Package: httpuv (via r-universe)

October 20, 2026

Type Package

Title HTTP and WebSocket Server Library

Version 1.6.16.9001

Description Provides low-level socket and protocol support for handling HTTP and WebSocket requests directly from within R. It is primarily intended as a building block for other packages, rather than making it particularly easy to create complete web applications using httpuv alone. httpuv is built on top of the libuv and http-parser C libraries, both of which were developed by Joyent, Inc. (See LICENSE file for libuv and http-parser license information.)

License GPL (>= 2) | file LICENSE

URL https://github.com/abiomix/abiomix-r-open, https://rstudio.github.io/httpuv/, https://github.com/rstudio/httpuv

BugReports https://github.com/abiomix/abiomix-r-open/issues

Depends R (>= 2.15.1)

Imports later (>= 0.8.0), promises, R6, Rcpp (>= 1.0.7), utils

Suggests callr, curl, jsonlite, testthat (>= 3.0.0), websocket

LinkingTo later, Rcpp

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-07-01

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements GNU make, zlib

Additional_repositories https://abiomix.r-universe.dev

Config/abiomix/public-license GPL-2.0-or-later

Config/abiomix/commercial-license Separate written agreement for Abiomix-owned contributions

Config/abiomix/license-policy https://github.com/abiomix/abiomix-r-open/blob/main/LICENSES.md

Collate 'RcppExports.R' 'httpuv-package.R' 'httpuv.R' 'random_port.R' 'server.R' 'staticServer.R' 'static_paths.R' 'utils.R'

Config/pak/sysreqs make zlib1g-dev

Repository https://abiomix.r-universe.dev

Date/Publication 2026-07-22 06:07:58 UTC

RemoteUrl https://github.com/abiomix/abiomix-r-open

RemoteRef HEAD

RemoteSha 025cfad5454b13ce6c9b7323147fddfcb2df8221

RemoteSubdir packages/httpuv

Contents

httpuv-package . . . 2
encodeURI . . . 3
interrupt . . . 4
ipFamily . . . 4
listServers . . . 5
randomPort . . . 5
rawToBase64 . . . 6
runServer . . . 6
runStaticServer . . . 7
service . . . 9
startDaemonizedServer . . . 10
startServer . . . 10
staticPath . . . 13
staticPathOptions . . . 15
stopAllServers . . . 16
stopDaemonizedServer . . . 16
stopServer . . . 16
WebSocket . . . 17

Index **20**

`httpuv-package` *HTTP and WebSocket server*

Description

HTTP and WebSocket server

Details

Allows R code to listen for and interact with HTTP and WebSocket clients, so you can serve web traffic directly out of your R process. Implementation is based on libuv and http-parser.

This is a low-level library that provides little more than network I/O and implementations of the HTTP and WebSocket protocols. For an easy way to create web applications, try Shiny instead.

Author(s)

Joe Cheng <joe@rstudio.com>

See Also

startServer

Examples

```
## Not run:
demo("echo", package="httpuv")

## End(Not run)
```

---

encodeURI *URI encoding/decoding*

Description

Encodes/decodes strings using URI encoding/decoding in the same way that web browsers do. The precise behaviors of these functions can be found at developer.mozilla.org: encodeURI, encodeURIComponent, decodeURI, decodeURIComponent

Usage

```
encodeURI(value)

encodeURIComponent(value)

decodeURI(value)

decodeURIComponent(value)
```

Arguments

value	Character vector to be encoded or decoded.

Details

Intended as a faster replacement for `utils::URLencode()` and `utils::URLdecode()`.

encodeURI differs from encodeURIComponent in that the former will not encode reserved characters: `;,/?:@&=+$`

decodeURI differs from decodeURIComponent in that it will refuse to decode encoded sequences that decode to a reserved character. (If in doubt, use decodeURIComponent.)

For `encodeURI` and `encodeURIComponent`, input strings will be converted to UTF-8 before URL-encoding.

Value

Encoded or decoded character vector of the same length as the input value. `decodeURI` and `decodeURIComponent` will return strings that are UTF-8 encoded.

