector.
server	A server handle from start_server(), or NULL for the default server.

Value

TRUE if the frame was written, FALSE if the connection is not open.

Examples

```
srv <- start_server(port = 0L, register = FALSE)
# no connection has this id, so nothing is written
ws_send(1L, "hello", srv)
stop_server(srv)
```

Index

`clear_handlers`, [2](#)
`group`, [3](#)
`handle`, [3](#)
`has_tls`, [4](#)
`next_event`, [4](#)
`on_ws`, [6](#)
`req_header`, [7](#)
`req_query`, [7](#)
`run_server`, [8](#)
`send_response`, [8](#)
`serve`, [9](#)
`server_port`, [10](#)
`start_server`, [10](#), [11](#)
`static_files`, [12](#)
`stop_server`, [13](#)
`unhandle`, [14](#)
`ws_close`, [14](#)
`ws_send`, [15](#)