

# Package ‘ambiorix’

April 25, 2026

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

**Version** 3.0.0

**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

**RoxygenNote** 7.3.3

**Imports** log, cli, glue, httpuv, methods, assertthat, webutils (>= 1.2.0), yyjsonr (>= 0.1.20), mime (>= 0.13), promises (>= 1.5.0)

**Suggests** readr, readxl, ggplot2, jsonlite, websocket, htmltools, commonmark, htmlwidgets, testthat (>= 3.0.0)

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

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

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** John Coene [aut] (ORCID: <<https://orcid.org/0000-0002-6637-4107>>),  
Opifex [fnd],  
Kennedy Mwavu [cre] (ORCID: <<https://orcid.org/0009-0006-3157-7234>>),  
Julio Collazos [ctb] (ORCID: <<https://orcid.org/0009-0006-5503-0997>>)

**Maintainer** Kennedy Mwavu <[mwavukennedy@gmail.com](mailto:mwavukennedy@gmail.com)>

**Repository** CRAN

**Date/Publication** 2026-04-25 04:00:03 UTC

## Contents

|                                  |    |
|----------------------------------|----|
| Ambiorix . . . . .               | 2  |
| as_cookie_parser . . . . .       | 9  |
| as_cookie_preprocessor . . . . . | 10 |

|                                 |           |
|---------------------------------|-----------|
| as_path_to_pattern . . . . .    | 10        |
| as_renderer . . . . .           | 11        |
| content . . . . .               | 12        |
| default_cookie_parser . . . . . | 12        |
| forward . . . . .               | 13        |
| import . . . . .                | 14        |
| jobj . . . . .                  | 15        |
| mockRequest . . . . .           | 15        |
| new_log . . . . .               | 16        |
| parse_form_urlencoded . . . . . | 16        |
| parse_json . . . . .            | 20        |
| parse_multipart . . . . .       | 24        |
| pre_hook . . . . .              | 29        |
| Request . . . . .               | 30        |
| Response . . . . .              | 33        |
| responses . . . . .             | 40        |
| robject . . . . .               | 42        |
| Router . . . . .                | 42        |
| Routing . . . . .               | 44        |
| routing-http-methods . . . . .  | 49        |
| serialise . . . . .             | 50        |
| set_log . . . . .               | 52        |
| stop_all . . . . .              | 52        |
| token_create . . . . .          | 53        |
| use_html_template . . . . .     | 54        |
| Websocket . . . . .             | 54        |
| websocket_client . . . . .      | 56        |
| <b>Index</b>                    | <b>58</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 <- function(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 <- function(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 <- function(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 <- function(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`

## Value

Object of class "cookieParser".

## 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.

**Value**

Object of class "cookiePreprocessor".

**Examples**

```
func <- function(name, value, ...) {  
  sprintf("prefix.%s", value)  
}  
  
prep <- as_cookie_preprocessor(func)  
  
app <- Ambiorix$new()  
app$use(prepare)
```

---

`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.

**Value**

Object of class "pathToPattern".

**Examples**

```
fn <- function(path) {
  pattern <- gsub(":[^/]+)", "(\\\\\\\\w+)", path)
  paste0("^", pattern, "$")
}

path_to_pattern <- as_path_to_pattern(fn)

path <- "/dashboard/profile/:user_id"
pattern <- path_to_pattern(path) # "^/dashboard/profile/(\\\\w+)$"
```

---

as\_renderer

*Create a Renderer*

---

**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.

**Value**

A renderer function.

**Examples**

```
if (interactive()) {
  fn <- function(path, data) {
    # ...
  }

  as_renderer(fn)
}
```

---

content

*Content Headers*

---

### **Description**

Convenient functions for more readable content type headers.

### **Usage**

content\_html()

content\_plain()

content\_json()

content\_csv()

content\_tsv()

content\_protobuf()

### **Value**

Length 1 character vector.

### **Examples**

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

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

---

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.