---

interrupt *Interrupt httpuv runloop*

Description

Interrupts the currently running httpuv runloop, meaning `runServer()` or `service()` will return control back to the caller and no further tasks will be processed until those methods are called again. Note that this may cause in-process uploads or downloads to be interrupted in mid-request.

Usage

```
interrupt()
```

---

ipFamily *Check whether an address is IPv4 or IPv6*

Description

Given an IP address, this checks whether it is an IPv4 or IPv6 address.

Usage

```
ipFamily(ip)
```

Arguments

`ip`	A single string representing an IP address.

Value

For IPv4 addresses, `4`; for IPv6 addresses, `6`. If the address is neither, `-1`.

Examples

```
ipFamily("127.0.0.1")   # 4
ipFamily("500.0.0.500") # -1
ipFamily("500.0.0.500") # -1

ipFamily("::")          # 6
ipFamily("::1")         # 6
ipFamily("fe80::1ff:fe23:4567:890a") # 6
```

---

listServers *List all running httpuv servers*

Description

This returns a list of all running httpuv server applications.

Usage

```
listServers()
```

---

randomPort *Find an open TCP port*

Description

Finds a random available TCP port for listening on, within a specified range of ports. The default range of ports to check is 1024 to 49151, which is the set of TCP User Ports. This function automatically excludes some ports which are considered unsafe by web browsers.

Usage

```
randomPort(min = 1024L, max = 49151L, host = "127.0.0.1", n = 20)
```

Arguments

`min`	Minimum port number.
`max`	Maximum port number.
`host`	A string that is a valid IPv4 or IPv6 address that is owned by this server, which the application will listen on. `"0.0.0.0"` represents all IPv4 addresses and `"::/0"` represents all IPv6 addresses.
`n`	Number of ports to try before giving up.

Value

A port that is available to listen on.

Examples

```
## Not run:
s <- startServer("127.0.0.1", randomPort(), list())
browseURL(paste0("http://127.0.0.1:", s$getPort()))

s$stop()

## End(Not run)
```

`rawToBase64` *Convert raw vector to Base64-encoded string*

Description

Converts a raw vector to its Base64 encoding as a single-element character vector.

Usage

```
rawToBase64(x)
```

Arguments

x	A raw vector.

Examples

```
set.seed(100)
result <- rawToBase64(as.raw(runif(19, min=0, max=256)))
stopifnot(identical(result, "TkGNDnd7z16LK5/hR2bDqzRbXA=="))
```

`runServer` *Run a server*

Description

This is a convenience function that provides a simple way to call `startServer()`, `service()`, and `stopServer()` in the correct sequence. It does not return unless interrupted or an error occurs.

Usage

```
runServer(host, port, app, interruptIntervalMs = NULL)
```

Arguments

host	A string that is a valid IPv4 or IPv6 address that is owned by this server, which the application will listen on. "0.0.0.0" represents all IPv4 addresses and "::/0" represents all IPv6 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.
app	A collection of functions that define your application. See startServer().
interruptIntervalMs	Deprecated (last used in httpuv 1.3.5).

Details

If you have multiple hosts and/or ports to listen on, call the individual functions instead of runServer.

See Also

startServer(), service(), stopServer()

Examples

```
## Not run:
# A very basic application
runServer("0.0.0.0", 5000,
  list(
    call = function(req) {
      list(
        status = 200L,
        headers = list(
          'Content-Type' = 'text/html'
        ),
        body = "Hello world!"
      )
    }
  )
)

## End(Not run)
```

---

runStaticServer *Serve a directory*

Description

runStaticServer() provides a convenient interface to start a server to host a single static directory, either in the foreground or the background.

