

# Package: ambiorix (via r-universe)

September 18, 2024

**Title** Web Framework Inspired by 'Express.js'

**Version** 2.1.1

**Description** A web framework inspired by 'express.js' to build any web service from multi-page websites to 'RESTful' application programming interfaces.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** false

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.1

**Depends** R (>= 4.1.0)

**Imports** fs, log, cli, glue, httpuv, methods, jsonlite, assertthat

**Suggests** mime, readr, ggplot2, promises, websocket, htmltools, commonmark, htmlwidgets, testthat (>= 3.0.0)

**URL** <https://github.com/devOpifex/ambiorix>, <https://ambiorix.dev>

**BugReports** <https://github.com/devOpifex/ambiorix/issues>

**Config/testthat/edition** 3

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

**RemoteUrl** <https://github.com/devOpifex/ambiorix>

**RemoteRef** HEAD

**RemoteSha** e4fdfd2e8aece949187b04f95deaac1af4d5930e

## Contents

Ambiorix	2
as_cookie_parser	9
as_cookie_preprocessor	10
as_path_to_pattern	10
as_renderer	11

content . . . . .	11
create_dockerfile . . . . .	12
default_cookie_parser . . . . .	12
forward . . . . .	13
import . . . . .	13
is_renderer_obj . . . . .	14
jobj . . . . .	14
mockRequest . . . . .	15
new_log . . . . .	15
parsers . . . . .	16
pre_hook . . . . .	16
Request . . . . .	17
Response . . . . .	20
responses . . . . .	28
robject . . . . .	29
Router . . . . .	30
Routing . . . . .	31
serialise . . . . .	36
set_log . . . . .	37
set_params . . . . .	37
stop_all . . . . .	38
token_create . . . . .	38
use_html_template . . . . .	38
Websocket . . . . .	39
websocket_client . . . . .	40
<b>Index</b>	<b>41</b>

---

 Ambiorix

*Ambiorix*


---

**Description**

Web server.

**Value**

An object of class `Ambiorix` from which one can add routes, routers, and run the application.

**Super class**

`ambiorix::Routing` -> `Ambiorix`

**Public fields**

not\_found 404 Response, must be a handler function that accepts the request and the response, by default uses `response_404()`.

error 500 response when the route errors, must a handler function that accepts the request and the response, by default uses `response_500()`.

on\_stop Callback function to run when the app stops, takes no argument.

**Active bindings**

port Port to run the application.

host Host to run the application.

limit Max body size, defaults to  $5 * 1024 * 1024$ .

**Methods****Public methods:**

- `Ambiorix$new()`
- `Ambiorix$cache_templates()`
- `Ambiorix$listen()`
- `Ambiorix$set_404()`
- `Ambiorix$set_error()`
- `Ambiorix$static()`
- `Ambiorix$start()`
- `Ambiorix$serialiser()`
- `Ambiorix$stop()`
- `Ambiorix$print()`
- `Ambiorix$clone()`

**Method new():**

*Usage:*

```
Ambiorix$new(  
  host = getOption("ambiorix.host", "0.0.0.0"),  
  port = getOption("ambiorix.port", NULL),  
  log = getOption("ambiorix.logger", TRUE)  
)
```

*Arguments:*

host A string defining the host.

port Integer defining the port, defaults to `ambiorix.port` option: uses a random port if NULL.

log Whether to generate a log of events.

*Details:* Define the webserver.

**Method cache\_templates():**

*Usage:*

```
Ambiorix$cache_templates()
```

*Details:* Cache templates in memory instead of reading them from disk.

**Method** listen():

*Usage:*

```
Ambiorix$listen(port)
```

*Arguments:*

port Port number.

*Details:* Specifies the port to listen on.

*Examples:*

```
app <- Ambiorix$new()

app$listen(3000L)

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start()
```

**Method** set\_404():

*Usage:*

```
Ambiorix$set_404(handler)
```

*Arguments:*

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

*Details:* Sets the 404 page.

*Examples:*

```
app <- Ambiorix$new()

app$set_404(function(req, res){
  res$send("Nothing found here")
})

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start()
```

**Method** set\_error():

*Usage:*

```
Ambiorix$set_error(handler)
```

*Arguments:*

handler Function that accepts a request, response and an error object.

*Details:* Sets the error handler.

*Examples:*

```
# my custom error handler:
error_handler <- \(req, res, error) {
  if (!is.null(error)) {
    error_msg <- conditionMessage(error)
    cli::cli_alert_danger("Error: {error_msg}")
  }
  response <- list(
    code = 500L,
    msg = "Uhhmmm... Looks like there's an error from our side :("
  )
  res$
    set_status(500L)$
    json(response)
}

# handler for GET at /whoami:
whoami <- \(req, res) {
  # simulate error (object 'Pikachu' is not defined)
  print(Pikachu)
}

app <- Ambiorix$
  new()$
  set_error(error_handler)$
  get("/whoami", whoami)

if (interactive()) {
  app$start(open = FALSE)
}
```

**Method** static():

*Usage:*

```
Ambiorix$static(path, uri = "www")
```

*Arguments:*

path Local path to directory of assets.

uri URL path where the directory will be available.