**Examples**

```
if (interactive()) {
  library(ambiorix)

  #' Handle GET at '/greet'
  #'
  #' @export
  say_hello <- function(req, res) {
    cookies <- default_cookie_parser(req)
    print(cookies)

    res$send("hello there!")
  }

  app <- Ambiorix$new()
  app$get("/greet", say_hello)
  app$start()
}
```

---

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**

```
if (interactive()) {
  import("views")
}
```

---

jobj

*JSON Object*

---

### Description

Serialises an object to JSON in res\$render.

### Usage

```
jobj(obj)
```

### Arguments

obj            Object to serialise.

### Value

Object of class "jobj".

### Examples

```
if (interactive()) {  
  l <- list(a = "hello", b = 2L, c = 3)  
  jobj(l)  
}
```

---

mockRequest

*Mock Request*

---

### Description

Mock a request, used for tests.

### Usage

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

### Arguments

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

### Value

A Request object.

**Examples**

```
mockRequest()
```

---

```
new_log
```

```
Logger
```

---

**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")
```

---

```
parse_form_urlencoded Parse application/x-www-form-urlencoded data
```

---

**Description**

This function parses application/x-www-form-urlencoded data, typically used in form submissions.

**Usage**

```
parse_form_urlencoded(req, ...)
```

**Arguments**

req                    The request object.  
 ...                    Additional parameters passed to the parser function.

**Details****Overriding Default Parser:**

By default, `parse_form_urlencoded()` uses `webutils::parse_http()`. You can override this globally by setting the `AMBIORIX_FORM_URLENCODED_PARSER` option:

```
options(AMBIORIX_FORM_URLENCODED_PARSER = my_other_custom_parser)
```

Your custom parser function *MUST* accept the following parameters:

1. `body`: Raw vector containing the form data.
2. `...` : Additional optional parameters.

**Value**

A list of parsed form fields, with each key representing a form field name and each value representing the form field's value.

Named list

**See Also**

[parse\\_multipart\(\)](#), [parse\\_json\(\)](#)

**Examples**

```
if (interactive()) {
  library(ambiorix)
  library(htmltools)
  library(readxl)

  page_links <- function() {
    Map(
      f = function(href, label) {
        tags$a(href = href, label)
      },
      c("/", "/about", "/contact"),
      c("Home", "About", "Contact")
    )
  }

  forms <- function() {
    form1 <- tags$form(
      action = "/url-form-encoded",
      method = "POST",
      enctype = "application/x-www-form-urlencoded",
      tags$h4("form-url-encoded:"),
      tags$label(`for` = "first_name", "First Name"),

```

```

tags$input(id = "first_name", name = "first_name", value = "John"),
tags$label(`for` = "last_name", "Last Name"),
tags$input(id = "last_name", name = "last_name", value = "Coene"),
tags$button(type = "submit", "Submit")
)

form2 <- tags$form(
  action = "/multipart-form-data",
  method = "POST",
  enctype = "multipart/form-data",
tags$h4("multipart/form-data:"),
tags$label(`for` = "email", "Email"),
tags$input(id = "email", name = "email", value = "john@mail.com"),
tags$label(`for` = "framework", "Framework"),
tags$input(id = "framework", name = "framework", value = "ambiorix"),
tags$label(`for` = "file", "Upload CSV file"),
tags$input(type = "file", id = "file", name = "file", accept = ".csv"),
tags$label(`for` = "file2", "Upload xlsx file"),
tags$input(type = "file", id = "file2", name = "file2", accept = ".xlsx"),
tags$button(type = "submit", "Submit")
)

tagList(form1, form2)
}

home_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("hello, world!"),
    forms()
  )

  res$send(html)
}

home_post <- function(req, res) {
  body <- parse_json(req)
  # print(body)

  response <- list(
    code = 200L,
    msg = "hello, world"
  )
  res$json(response)
}

url_form_encoded_post <- function(req, res) {
  body <- parse_form_urlencoded(req)
  # print(body)

  list_items <- lapply(
    X = names(body),
    FUN = function(nm) {

```

```

        tags$li(
          nm,
          ":",
          body[[nm]]
        )
      }
    )
    input_vals <- tags$ul(list_items)

    html <- tagList(
      page_links(),
      tags$h3("Request processed"),
      input_vals
    )

    res$send(html)
  }

multipart_form_data_post <- function(req, res) {
  body <- parse_multipart(req)

  list_items <- lapply(
    X = names(body),
    FUN = function(nm) {
      field <- body[[nm]]