Usage

```
runStaticServer(
  dir = getwd(),
  host = "127.0.0.1",
  port = NULL,
  ...,
  background = FALSE,
  browse = interactive()
)
```

Arguments

dir	The directory to serve. Defaults to the current working directory.
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.
...	Arguments passed on to `staticPath` `path` The local path. `indexhtml` If an index.html file is present, should it be served up when the client requests the static path or any subdirectory? `fallthrough` With the default value, `FALSE`, if a request is made for a file that doesn't exist, then httpuv will immediately send a 404 response from the background I/O thread, without needing to call back into the main R thread. This offers the best performance. If the value is `TRUE`, then instead of sending a 404 response, httpuv will call the application's `call` function, and allow it to handle the request. `html_charset` When HTML files are served, the value that will be provided for `charset` in the Content-Type header. For example, with the default value, `"utf-8"`, the header is `Content-Type: text/html; charset=utf-8`. If `""` is used, then no `charset` will be added in the Content-Type header. `headers` Additional headers and values that will be included in the response. `validation` An optional validation pattern. Presently, the only type of validation supported is an exact string match of a header. For example, if `validation` is `'"abc" = "xyz"'`, then HTTP requests must have a header named `abc` (case-insensitive) with the value `xyz` (case-sensitive). If a request does not have a matching header, than httpuv will give a 403 Forbidden response. If the `character(0)` (the default), then no validation check will be performed.
background	Whether to run the server in the background. By default, the server runs in the foreground and blocks the R console. You can stop the server by interrupting it with `Ctrl + C`. When `background = TRUE`, the server will run in the background and will process requests when the R console is idle. To stop a background server, call `stopAllServers()` or call `stopServer()` on the server object returned (invisibly) by this function.

browse — Whether to automatically open the served directory in a web browser. Defaults to TRUE when running interactively.

Value

Starts a server on the specified host and port. By default the server runs in the foreground and is accessible at http://127.0.0.1:7446. When background = TRUE, the server object is returned invisibly.

See Also

runServer() provides a similar interface for running a dynamic app server. Both runStaticServer() and runServer() are built on top of startServer(), service() and stopServer(). Learn more about httpuv servers in startServer().

Examples

```
website_dir <- system.file("example-static-site", package = "httpuv")
runStaticServer(dir = website_dir)
```

---

service *Process requests*

Description

Process HTTP requests and WebSocket messages. If there is nothing on R's call stack – if R is sitting idle at the command prompt – it is not necessary to call this function, because requests will be handled automatically. However, if R is executing code, then requests will not be handled until either the call stack is empty, or this function is called (or alternatively, later::run_now() is called).

Usage

```
service(timeoutMs = ifelse(interactive(), 100, 1000))
```

Arguments

timeoutMs — Approximate number of milliseconds to run before returning. It will return this duration has elapsed. If 0 or Inf, then the function will continually process requests without returning unless an error occurs. If NA, performs a non-blocking run without waiting.

Details

In previous versions of httpuv (1.3.5 and below), even if a server created by startServer() exists, no requests were serviced unless and until service was called.

This function simply calls later::run_now(), so if your application schedules any later::later() callbacks, they will be invoked.

Examples

```
## Not run:
while (TRUE) {
  service()
}

## End(Not run)
```

`startDaemonizedServer` *Create an HTTP/WebSocket daemonized server (deprecated)*

Description

This function will be removed in a future release of httpuv. It is simply a wrapper for `startServer()`. In previous versions of httpuv (1.3.5 and below), `startServer` ran applications in the foreground and `startDaemonizedServer` ran applications in the background, but now both of them run applications in the background.

Usage

```
startDaemonizedServer(host, port, app, quiet = FALSE)
```

Arguments

`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.
`app`	A collection of functions that define your application. See Details.
`quiet`	If `TRUE`, suppress error messages from starting app.

`startServer` *Create an HTTP/WebSocket server*

Description

Creates an HTTP/WebSocket server on the specified host and port.

Usage

```
startServer(host, port, app, quiet = FALSE)

startPipeServer(name, mask, app, quiet = FALSE)
```

Arguments

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.
app	A collection of functions that define your application. See Details.
quiet	If TRUE, suppress error messages from starting app.
name	A string that indicates the path for the domain socket (on Unix-like systems) or the name of the named pipe (on Windows).
mask	If non-NULL and non-negative, this numeric value is used to temporarily modify the process's umask while the domain socket is being created. To ensure that only root can access the domain socket, use strtoi("777", 8); or to allow owner and group read/write access, use strtoi("117", 8). If the value is NULL then the process's umask is left unchanged. (This parameter has no effect on Windows.)