*Details:* Static directories

**Method** start():

*Usage:*

```
Ambiorix$start(port = NULL, host = NULL, open = interactive())
```

*Arguments:*

port Integer defining the port, defaults to `ambiorix.port` option: uses a random port if NULL.

host A string defining the host.

open Whether to open the app the browser.

*Details:* Start Start the webserver.

*Examples:*

```
app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start(port = 3000L)
```

**Method** `serialiser()`:

*Usage:*

```
Ambiorix$serialiser(handler)
```

*Arguments:*

handler Function to use to serialise. This function should accept two arguments: the object to serialise and ...

*Details:* Define Serialiser

*Examples:*

```
app <- Ambiorix$new()

app$serialiser(function(data, ...){
  jsonlite::toJSON(x, ..., pretty = TRUE)
})

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start()
```

**Method** `stop()`:

*Usage:*

```
Ambiorix$stop()
```

*Details:* Stop Stop the webserver.

**Method** `print()`:

*Usage:*

```
Ambiorix$print()
```

*Details:* Print

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

app$on_stop <- function(){
  cat("Bye!\n")
}

if(interactive())
  app$start()

## -----
## Method `Ambiorix$listen`
## -----

app <- Ambiorix$new()

app$listen(3000L)

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start()

## -----
## Method `Ambiorix$set_404`
## -----

app <- Ambiorix$new()

app$set_404(function(req, res){
  res$send("Nothing found here")
})

app$get("/", function(req, res){
```

```

    res$send("Using {ambiorix}!")
  })

  if(interactive())
    app$start()

  ## -----
  ## Method `Ambiorix$set_error`
  ## -----

  # my custom error handler:
  error_handler <- \(req, res, error) {
    if (!is.null(error)) {
      error_msg <- conditionMessage(error)
      cli::cli_alert_danger("Error: {error_msg}")
    }
    response <- list(
      code = 500L,
      msg = "Uhhmm... Looks like there's an error from our side :("
    )
    res$
      set_status(500L)$
      json(response)
  }

  # handler for GET at /whoami:
  whoami <- \(req, res) {
    # simulate error (object 'Pikachu' is not defined)
    print(Pikachu)
  }

  app <- Ambiorix$
    new()$
    set_error(error_handler)$
    get("/whoami", whoami)

  if (interactive()) {
    app$start(open = FALSE)
  }

  ## -----
  ## Method `Ambiorix$start`
  ## -----

  app <- Ambiorix$new()

  app$get("/", function(req, res){
    res$send("Using {ambiorix}!")
  })

  if(interactive())
    app$start(port = 3000L)

```



```
## -----  
## Method `Ambiorix$serialiser`  
## -----  
  
app <- Ambiorix$new()  
  
app$serialiser(function(data, ...){  
  jsonlite::toJSON(x, ..., pretty = TRUE)  
})  
  
app$get("/", function(req, res){  
  res$send("Using {ambiorix}!")  
})  
  
if(interactive())  
  app$start()
```

---

as\_cookie\_parser

*Define a Cookie Parser*

---

## Description

Identifies a function as a cookie parser (see example).

## Usage

```
as_cookie_parser(fn)
```

## Arguments

**fn** A function that accepts a single argument, req the [Request](#) and returns the parsed cookie string, generally a list. Note that the original cookie string is available on the [Request](#) at the HTTP\_COOKIE field, get it with: req\$HTTP\_COOKIE

## Examples

```
func <- function(req) {  
  req$HTTP_COOKIE  
}  
  
parser <- as_cookie_parser(func)  
  
app <- Ambiorix$new()  
app$use(parser)
```

---

`as_cookie_preprocessor`*Define a Cookie Preprocessor*

---

**Description**

Identifies a function as a cookie preprocessor.

**Usage**

```
as_cookie_preprocessor(fn)
```

**Arguments**

`fn` A function that accepts the same arguments as the `cookie` method of the [Response](#) class (`name`, `value`, ...), and returns a modified value.

**Examples**

```
func <- \(name, value, ...) {  
  sprintf("prefix.%s", value)  
}
```

```
prep <- as_cookie_preprocessor(func)
```

```
app <- Ambiorix$new()  
app$use(prep)
```

---

`as_path_to_pattern`      *Path to pattern*

---

**Description**

Identify a function as a path to pattern function; a function that accepts a path and returns a matching pattern.

**Usage**

```
as_path_to_pattern(path)
```

**Arguments**

`path` A function that accepts a character vector of length 1 and returns another character vector of length 1.

---

as_renderer	<i>Create a Renderer</i>
-------------	--------------------------

---

**Description**

Create a custom renderer.

**Usage**

```
as_renderer(fn)
```

**Arguments**

fn	A function that accepts two arguments, the full path to the file to render, and the data to render.
----	---

---

content	<i>Content Headers</i>
---------	------------------------

---

**Description**

Convenient functions for more readable content type headers.

**Usage**