      # if 'field' is a file, parse it & print on console:
      is_file <- "filename" %in% names(field)
      is_csv <- is_file && identical(field[["content_type"]], "text/csv")
      is_xlsx <- is_file &&
        identical(
          field[["content_type"]],
          "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
        )

      if (is_file) {
        file_path <- tempfile()
        writeBin(object = field$value, con = file_path)
        on.exit(unlink(x = file_path))
      }

      if (is_csv) {
        # print(read.csv(file = file_path))
      }

      if (is_xlsx) {
        # print(readxl::read_xlsx(path = file_path))
      }

      tags$li(
        nm,
        ":",
        if (is_file) "printed on console" else field
      )
    }
  )
}

```

```
    )
  }
)
input_vals <- tags$ul(list_items)

html <- tagList(
  page_links(),
  tags$h3("Request processed"),
  input_vals
)

res$send(html)
}

about_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("About Us")
  )
  res$send(html)
}

contact_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("Get In Touch!")
  )
  res$send(html)
}

app <- Ambiorix$new(port = 5000L)

app$
  get("/", home_get)$
  post("/", home_post)$
  get("/about", about_get)$
  get("/contact", contact_get)$
  post("/url-form-encoded", url_form_encoded_post)$
  post("/multipart-form-data", multipart_form_data_post)

app$start()
}
```

---

parse\_json

*Parse application/json data*

---

### **Description**

This function parses JSON data from the request body.

**Usage**

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

**Arguments**

|     |  |
|-----|--|
| req | The request object.                                  |
| ... | Additional parameters passed to the parser function. |

**Details****Overriding Default Parser:**

By default, `parse_json()` uses `yyjsonr::read_json_raw()` for JSON parsing. You can override this globally by setting the `AMBIORIX_JSON_PARSER` option:

```
my_json_parser <- function(body, ...) {  
  txt <- rawToChar(body)  
  jsonlite::fromJSON(txt, ...)  
}  
options(AMBIORIX_JSON_PARSER = my_json_parser)
```

Your custom parser *MUST* accept the following parameters:

1. `body`: Raw vector containing the JSON data.
2. `...`: Additional optional parameters.

**Value**

An R object (e.g., list or data frame) parsed from the JSON data.

Named list

**See Also**

[parse\\_multipart\(\)](#), [parse\\_form\\_urlencoded\(\)](#)

**Examples**

```
if (interactive()) {  
  library(ambiorix)  
  library(htmltools)  
  library(readxl)  
  
  page_links <- function() {  
    Map(  
      f = function(href, label) {  
        tags$a(href = href, label)  
      },  
      c("/", "/about", "/contact"),  
      c("Home", "About", "Contact")  
    )  
  }  
}
```

```

forms <- function() {
  form1 <- tags$form(
    action = "/url-form-encoded",
    method = "POST",
    enctype = "application/x-www-form-urlencoded",
    tags$h4("form-url-encoded:"),
    tags$label(`for` = "first_name", "First Name"),
    tags$input(id = "first_name", name = "first_name", value = "John"),
    tags$label(`for` = "last_name", "Last Name"),
    tags$input(id = "last_name", name = "last_name", value = "Coene"),
    tags$button(type = "submit", "Submit")
  )

  form2 <- tags$form(
    action = "/multipart-form-data",
    method = "POST",
    enctype = "multipart/form-data",
    tags$h4("multipart/form-data:"),
    tags$label(`for` = "email", "Email"),
    tags$input(id = "email", name = "email", value = "john@email.com"),
    tags$label(`for` = "framework", "Framework"),
    tags$input(id = "framework", name = "framework", value = "ambiorix"),
    tags$label(`for` = "file", "Upload CSV file"),
    tags$input(type = "file", id = "file", name = "file", accept = ".csv"),
    tags$label(`for` = "file2", "Upload xlsx file"),
    tags$input(type = "file", id = "file2", name = "file2", accept = ".xlsx"),
    tags$button(type = "submit", "Submit")
  )

  tagList(form1, form2)
}

home_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("hello, world!"),
    forms()
  )

  res$send(html)
}

home_post <- function(req, res) {
  body <- parse_json(req)
  # print(body)

  response <- list(
    code = 200L,
    msg = "hello, world"
  )
  res$json(response)
}

```