Details

startServer binds the specified port and listens for connections on an thread running in the background. This background thread handles the I/O, and when it receives a HTTP request, it will schedule a call to the user-defined R functions in app to handle the request. This scheduling is done with later::later(). When the R call stack is empty – in other words, when an interactive R session is sitting idle at the command prompt – R will automatically run the scheduled calls. However, if the call stack is not empty – if R is evaluating other R code – then the callbacks will not execute until either the call stack is empty, or the later::run_now() function is called. This function tells R to execute any callbacks that have been scheduled by later::later(). The service() function is essentially a wrapper for later::run_now().

In older versions of httpuv (1.3.5 and below), it did not use a background thread for I/O, and when this function was called, it did not accept connections immediately. It was necessary to call service() repeatedly in order to actually accept and handle connections.

If the port cannot be bound (most likely due to permissions or because it is already bound), an error is raised.

The application can also specify paths on the filesystem which will be served from the background thread, without invoking $call() or $onHeaders(). Files served this way will be only use a C++ code, which is faster than going through R, and will not be blocked when R code is executing. This can greatly improve performance when serving static assets.

The app parameter is where your application logic will be provided to the server. This can be a list, environment, or reference class that contains the following methods and fields:

call(req) Process the given HTTP request, and return an HTTP response (see Response Values). This method should be implemented in accordance with the Rook specification. Note that httpuv augments req with an additional item, req$HEADERS, which is a named character vector of request headers.

`onHeaders(req)` Optional. Similar to `call`, but occurs when headers are received. 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.)

`onWSOpen(ws)` Called back when a WebSocket connection is established. The given object can be used to be notified when a message is received from the client, to send messages to the client, etc. See `WebSocket()`.

`staticPaths` A named list of paths that will be served without invoking `call()` 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 `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 `staticPathOptions()` with no arguments.

The `startPipeServer` variant can be used instead of `startServer` to listen on a Unix domain socket or named pipe rather than a TCP socket (this is not common).

Value

A handle for this server that can be passed to `stopServer()` to shut the server down.

A `WebServer()` or `PipeServer()` object.

Response Values

The `call` function is expected to return a list containing the following, which are converted to an HTTP response and sent to the client:

`status` A numeric HTTP status code, e.g. `200` or `404L`.

`headers` A named list of HTTP headers and their values, as strings. This can also be missing, an empty list, or `NULL`, in which case no headers (other than the `Date` and `Content-Length` headers, as required) will be added.

`body` A string (or `raw` vector) to be sent as the body of the HTTP response. This can also be omitted or set to `NULL` to avoid sending any body, which is useful for HTTP `1xx`, `204`, and `304` responses, as well as responses to `HEAD` requests.

See Also

`stopServer()`, `runServer()`, `listServers()`, `stopAllServers()`.

Examples

```
## Not run:
# A very basic application
s <- startServer("0.0.0.0", 5000,
  list(
    call = function(req) {
      list(
        status = 200L,
        headers = list(
          'Content-Type' = 'text/html'
```

```
        ),
        body = "Hello world!"
      )
    }
  )
)

s$stop()


# An application that serves static assets at the URL paths /assets and /lib
s <- startServer("0.0.0.0", 5000,
  list(
    call = function(req) {
      list(
        status = 200L,
        headers = list(
          'Content-Type' = 'text/html'
        ),
        body = "Hello world!"
      )
    },
    staticPaths = list(
      "/assets" = "content/assets/",
      "/lib" = staticPath(
        "content/lib",
        indexhtml = FALSE
      ),
      # This subdirectory of /lib should always be handled by the R code path
      "/lib/dynamic" = excludeStaticPath()
    ),
    staticPathOptions = staticPathOptions(
      indexhtml = TRUE
    )
  )
)

s$stop()

## End(Not run)
```

`staticPath` *Create a staticPath object*

Description

The `staticPath` function creates a `staticPath` object. Note that if any of the arguments (other than `path`) are `NULL`, then that means that for this particular static path, it should inherit the behavior from the staticPathOptions set for the application as a whole.