```
content_html()
```

```
content_plain()
```

```
content_json()
```

```
content_csv()
```

```
content_tsv()
```

```
content_protobuf()
```

**Examples**

```
list(  
  "Content-Type",  
  content_json()  
)
```

```
if(FALSE)  
  req$header(  
    "Content-Type",  
    content_json()  
  )
```

```
"Content-Type",  
content_json()  
)
```

---

create\_dockerfile      *Dockerfile*

---

### Description

Create the dockerfile required to run the application. The dockerfile created will install packages from RStudio Public Package Manager which comes with pre-built binaries that much improve the speed of building of Dockerfiles.

### Usage

```
create_dockerfile(port, host = "0.0.0.0")
```

### Arguments

port, host      Port and host to serve the application.

### Details

Reads the DESCRIPTION file of the project to produce the Dockerfile.

### Examples

```
## Not run: create_dockerfile()
```

---

default\_cookie\_parser      *Cookie Parser*

---

### Description

Parses the cookie string.

### Usage

```
default_cookie_parser(req)
```

### Arguments

req      A [Request](#).

### Value

A list of key value pairs or cookie values.

---

forward

*Forward Method*

---

### **Description**

Makes it such that the web server skips this method and uses the next one in line instead.

### **Usage**

```
forward()
```

### **Value**

An object of class forward.

### **Examples**

```
app <- Ambiorix$new()

app$get("/next", function(req, res){
  forward()
})

app$get("/next", function(req, res){
  res$send("Hello")
})

if(interactive())
  app$start()
```

---

import

*Import Files*

---

### **Description**

Import all R-files in a directory.

### **Usage**

```
import(...)
```

### **Arguments**

... Directory from which to import .R or .r files.

**Value**

Invisibly returns NULL.

**Examples**

```
## Not run: import("views")
```

---

is_renderer_obj	<i>Is Renderer</i>
-----------------	--------------------

---

**Description**

Check whether an object is a renderer.

**Usage**

```
is_renderer_obj(obj)
```

**Arguments**

obj	Object to check.
-----	------------------

**Value**

Boolean

---

jobj	<i>JSON Object</i>
------	--------------------

---

**Description**

Serialises an object to JSON in res\$render.

**Usage**

```
jobj(obj)
```

**Arguments**

obj	Object to serialise.
-----	----------------------

---

mockRequest	<i>Mock Request</i>
-------------	---------------------

---

**Description**

Mock a request, used for tests.

**Usage**

```
mockRequest(cookie = "", query = "", path = "/")
```

**Arguments**

cookie	Cookie string.
query	Query string.
path	Path string.

**Examples**

```
mockRequest()
```

---

new_log	<i>Logger</i>
---------	---------------

---

**Description**

Returns a new logger using the log package.

**Usage**

```
new_log(prefix = ">", write = FALSE, file = "ambiorix.log", sep = "")
```

**Arguments**

prefix	String to prefix all log messages.
write	Whether to write the log to the file.
file	Name of the file to dump the logs to, only used if write is TRUE.
sep	Separator between prefix and other flags and messages.

**Value**

An R& of class log::Logger.

**Examples**

```
log <- new_log()
log$log("Hello world")
```

---

parsers

*Parsers*

---

**Description**

Collection of parsers to translate request data.

**Usage**

```
parse_multipart(req)
parse_json(req, ...)
```

**Arguments**

req	The request object.
...	Additional arguments passed to the internal parsers.

**Value**

Returns the parsed value as a list or NULL if it failed to parse.

**Functions**

- [parse\\_multipart\(\)](#): Parse multipart/form-data using `mime::parse_multipart()`.
- [parse\\_json\(\)](#): Parse multipart/form-data using `jsonlite::fromJSON()`.

---

pre\_hook

*Pre Hook Response*

---

**Description**

Pre Hook Response

**Usage**