```

url_form_encoded_post <- function(req, res) {
  body <- parse_form_urlencoded(req)
  # print(body)

  list_items <- lapply(
    X = names(body),
    FUN = function(nm) {
      tags$li(
        nm,
        ":",
        body[[nm]]
      )
    }
  )
  input_vals <- tags$ul(list_items)

  html <- tagList(
    page_links(),
    tags$h3("Request processed"),
    input_vals
  )

  res$send(html)
}

multipart_form_data_post <- function(req, res) {
  body <- parse_multipart(req)

  list_items <- lapply(
    X = names(body),
    FUN = function(nm) {
      field <- body[[nm]]

      # if 'field' is a file, parse it & print on console:
      is_file <- "filename" %in% names(field)
      is_csv <- is_file && identical(field[["content_type"]], "text/csv")
      is_xlsx <- is_file &&
        identical(
          field[["content_type"]],
          "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
        )

      if (is_file) {
        file_path <- tempfile()
        writeBin(object = field$value, con = file_path)
        on.exit(unlink(x = file_path))
      }

      if (is_csv) {
        # print(read.csv(file = file_path))
      }

      if (is_xlsx) {

```

```

    # print(readxl::read_xlsx(path = file_path))
  }

  tags$li(
    nm,
    ":",
    if (is_file) "printed on console" else field
  )
}
)
input_vals <- tags$ul(list_items)

html <- tagList(
  page_links(),
  tags$h3("Request processed"),
  input_vals
)

res$send(html)
}

about_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("About Us")
  )
  res$send(html)
}

contact_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("Get In Touch!")
  )
  res$send(html)
}

app <- Ambiorix$new(port = 5000L)

app$
  get("/")$home_get$
  post("/")$home_post$
  get("/about")$about_get$
  get("/contact")$contact_get$
  post("/url-form-encoded", url_form_encoded_post)$
  post("/multipart-form-data", multipart_form_data_post)

app$start()
}

```

**Description**

Parses multipart form data, including file uploads, and returns the parsed fields as a list.

**Usage**

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

**Arguments**

|     |  |
|-----|--|
| req | The request object.                                  |
| ... | Additional parameters passed to the parser function. |

**Details**

If a field is a file upload it is returned as a named list with:

- `value`: Raw vector representing the file contents. You must process this further (eg. convert to `data.frame`). See the examples section.
- `content_disposition`: Typically "form-data", indicating how the content is meant to be handled.
- `content_type`: MIME type of the uploaded file (e.g., "image/png" or "application/pdf").
- `name`: Name of the form input field.
- `filename`: Original name of the uploaded file.

If no body data, an empty list is returned.

**Overriding Default Parser:**

By default, `parse_multipart()` uses `webutils::parse_http()` internally. You can override this globally by setting the `AMBIORIX_MULTIPART_FORM_DATA_PARSER` option:

```
options(AMBIORIX_MULTIPART_FORM_DATA_PARSER = my_custom_parser)
```

Your custom parser function must accept the following parameters:

1. `body`: Raw vector containing the form data.
2. `content_type`: The 'Content-Type' header of the request as defined by the client.
3. `...`: Additional optional parameters.

**Value**

Named list.

**See Also**

[parse\\_form\\_urlencoded\(\)](#), [parse\\_json\(\)](#)

**Examples**