Usage

```
staticPath(
  path,
  indexhtml = NULL,
  fallthrough = NULL,
  html_charset = NULL,
  headers = NULL,
  validation = NULL
)

excludeStaticPath()
```

Arguments

path	The local path.
indexhtml	If an index.html file is present, should it be served up when the client requests the static path or any subdirectory?
fallthrough	With the default value, FALSE, if a request is made for a file that doesn't exist, then httpuv will immediately send a 404 response from the background I/O thread, without needing to call back into the main R thread. This offers the best performance. If the value is TRUE, then instead of sending a 404 response, httpuv will call the application's call function, and allow it to handle the request.
html_charset	When HTML files are served, the value that will be provided for charset in the Content-Type header. For example, with the default value, "utf-8", the header is Content-Type: text/html; charset=utf-8. If "" is used, then no charset will be added in the Content-Type header.
headers	Additional headers and values that will be included in the response.
validation	An optional validation pattern. Presently, the only type of validation supported is an exact string match of a header. For example, if validation is '"abc" = "xyz"', then HTTP requests must have a header named abc (case-insensitive) with the value xyz (case-sensitive). If a request does not have a matching header, than httpuv will give a 403 Forbidden response. If the character(0) (the default), then no validation check will be performed.

Details

The excludeStaticPath function tells the application to ignore a particular path for static serving. This is useful when you want to include a path for static serving (like "/") but then exclude a subdirectory of it (like "/dynamic") so that the subdirectory will always be passed to the R code for handling requests. excludeStaticPath can be used not only for directories; it can also exclude specific files.

See Also

staticPathOptions().

staticPathOptions *Create options for static paths*

Description

Create options for static paths

Usage

```
staticPathOptions(
  indexhtml = TRUE,
  fallthrough = FALSE,
  html_charset = "utf-8",
  headers = list(),
  validation = character(0),
  exclude = FALSE
)
```

Arguments

indexhtml	If an index.html file is present, should it be served up when the client requests the static path or any subdirectory?
fallthrough	With the default value, FALSE, if a request is made for a file that doesn't exist, then httpuv will immediately send a 404 response from the background I/O thread, without needing to call back into the main R thread. This offers the best performance. If the value is TRUE, then instead of sending a 404 response, httpuv will call the application's call function, and allow it to handle the request.
html_charset	When HTML files are served, the value that will be provided for charset in the Content-Type header. For example, with the default value, "utf-8", the header is Content-Type: text/html; charset=utf-8. If "" is used, then no charset will be added in the Content-Type header.
headers	Additional headers and values that will be included in the response.
validation	An optional validation pattern. Presently, the only type of validation supported is an exact string match of a header. For example, if validation is '"abc" = "xyz"', then HTTP requests must have a header named abc (case-insensitive) with the value xyz (case-sensitive). If a request does not have a matching header, than httpuv will give a 403 Forbidden response. If the character(0) (the default), then no validation check will be performed.
exclude	Should this path be excluded from static serving? (This is only to be used internally, for excludeStaticPath().)

stopAllServers *Stop all servers*

Description

This will stop all applications which were created by `startServer()` or `startPipeServer()`.

Usage

```
stopAllServers()
```

See Also

`stopServer()` to stop a specific server.

stopDaemonizedServer *Stop a running daemonized server in Unix environments (deprecated)*

Description

This function will be removed in a future release of httpuv. Instead, use `stopServer()`.

Usage

```
stopDaemonizedServer(server)
```

Arguments

`server`	A server object that was previously returned from `startServer()` or `startPipeServer()`.

stopServer *Stop a server*

Description

Given a server object that was returned from a previous invocation of `startServer()` or `startPipeServer()`, this closes all open connections for that server and unbinds the port.

Usage

```
stopServer(server)
```

Arguments

`server`	A server object that was previously returned from `startServer()` or `startPipeServer()`.

See Also

`stopAllServers()` to stop all servers.

`WebSocket` *WebSocket class*

Description

A `WebSocket` object represents a single WebSocket connection. The object can be used to send messages and close the connection, and to receive notifications when messages are received or the connection is closed.