```
pre_hook(content, data)
```

**Arguments**

content	File content, a character vector.
data	A list of data passed to <code>glue::glue_data</code> .



Request

*Request***Description**

A request.

**Public fields**

HEADERS Headers from the request.

HTTP\_ACCEPT Content types to accept.

HTTP\_ACCEPT\_ENCODING Encoding of the request.

HTTP\_ACCEPT\_LANGUAGE Language of the request.

HTTP\_CACHE\_CONTROL Directorives for the cache (case-insensitive).

HTTP\_CONNECTION Controls whether the network connection stays open after the current transaction finishes.

HTTP\_COOKIE Cookie data.

HTTP\_HOST Host making the request.

HTTP\_SEC\_FETCH\_DEST Indicates the request's destination. That is the initiator of the original fetch request, which is where (and how) the fetched data will be used.

HTTP\_SEC\_FETCH\_MODE Indicates mode of the request.

HTTP\_SEC\_FETCH\_SITE Indicates the relationship between a request initiator's origin and the origin of the requested resource.

HTTP\_SEC\_FETCH\_USER Only sent for requests initiated by user activation, and its value will always be ?1.

HTTP\_UPGRADE\_INSECURE\_REQUESTS Signals that server supports upgrade.

HTTP\_USER\_AGENT User agent.

httpuv.version Version of httpuv.

PATH\_INFO Path of the request.

QUERY\_STRING Query string of the request.

REMOTE\_ADDR Remote address.

REMOTE\_PORT Remote port.

REQUEST\_METHOD Method of the request, e.g.: GET.

rook.errors Errors from rook.

rook.input Rook inputs.

rook.url\_scheme Rook url scheme.

rook.version Rook version.

SCRIPT\_NAME The initial portion of the request URL's "path" that corresponds to the application object, so that the application knows its virtual "location".

SERVER\_NAME Server name.  
SERVER\_PORT Server port  
CONTENT\_LENGTH Size of the message body.  
CONTENT\_TYPE Type of content of the request.  
HTTP\_REFERER Contains an absolute or partial address of the page that makes the request.  
body Request, an environment.  
query Parsed QUERY\_STRING, list.  
params A list of parameters.  
cookie Parsed HTTP\_COOKIE.

## Methods

### Public methods:

- [Request\\$new\(\)](#)
- [Request#print\(\)](#)
- [Request\\$set\(\)](#)
- [Request\\$get\(\)](#)
- [Request\\$get\\_header\(\)](#)
- [Request\\$parse\\_multipart\(\)](#)
- [Request\\$parse\\_json\(\)](#)
- [Request\\$clone\(\)](#)

### Method new():

*Usage:*

Request\$new(req)

*Arguments:*

req Original request (environment).

*Details:* Constructor

### Method print():

*Usage:*

Request#print()

*Details:* Print

### Method set():

*Usage:*

Request\$set(name, value)

*Arguments:*

name Name of the variable.

value Value of the variable.

*Details:* Set Data

*Returns:* Invisible returns self.

**Method** `get()`:

*Usage:*

`Request$get(name)`

*Arguments:*

name Name of the variable to get.

*Details:* Get data

**Method** `get_header()`:

*Usage:*

`Request$get_header(name)`

*Arguments:*

name Name of the header

*Details:* Get Header

**Method** `parse_multipart()`:

*Usage:*

`Request$parse_multipart()`

*Details:* Parse Multipart encoded data

**Method** `parse_json()`:

*Usage:*

`Request$parse_json(...)`

*Arguments:*

... Arguments passed to [jsonlite::fromJSON\(\)](#).

*Details:* Parse JSON encoded data

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

*Usage:*

`Request$clone(deep = FALSE)`

*Arguments:*

deep Whether to make a deep clone.

---

Response

*Response*

---

## Description

Response class to generate responses sent from the server.

## Active bindings

status Status of the response, defaults to 200L.

headers Named list of headers.

## Methods

### Public methods:

- [Response\\$set\\_status\(\)](#)
- [Response\\$send\(\)](#)
- [Response\\$sendf\(\)](#)
- [Response\\$text\(\)](#)
- [Response\\$send\\_file\(\)](#)
- [Response\\$redirect\(\)](#)
- [Response\\$render\(\)](#)
- [Response\\$json\(\)](#)
- [Response\\$csv\(\)](#)
- [Response\\$tsv\(\)](#)
- [Response\\$htmlwidget\(\)](#)
- [Response\\$md\(\)](#)
- [Response\\$png\(\)](#)
- [Response\\$jpeg\(\)](#)
- [Response\\$image\(\)](#)
- [Response\\$ggplot2\(\)](#)
- [Response\\$print\(\)](#)
- [Response\\$set\(\)](#)
- [Response\\$get\(\)](#)
- [Response\\$header\(\)](#)
- [Response\\$header\\_content\\_json\(\)](#)
- [Response\\$header\\_content\\_html\(\)](#)
- [Response\\$header\\_content\\_plain\(\)](#)
- [Response\\$header\\_content\\_csv\(\)](#)
- [Response\\$header\\_content\\_tsv\(\)](#)
- [Response\\$get\\_headers\(\)](#)
- [Response\\$get\\_header\(\)](#)

- [Response\\$set\\_headers\(\)](#)
- [Response\\$set\\_header\(\)](#)
- [Response\\$pre\\_render\\_hook\(\)](#)
- [Response\\$post\\_render\\_hook\(\)](#)
- [Response\\$cookie\(\)](#)
- [Response\\$clear\\_cookie\(\)](#)
- [Response\\$clone\(\)](#)

**Method** `set_status()`:

*Usage:*

`Response$set_status(status)`

*Arguments:*

`status` An integer defining the status.

*Details:* Set the status of the response.

**Method** `send()`:

*Usage:*

`Response$send(body, headers = NULL, status = NULL)`

*Arguments:*

`body` Body of the response.

`headers` HTTP headers to set.

`status` Status of the response, if NULL uses `self$status`.

*Details:* Send a plain HTML response.

**Method** `sendf()`:

*Usage:*

`Response$sendf(body, ..., headers = NULL, status = NULL)`

*Arguments:*

`body` Body of the response.

`...` Passed to `...` of `sprintf`.

`headers` HTTP headers to set.

`status` Status of the response, if NULL uses `self$status`.

*Details:* Send a plain HTML response, pre-processed with `sprintf`.

**Method** `text()`:

*Usage:*

`Response$text(body, headers = NULL, status = NULL)`

*Arguments:*

`body` Body of the response.

`headers` HTTP headers to set.

`status` Status of the response, if NULL uses `self$status`.

*Details:* Send a plain text response.

**Method** `send_file()`:

*Usage:*