```

if (interactive()) {
  library(ambiorix)
  library(htmltools)
  library(readxl)

  page_links <- function() {
    Map(
      f = function(href, label) {
        tags$a(href = href, label)
      },
      c("/", "/about", "/contact"),
      c("Home", "About", "Contact")
    )
  }

  forms <- function() {
    form1 <- tags$form(
      action = "/url-form-encoded",
      method = "POST",
      enctype = "application/x-www-form-urlencoded",
      tags$h4("form-url-encoded:"),
      tags$label(`for` = "first_name", "First Name"),
      tags$input(id = "first_name", name = "first_name", value = "John"),
      tags$label(`for` = "last_name", "Last Name"),
      tags$input(id = "last_name", name = "last_name", value = "Coene"),
      tags$button(type = "submit", "Submit")
    )

    form2 <- tags$form(
      action = "/multipart-form-data",
      method = "POST",
      enctype = "multipart/form-data",
      tags$h4("multipart/form-data:"),
      tags$label(`for` = "email", "Email"),
      tags$input(id = "email", name = "email", value = "john@mail.com"),
      tags$label(`for` = "framework", "Framework"),
      tags$input(id = "framework", name = "framework", value = "ambiorix"),
      tags$label(`for` = "file", "Upload CSV file"),
      tags$input(type = "file", id = "file", name = "file", accept = ".csv"),
      tags$label(`for` = "file2", "Upload xlsx file"),
      tags$input(type = "file", id = "file2", name = "file2", accept = ".xlsx"),
      tags$button(type = "submit", "Submit")
    )

    tagList(form1, form2)
  }

  home_get <- function(req, res) {
    html <- tagList(
      page_links(),
      tags$h3("hello, world!"),

```

```
    forms()
  )

  res$send(html)
}

home_post <- function(req, res) {
  body <- parse_json(req)
  # print(body)

  response <- list(
    code = 200L,
    msg = "hello, world"
  )
  res$json(response)
}

url_form_encoded_post <- function(req, res) {
  body <- parse_form_urlencoded(req)
  # print(body)

  list_items <- lapply(
    X = names(body),
    FUN = function(nm) {
      tags$li(
        nm,
        ":",
        body[[nm]]
      )
    }
  )
  input_vals <- tags$ul(list_items)

  html <- tagList(
    page_links(),
    tags$h3("Request processed"),
    input_vals
  )

  res$send(html)
}

multipart_form_data_post <- function(req, res) {
  body <- parse_multipart(req)

  list_items <- lapply(
    X = names(body),
    FUN = function(nm) {
      field <- body[[nm]]

      # if 'field' is a file, parse it & print on console:
      is_file <- "filename" %in% names(field)
      is_csv <- is_file && identical(field[["content_type"]], "text/csv")
    }
  )
}
```

```

is_xlsx <- is_file &&
  identical(
    field[["content_type"]],
    "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
  )

if (is_file) {
  file_path <- tempfile()
  writeBin(object = field$value, con = file_path)
  on.exit(unlink(x = file_path))
}

if (is_csv) {
  # print(read.csv(file = file_path))
}

if (is_xlsx) {
  # print(readxl::read_xlsx(path = file_path))
}

tags$li(
  nm,
  ":",
  if (is_file) "printed on console" else field
)
}
)
input_vals <- tags$ul(list_items)

html <- tagList(
  page_links(),
  tags$h3("Request processed"),
  input_vals
)

res$send(html)
}

about_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("About Us")
  )
  res$send(html)
}

contact_get <- function(req, res) {
  html <- tagList(
    page_links(),
    tags$h3("Get In Touch!")
  )
  res$send(html)
}

```

```

app <- Ambiorix$new(port = 5000L)

app$
  get("/", home_get)$
  post("/", home_post)$
  get("/about", about_get)$
  get("/contact", contact_get)$
  post("/url-form-encoded", url_form_encoded_post)$
  post("/multipart-form-data", multipart_form_data_post)

app$start()
}

```

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 glue::glue_data. |

**Value**

A response pre-hook.

**Examples**

```

my_prh <- function(self, content, data, ext, ...) {
  data$title <- "Mansion"
  pre_hook(content, data)
}

#' Handler for GET at '/'
#'
#' @details Renders the homepage
#' @export
home_get <- function(req, res) {
  res$pre_render_hook(my_prh)
  res$render(
    file = "page.html",

```