Details

Note that this WebSocket class is different from the one provided by the package named websocket. This class is meant to be used on the server side, whereas the one in the websocket package is to be used as a client. The WebSocket class in httpuv has an older API than the one in the websocket package.

WebSocket objects should never be created directly. They are obtained by passing an `onWSOpen` function to `startServer()`.

Public fields

`handle` The server handle

`messageCallbacks` A list of callback functions that will be invoked when a message is received on this connection.

`closeCallbacks` A list of callback functions that will be invoked when the connection is closed.

`request` The Rook request environment that opened the connection. This can be used to inspect HTTP headers, for example.

Methods

Public methods:

- `WebSocket$new()`
- `WebSocket$onMessage()`
- `WebSocket$onClose()`
- `WebSocket$send()`
- `WebSocket$close()`
- `WebSocket$clone()`

Method `new()`**:** Initializes a new WebSocket object.

Usage:

```
WebSocket$new(handle, req)
```

Arguments:

`handle` An C++ WebSocket handle.

`req` The Rook request environment that opened the connection.

Method `onMessage()`**:** Registers a callback function that will be invoked whenever a message is received on this connection.

Usage:

```
WebSocket$onMessage(func)
```

Arguments:

`func` The callback function to be registered. The callback function will be invoked with two arguments. The first argument is `TRUE` if the message is binary and `FALSE` if it is text. The second argument is either a raw vector (if the message is binary) or a character vector.

Method `onClose()`**:** Registers a callback function that will be invoked when the connection is closed.

Usage:

```
WebSocket$onClose(func)
```

Arguments:

`func` The callback function to be registered.

Method `send()`**:** Begins sending the given message over the websocket.

Usage:

```
WebSocket$send(message)
```

Arguments:

`message` Either a raw vector, or a single-element character vector that is encoded in UTF-8.

Method `close()`**:** Closes the websocket connection

Usage:

```
WebSocket$close(code = 1000L, reason = "")
```

Arguments:

`code` An integer that indicates the WebSocket close code.

`reason` A concise human-readable prose explanation for the closure.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
WebSocket$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# A WebSocket echo server that listens on port 8080
startServer("0.0.0.0", 8080,
  list(
    onHeaders = function(req) {
      # Print connection headers
      cat(capture.output(str(as.list(req))), sep = "\n")
    },
    onWSOpen = function(ws) {
      cat("Connection opened.\n")

      ws$onMessage(function(binary, message) {
        cat("Server received message:", message, "\n")
        ws$send(message)
      })
      ws$onClose(function() {
        cat("Connection closed.\n")
      })

    }
  )
)

## End(Not run)
```

Index

∗ **package**
    httpuv-package, 2

decodeURI *(*encodeURI*)*, 3
decodeURIComponent *(*encodeURI*)*, 3

encodeURI, 3
encodeURIComponent *(*encodeURI*)*, 3
excludeStaticPath *(*staticPath*)*, 13
excludeStaticPath(), *15*

httpuv *(*httpuv-package*)*, 2
httpuv-package, 2

interrupt, 4
ipFamily, 4

later::later(), *9*, *11*
later::run_now(), *9*, *11*
listServers, 5
listServers(), *12*

PipeServer(), *12*

randomPort, 5
rawToBase64, 6
runServer, 6
runServer(), *4*, *9*, *12*
runStaticServer, 7

service, 9
service(), *4*, *6*, *7*, *9*, *11*
startDaemonizedServer, 10
startPipeServer *(*startServer*)*, 10
startPipeServer(), *16*
startServer, *3*, 10
startServer(), *6*, *7*, *9*, *10*, *16*, *17*
staticPath, *8*, 13
staticPath(), *12*
staticPathOptions, 15
staticPathOptions(), *12*, *14*

stopAllServers, 16
stopAllServers(), *8*, *12*, *17*
stopDaemonizedServer, 16
stopServer, 16
stopServer(), *6–9*, *12*, *16*

utils::URLdecode(), *4*
utils::URLencode(), *4*

WebServer(), *12*
WebSocket, 17
WebSocket(), *12*