```
Response$send_file(file, headers = NULL, status = NULL)
```

*Arguments:*

`file` File to send.

`headers` HTTP headers to set.

`status` Status of the response.

*Details:* Send a file.

**Method** `redirect()`:

*Usage:*

```
Response$redirect(path, status = NULL)
```

*Arguments:*

`path` Path or URL to redirect to.

`status` Status of the response, if NULL uses `self$status`.

*Details:* Redirect to a path or URL.

**Method** `render()`:

*Usage:*

```
Response$render(file, data = list(), headers = NULL, status = NULL)
```

*Arguments:*

`file` Template file.

`data` List to fill [% tags %].

`headers` HTTP headers to set.

`status` Status of the response, if NULL uses `self$status`.

*Details:* Render a template file.

**Method** `json()`:

*Usage:*

```
Response$json(body, headers = NULL, status = NULL, ...)
```

*Arguments:*

`body` Body of the response.

`headers` HTTP headers to set.

`status` Status of the response, if NULL uses `self$status`.

`...` Additional arguments passed to the serialiser.

*Details:* Render an object as JSON.

**Method** `csv()`:

*Usage:*

```
Response$csv(data, name = "data", status = NULL, ...)
```

*Arguments:*

data Data to convert to CSV.  
name Name of the file.  
status Status of the response, if NULL uses self\$status.  
... Additional arguments passed to `readr::format_csv()`.

*Details:* Sends a comma separated value file

**Method** tsv():*Usage:*

```
Response$tsv(data, name = "data", status = NULL, ...)
```

*Arguments:*

data Data to convert to CSV.  
name Name of the file.  
status Status of the response, if NULL uses self\$status.  
... Additional arguments passed to `readr::format_tsv()`.

*Details:* Sends a tab separated value file

**Method** htmlwidget():*Usage:*

```
Response$htmlwidget(widget, status = NULL, ...)
```

*Arguments:*

widget The widget to use.  
status Status of the response, if NULL uses self\$status.  
... Additional arguments passed to `htmlwidgets::saveWidget()`.

*Details:* Sends an htmlwidget.

**Method** md():*Usage:*

```
Response$md(file, data = list(), headers = NULL, status = NULL)
```

*Arguments:*

file Template file.  
data List to fill [% tags %].  
headers HTTP headers to set.  
status Status of the response, if NULL uses self\$status.

*Details:* Render a markdown file.

**Method** png():*Usage:*

```
Response$png(file)
```

*Arguments:*

file Path to local file.

*Details:* Send a png file

**Method jpeg():**

*Usage:*

Response\$jpeg(file)

*Arguments:*

file Path to local file.

*Details:* Send a jpeg file

**Method image():**

*Usage:*

Response\$image(file)

*Arguments:*

file Path to local file.

*Details:* Send an image Similar to png and jpeg methods but guesses correct method based on file extension.

**Method ggplot2():**

*Usage:*

Response\$ggplot2(plot, ..., type = c("png", "jpeg"))

*Arguments:*

plot Ggplot2 plot object.

... Passed to [ggplot2::ggsave\(\)](#)

type Type of image to save.

*Details:* Ggplot2

**Method print():**

*Usage:*

Response\$print()

*Details:* Print

**Method set():**

*Usage:*

Response\$set(name, value)

*Arguments:*

name Name of the variable.

value Value of the variable.

*Details:* Set Data

*Returns:* Invisible returns self.

**Method get():**

*Usage:*



Response\$get(name)

*Arguments:*

name Name of the variable to get.

*Details:* Get data

**Method** header():

*Usage:*

Response\$header(name, value)

*Arguments:*

name, value Name and value of the header.

*Details:* Add headers to the response.

*Returns:* Invisibly returns self.

**Method** header\_content\_json():

*Usage:*

Response\$header\_content\_json()

*Details:* Set Content Type to JSON

*Returns:* Invisibly returns self.

**Method** header\_content\_html():

*Usage:*

Response\$header\_content\_html()

*Details:* Set Content Type to HTML

*Returns:* Invisibly returns self.

**Method** header\_content\_plain():

*Usage:*

Response\$header\_content\_plain()

*Details:* Set Content Type to Plain Text

*Returns:* Invisibly returns self.

**Method** header\_content\_csv():

*Usage:*

Response\$header\_content\_csv()

*Details:* Set Content Type to CSV

*Returns:* Invisibly returns self.

**Method** header\_content\_tsv():

*Usage:*

Response\$header\_content\_tsv()

*Details:* Set Content Type to TSV

*Returns:* Invisibly returns self.

**Method** `get_headers()`:

*Usage:*

`Response$get_headers()`

*Details:* Get headers Returns the list of headers currently set.

**Method** `get_header()`:

*Usage:*

`Response$get_header(name)`

*Arguments:*

`name` Name of the header to return.

*Details:* Get a header Returns a single header currently, NULL if not set.

**Method** `set_headers()`:

*Usage:*

`Response$set_headers(headers)`

*Arguments:*

`headers` A named list of headers to set.

*Details:* Set headers

**Method** `set_header()`:

*Usage:*

`Response$set_header(name, value)`

*Arguments:*

`name` Name of the header.

`value` Value to set.

*Details:* Set a Header

*Returns:* Invisible returns self.

**Method** `pre_render_hook()`:

*Usage:*

`Response$pre_render_hook(hook)`

*Arguments:*

`hook` A function that accepts at least 4 arguments:

- `self`: The Request class instance.
- `content`: File content a vector of character string, content of the template.
- `data`: list passed from render method.
- `ext`: File extension of the template file.

This function is used to add pre-render hooks to the render method. The function should return an object of class `responsePreHook` as obtained by `pre_hook()`. This is meant to be used by middlewares to, if necessary, pre-process rendered data.

Include `...` in your hook to ensure it will handle potential updates to hooks in the future.

*Details:* Add a pre render hook. Runs before the render and send\_file method.

*Returns:* Invisible returns self.

**Method** post\_render\_hook():

*Usage:*