```

    data = list(
        title = "Home"
    )
}

```

---

Request

*Request*


---

### Description

A request.

### Value

A Request object.

### 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.

SERVER\_NAME Name of the server.

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". # @field 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\\$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 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 [parse\\_json\(\)](#).

*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.

## Examples

```
if (interactive()) {  
  library(ambiorix)  
  
  app <- Ambiorix$new()  
  
  app$get("/", function(req, res) {  
    print(req)  
    res$send("Using {ambiorix}!")  
  })  
  
  app$start()  
}
```

---

Response

*Response*

---

### Description

Response class to generate responses sent from the server.

### Value

A Response object.

### 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\\$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\\$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)
```

*Arguments:*

`body` Body of the response.

*Details:* Send a plain HTML response.

**Method** `sendf()`:

*Usage:*

```
Response$sendf(body, ...)
```

*Arguments:*

`body` Body of the response.

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

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

**Method** `text()`:

*Usage:*

```
Response$text(body)
```

*Arguments:*

`body` Body of the response.

*Details:* Send a plain text response.

**Method** `send_file()`:

*Usage:*

```
Response$send_file(file)
```

*Arguments:*

`file` File to send.

*Details:* Send a file.

**Method** `redirect()`:

*Usage:*

`Response$redirect(path)`

*Arguments:*

`path` Path or URL to redirect to.

*Details:* Redirect to a path or URL.

**Method** `render()`:

*Usage:*

`Response$render(file, data = list())`

*Arguments:*

`file` Template file.

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

*Details:* Render a template file.

**Method** `json()`:

*Usage:*

`Response$json(body, ...)`

*Arguments:*

`body` Body of the response.

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

*Details:* Render an object as JSON.

**Method** `csv()`:

*Usage:*

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

*Arguments:*

`data` Data to convert to CSV.

`name` Name of the file.

`...` Additional arguments passed to `readr::format_csv()`.

*Details:* Sends a comma separated value file

**Method** `tsv()`:

*Usage:*

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

*Arguments:*

`data` Data to convert to CSV.

`name` Name of the file.

`...` Additional arguments passed to `readr::format_tsv()`.

*Details:* Sends a tab separated value file

**Method** `htmlwidget()`:

*Usage:*

```
Response$htmlwidget(widget, ...)
```

*Arguments:*

`widget` The widget to use.

... Additional arguments passed to `htmlwidgets::saveWidget()`.

*Details:* Sends an htmlwidget.

**Method** `md()`:

*Usage:*

```
Response$md(file, data = list())
```

*Arguments:*

`file` Template file.

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

*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** header():

*Usage:*

```
Response$header(name, value)
```

*Arguments:*

name String. Name of the header.

value 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** `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 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 Response 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` String. 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.

**Examples**

```
if (interactive()) {
  library(ambiorix)

  app <- Ambiorix$new()

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

  app$start()
}
```

---

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   |

**Value**

An Ambiorix response.

**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()
```

---

|      |                 |
|------|-----------------|
| rojb | <i>R Object</i> |
|------|-----------------|

---

**Description**

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

**Usage**

```
rojb(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 `rojb` one would obtain `x <- "hello"`.

**Value**

Object of class "rojb".

**Examples**

```
rojb(1:10)
```

---

|        |               |
|--------|---------------|
| Router | <i>Router</i> |
|--------|---------------|

---

**Description**

Web server.

**Value**

A Router object.

**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$get("/:id/profile", function(req, res){
  msg <- sprintf("This is the profile of user #%"s", req$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](#).

### Value

A Routing object.

### HTTP methods

See [routing-http-methods](#) for the full argument reference. The routing instance exposes helpers for common HTTP verbs; they are registered when the object is initialised and share the same signature.

- `get()`, `put()`, `patch()`, `delete()`, `post()`, `options()` register a handler for the single corresponding HTTP verb; see [routing-http-methods](#).
- `all()` registers a handler that responds to GET, POST, PUT, DELETE, and PATCH; see [routing-http-methods](#).