```
Response$post_render_hook(hook)
```

*Arguments:*

hook A function to run after the rendering of HTML. It should accept at least 3 arguments:

- self: The Request class instance.
- content: File content a vector of character string, content of the template.
- ext: File extension of the template file.

Include ... in your hook to ensure it will handle potential updates to hooks in the future.

*Details:* Post render hook.

*Returns:* Invisible returns self.

**Method** cookie():

*Usage:*

```
Response$cookie(
  name,
  value,
  expires = getOption("ambiorix.cookie.expire"),
  max_age = getOption("ambiorix.cookie.maxage"),
  domain = getOption("ambiorix.cookie.domain"),
  path = getOption("ambiorix.cookie.path", "/"),
  secure = getOption("ambiorix.cookie.secure", TRUE),
  http_only = getOption("ambiorix.cookie.httponly", TRUE),
  same_site = getOption("ambiorix.cookie.savesite")
)
```

*Arguments:*

name Name of the cookie.

value value of the cookie.

expires Expiry, if an integer assumes it's the number of seconds from now. Otherwise accepts an object of class POSIXct or Date. If a character string then it is set as-is and not pre-processed. If unspecified, the cookie becomes a session cookie. A session finishes when the client shuts down, after which the session cookie is removed.

max\_age Indicates the number of seconds until the cookie expires. A zero or negative number will expire the cookie immediately. If both expires and max\_age are set, the latter has precedence.

domain Defines the host to which the cookie will be sent. If omitted, this attribute defaults to the host of the current document URL, not including subdomains.

path Indicates the path that must exist in the requested URL for the browser to send the Cookie header.

secure Indicates that the cookie is sent to the server only when a request is made with the https: scheme (except on localhost), and therefore, is more resistant to man-in-the-middle attacks.

`http_only` Forbids JavaScript from accessing the cookie, for example, through the `document.cookie` property.

`same_site` Controls whether or not a cookie is sent with cross-origin requests, providing some protection against cross-site request forgery attacks (CSRF). Accepts `Strict`, `Lax`, or `None`.

*Details:* Set a cookie Overwrites existing cookie of the same name.

*Returns:* Invisibly returns self.

**Method** `clear_cookie()`:

*Usage:*

```
Response$clear_cookie(name)
```

*Arguments:*

`name` Name of the cookie to clear.

*Details:* Clear a cookie Clears the value of a cookie.

*Returns:* Invisibly returns self.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

responses

*Plain Responses*

## Description

Plain HTTP Responses.

## Usage

```
response(body, headers = list(), status = 200L)
```

```
response_404(
  body = "404: Not found",
  headers = list(`Content-Type` = content_html()),
  status = 404L
)
```

```
response_500(
  body = "500: Server Error",
  headers = list(`Content-Type` = content_html()),
  status = 500L
)
```

**Arguments**

body	Body of response.
headers	HTTP headers.
status	Response status

**Examples**

```
app <- Ambiorix$new()

# html
app$get("/", function(req, res){
  res$send("hello!")
})

# text
app$get("/text", function(req, res){
  res$text("hello!")
})

if(interactive())
  app$start()
```

---

robj

*R Object*


---

**Description**

Treats a data element rendered in a response (`res$render`) as a data object and ultimately uses `dput()`.

**Usage**

```
robj(obj)
```

**Arguments**

obj	R object to treat.
-----	--------------------

**Details**

For instance in a template, `x <- [% var %]` will not work with `res$render(data=list(var = "hello"))` because this will be replace like `x <- hello` (missing quote): breaking the template. Using `robj` one would obtain `x <- "hello"`.

---

Router

*Router*

---

### Description

Web server.

### Super class

`ambiorix::Routing` -> Router

### Public fields

error 500 response when the route errors, must a handler function that accepts the request and the response, by default uses `response_500()`.

### Methods

#### Public methods:

- `Router$new()`
- `Router$print()`
- `Router$clone()`

#### Method `new()`:

*Usage:*

`Router$new(path)`

*Arguments:*

path The base path of the router.

*Details:* Define the base route.

#### Method `print()`:

*Usage:*

`Router$print()`

*Details:* Print

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

*Usage:*

`Router$clone(deep = FALSE)`

*Arguments:*

deep Whether to make a deep clone.

**Examples**