### Public fields

`error` Error handler.

`get` Register a route handler for HTTP GET requests. See [routing-http-methods](#).

`put` Register a route handler for HTTP PUT requests. See [routing-http-methods](#).

`patch` Register a route handler for HTTP PATCH requests. See [routing-http-methods](#).

`delete` Register a route handler for HTTP DELETE requests. See [routing-http-methods](#).

- `post` Register a route handler for HTTP POST requests. See [routing-http-methods](#).
- `options` Register a route handler for HTTP OPTIONS requests. See [routing-http-methods](#).
- `all` Register a route handler that responds to every HTTP verb used by Ambiorix. See [routing-http-methods](#).

### Active bindings

- `basepath` Basepath, read-only.
- `websocket` WebSocket handler.

### Methods

#### Public methods:

- [Routing\\$new\(\)](#)
- [Routing\\$param\(\)](#)
- [Routing\\$receive\(\)](#)
- [Routing\\$print\(\)](#)
- [Routing\\$engine\(\)](#)
- [Routing\\$use\(\)](#)
- [Routing\\$get\\_routes\(\)](#)
- [Routing\\$get\\_params\(\)](#)
- [Routing\\$get\\_receivers\(\)](#)
- [Routing\\$get\\_middleware\(\)](#)
- [Routing\\$prepare\(\)](#)
- [Routing\\$clone\(\)](#)

#### Method `new()`:

*Usage:*

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

*Arguments:*

`path` Prefix path.

*Details:* Initialise

#### Method `param()`:

*Usage:*

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

*Arguments:*

`name` Name of the parameter

`handler` Function that accepts the request, response, parameter value and the parameter name.

*Details:* PARAM Method

*Examples:*

```

app <- Ambiorix$new()

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

app$param("person", function(req, res, value, name){
  if(value == "notWanted"){
    res$status <- 403L
    res$send("This is the end.")
  }

  # continue processing the request...
})

app$get("/hi/:person", function(req,res){
  res$sendf("Hi! %s", req$params$person)
})
app$get("/info/:person", function(req,res){
  res$sendf("Here is all your info, %s", req$params$person)
})
if(interactive())
  app$start()

```

**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)
```

*Arguments:*

engine Engine function.

*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(routes = list(), parent = "")
```

*Arguments:*

routes Existing list of routes.

parent Parent path.

*Details:* Get the routes

**Method get\_params():**

*Usage:*

```
Routing$get_params(params = list(), parent = "")
```

*Arguments:*

params Existing list of parameter middlewares.

parent Parent path.

*Details:* Get the parameter middlewares

**Method get\_receivers():**

*Usage:*

```
Routing$get_receivers(receivers = list())
```

*Arguments:*

receivers Existing list of receivers

*Details:* Get the websocket receivers

**Method** get\_middleware():*Usage:*

```
Routing$get_middleware(middlewares = list(), parent = "")
```

*Arguments:*

middlewares Existing list of middlewares

parent Parent path

*Details:* Get the middleware

**Method** prepare():*Usage:*

```
Routing$prepare()
```

*Details:* Prepare routes and decomposes paths

**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.

**See Also**

[routing-http-methods](#)

**Examples**

```
## -----
## Method `Routing$param`
## -----

app <- Ambiorix$new()

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

app$param("person", function(req, res, value, name){
  if(value == "notWanted"){
    res$status <- 403L
    res$send("This is the end.")
  }

  # continue processing the request...
})
```

```

app$get("/hi/:person", function(req,res){
  res$sendf("Hi! %s", req$params$person)
})
app$get("/info/:person", function(req,res){
  res$sendf("Here is all your info, %s", req$params$person)
})
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()

```

---

routing-http-methods    *Routing HTTP Methods*

---

## Description

Register route handlers for HTTP verbs on a [Routing](#) instance.