```
# log
logger <- new_log()
# router
# create router
router <- Router$new("/users")

router$get("/", function(req, res){
  res$send("List of users")
})

router$get("/:id", function(req, res){
  logger$log("Return user id:", req$params$id)
  res$send(req$params$id)
})

router$get("/:id/profile", function(req, res){
  msg <- sprintf("This is the profile of user #%", req$params$id)
  res$send(msg)
})

# core app
app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Home!")
})

# mount the router
app$use(router)

if(interactive())
  app$start()
```

---

Routing

*Core Routing Class*

---

**Description**

Core routing class. Do not use directly, see [Ambiorix](#), and [Router](#).

**Public fields**

error Error handler.

**Methods****Public methods:**

- `Routing$new()`
- `Routing$get()`
- `Routing$put()`
- `Routing$patch()`
- `Routing$delete()`
- `Routing$post()`
- `Routing$options()`
- `Routing$all()`
- `Routing$receive()`
- `Routing$print()`
- `Routing$engine()`
- `Routing$use()`
- `Routing$get_routes()`
- `Routing$get_receivers()`
- `Routing$get_middleware()`
- `Routing$clone()`

**Method** `new()`:

*Usage:*

```
Routing$new(path = "")
```

*Arguments:*

path Prefix path.

*Details:* Initialise

**Method** `get()`:

*Usage:*

```
Routing$get(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to, : defines a parameter.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: `response()`.

error Handler function to run on error.

*Details:* GET Method

Add routes to listen to.

*Examples:*

```
app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start()
```



**Method put():**

*Usage:*

```
Routing$put(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to, : defines a parameter.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

error Handler function to run on error.

*Details:* PUT Method

Add routes to listen to.

**Method patch():**

*Usage:*

```
Routing$patch(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to, : defines a parameter.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

error Handler function to run on error.

*Details:* PATCH Method

Add routes to listen to.

**Method delete():**

*Usage:*

```
Routing$delete(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to, : defines a parameter.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

error Handler function to run on error.

*Details:* DELETE Method

Add routes to listen to.

**Method post():**

*Usage:*

```
Routing$post(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

error Handler function to run on error.

*Details:* POST Method  
Add routes to listen to.

**Method** options():

*Usage:*

```
Routing$options(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

error Handler function to run on error.

*Details:* OPTIONS Method  
Add routes to listen to.

**Method** all():

*Usage:*

```
Routing$all(path, handler, error = NULL)
```

*Arguments:*

path Route to listen to.

handler Function that accepts the request and returns an object describing an httpuv response, e.g.: [response\(\)](#).

error Handler function to run on error.

*Details:* All Methods  
Add routes to listen to for all methods GET, POST, PUT, DELETE, and PATCH.

**Method** receive():

*Usage:*

```
Routing$receive(name, handler)
```

*Arguments:*

name Name of message.

handler Function to run when message is received.

*Details:* Receive Websocket Message

*Examples:*

```
app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

app$receive("hello", function(msg, ws){
  print(msg) # print msg received

  # send a message back
```

```
    ws$send("hello", "Hello back! (sent from R)")
  })
```

```
if(interactive())
  app$start()
```

**Method print():**

*Usage:*

```
Routing#print()
```

*Details:* Print

**Method engine():**

*Usage:*

```
Routing$engine(engine)
```

*Details:* Engine to use for rendering templates.

**Method use():**

*Usage:*

```
Routing$use(use)
```

*Arguments:*

`use` Either a router as returned by [Router](#), a function to use as middleware, or a list of functions. If a function is passed, it must accept two arguments (the request, and the response): this function will be executed every time the server receives a request. *Middleware may but does not have to return a response, unlike other methods such as get* Note that multiple routers and middlewares can be used.

*Details:* Use a router or middleware

**Method get\_routes():**

*Usage:*

```
Routing$get_routes()
```

*Details:* Get the routes

**Method get\_receivers():**

*Usage:*

```
Routing$get_receivers()
```

*Details:* Get the receivers

**Method get\_middleware():**

*Usage:*

```
Routing$get_middleware()
```

*Details:* Get the middleware

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

**Examples**

```
## -----
## Method `Routing$get`
## -----

app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

if(interactive())
  app$start()

## -----
## Method `Routing$receive`
## -----

app <- Ambiorix$new()

app$get("/", function(req, res){
  res$send("Using {ambiorix}!")
})

app$receive("hello", function(msg, ws){
  print(msg) # print msg received

  # send a message back
  ws$send("hello", "Hello back! (sent from R)")
})

if(interactive())
  app$start()
```

---

**serialise***Serialise to JSON*

---

**Description**

Serialise an object to JSON. Default serialiser can be change by setting the AMBIORIX\_SERIALISER option to the desired function.

**Usage**

```
serialise(data, ...)
```

**Arguments**

<code>data</code>	Data to serialise.
<code>...</code>	Passed to serialiser.

**Examples**

```
## Not run: serialise(cars)
```

---

set_log	<i>Customise logs</i>
---------	-----------------------

---

**Description**

Customise the internal logs used by Ambiorix.

**Usage**

```
set_log_info(log)
```

```
set_log_success(log)
```

```
set_log_error(log)
```

**Arguments**

log            An object of class `Logger`, see [log::Logger](#).

---

set_params	<i>Set Parameters</i>
------------	-----------------------

---

**Description**

Set the query's parameters.

**Usage**

```
set_params(path, route = NULL)
```

**Arguments**

path            Correspond's the the requests' `PATH_INFO`

route           See `Route`

**Value**

Parameter list

stop\_all                      *Stop*

---

**Description**

Stop all servers.

**Usage**

stop\_all()

---

token\_create                 *Token*

---

**Description**

Create a token

**Usage**

token\_create(n = 16L)

**Arguments**

n                              Number of bytes.

---

use\_html\_template            *HTML Template*

---

**Description**

Use `htmltools::htmlTemplate()` as renderer. Passe to use method.

**Usage**

use\_html\_template()

---

Websocket

*Websocket*

---

## Description

Handle websocket messages.

## Methods

### Public methods:

- [Websocket\\$new\(\)](#)
- [Websocket\\$send\(\)](#)
- [Websocket\\$print\(\)](#)
- [Websocket\\$clone\(\)](#)

### Method new():

*Usage:*

Websocket\$new(ws)

*Arguments:*

ws

*Details:* Constructor

### Method send():

*Usage:*

Websocket\$send(name, message)

*Arguments:*

name Name, identifier, of the message.

message Content of the message, anything that can be serialised to JSON.

*Details:* Send a message

### Method print():

*Usage:*

Websocket\$print()

*Details:* Print

### 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.

---

websocket_client	<i>Websocket Client</i>
------------------	-------------------------

---

**Description**

Handle ambiorix websocket client.

**Usage**

```
copy_websocket_client(path)
```

```
get_websocket_client()
```

**Arguments**

path	Path to copy the file to.
------	---------------------------

**Functions**

- `copy_websocket_client` Copies the websocket client file, useful when ambiorix was not setup with the ambiorix generator.
- `get_websocket_client` Retrieves the full path to the local websocket client.



# Index

## \* export

- Routing, [31](#)
- Ambiorix, [2](#), [31](#)
- ambiorix::Routing, [2](#), [30](#)
- as\_cookie\_parser, [9](#)
- as\_cookie\_preprocessor, [10](#)
- as\_path\_to\_pattern, [10](#)
- as\_renderer, [11](#)
- content, [11](#)
- content\_csv (content), [11](#)
- content\_html (content), [11](#)
- content\_json (content), [11](#)
- content\_plain (content), [11](#)
- content\_protobuf (content), [11](#)
- content\_tsv (content), [11](#)
- copy\_websocket\_client (websocket\_client), [40](#)
- create\_dockerfile, [12](#)
- default\_cookie\_parser, [12](#)
- dput(), [29](#)
- forward, [13](#)
- get\_websocket\_client (websocket\_client), [40](#)
- ggplot2::ggsave(), [24](#)
- htmltools::htmlTemplate(), [38](#)
- htmlwidgets::saveWidget(), [23](#)
- import, [13](#)
- is\_renderer\_obj, [14](#)
- jobj, [14](#)
- jsonlite::fromJSON(), [16](#), [19](#)
- log::Logger, [37](#)
- mime::parse\_multipart(), [16](#)
- mockRequest, [15](#)
- new\_log, [15](#)
- parse\_json (parsers), [16](#)
- parse\_json(), [16](#)
- parse\_multipart (parsers), [16](#)
- parse\_multipart(), [16](#)
- parsers, [16](#)
- pre\_hook, [16](#)
- pre\_hook(), [26](#)
- readr::format\_csv(), [23](#)
- readr::format\_tsv(), [23](#)
- Request, [9](#), [12](#), [17](#)
- Response, [10](#), [20](#)
- response (responses), [28](#)
- response(), [4](#), [32–34](#)
- response\_404 (responses), [28](#)
- response\_404(), [3](#)
- response\_500 (responses), [28](#)
- response\_500(), [3](#), [30](#)
- responses, [28](#)
- robject, [29](#)
- Router, [30](#), [31](#), [35](#)
- Routing, [31](#)
- serialise, [36](#)
- set\_log, [37](#)
- set\_log\_error (set\_log), [37](#)
- set\_log\_info (set\_log), [37](#)
- set\_log\_success (set\_log), [37](#)
- set\_params, [37](#)
- stop\_all, [38](#)
- token\_create, [38](#)
- use\_html\_template, [38](#)
- Websocket, [39](#)
- websocket\_client, [40](#)