## Arguments

|         |   |
|---------|---|
| path    | String. Route to listen to; use <code>:</code> to define a parameter (e.g. <code>"/hello/:name"</code> ).   |
| handler | Function that accepts the request and response objects and returns an httpuv response (e.g. <a href="#">response()</a> ). Handlers can return the result of helper functions such as <code>Response\$text()</code> , <code>Response\$json()</code> , or the output of any renderer. |
| error   | Optional handler invoked if the route raises an error; receives the request, response, and the error condition.   |

## Details

The routing helpers provide a fluent API for attaching handlers to HTTP methods. Each helper shares the same signature and behaviour.

### Supported helpers:

- `get()`: Respond to HTTP GET requests.
- `post()`: Respond to HTTP POST requests.
- `put()`: Respond to HTTP PUT requests.
- `patch()`: Respond to HTTP PATCH requests.
- `delete()`: Respond to HTTP DELETE requests.
- `options()`: Respond to HTTP OPTIONS requests.
- `all()`: Respond to every method above.

## Value

The routing object invisibly so calls can be chained.

## See Also

[Routing](#)

## Examples

```
app <- Ambiorix$new()

app$get("/", function(req, res) {
  res$text("Hello, world!")
})

app$post("/echo", function(req, res) {
  res$json(list(received = req$body))
})

app$all("/health", function(req, res) {
  res$json(list(status = "ok"))
})
```

---

serialise

*Serialise an Object to JSON*

---

## Description

Serialise an Object to JSON

## Usage

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

## Arguments

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

## Details

Ambiorix uses `yyjsonr::write_json_str()` by default for serialization.

### Custom Serialiser:

To override the default, set the `AMBIORIX_SERIALISER` option to a function that accepts:

- `data`: Object to serialise.
- `...`: Additional arguments passed to the function.

For example:

```
my_serialiser <- function(data, ...) {  
  jsonlite::toJSON(x = data, ...)  
}  
  
options(AMBIORIX_SERIALISER = my_serialiser)
```

## Value

JSON string.

## Examples

```
if (interactive()) {  
  # a list:  
  response <- list(code = 200L, msg = "hello, world!")  
  
  serialise(response)  
  #> {"code":200,"msg":"hello, world"}  
  
  serialise(response, auto_unbox = FALSE)  
  #> {"code":[200],"msg":["hello, world"]}  
  
  # data.frame:  
  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](#).

**Value**

The log object.

**Examples**

```
# define custom loggers:
info_logger <- log::Logger$new("INFO")
success_logger <- log::Logger$new("SUCCESS")
error_logger <- log::Logger$new("ERROR")

info_logger$log("This is an info message.")
success_logger$log("This is a success message.")
error_logger$log("This is an error message.")

# set custom loggers for Ambiorix:
set_log_info(info_logger)
set_log_success(success_logger)
set_log_error(error_logger)
```

---

|          |             |
|----------|-------------|
| stop_all | <i>Stop</i> |
|----------|-------------|

---

**Description**

Stop all servers.

**Usage**

```
stop_all()
```

**Value**

NULL (invisibly)

**Examples**

```
if (interactive()) {  
  stop_all()  
}
```

---

|              |              |
|--------------|--------------|
| token_create | <i>Token</i> |
|--------------|--------------|

---

**Description**

Create a token

**Usage**

```
token_create(n = 16L)
```

**Arguments**

n                   Number of bytes.

**Value**

Length 1 character vector.

**Examples**

```
token_create()  
token_create(n = 32L)
```

use\_html\_template      *HTML Template*

---

**Description**

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

**Usage**

```
use_html_template()
```

**Value**

A renderer function.

**Examples**

```
use_html_template()
```

---

Websocket      *Websocket*

---

**Description**

Handle websocket messages.

**Value**

A Websocket object.

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

- `Websocket$new()`
- `Websocket$send()`
- `Websocket$print()`
- `Websocket$clone()`

**Method new():**

*Usage:*

```
Websocket$new(ws)
```

*Arguments:*

ws The websocket

*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.

**Examples**

```
# create an Ambiorix app with websocket support:
if (interactive()) {
  library(ambiorix)

  home_get <- function(req, res) {
    res$send("hello, world!")
  }

  greeting_ws_handler <- function(msg, ws) {
    cat("Received message:", "\n")
    print(msg)
    ws$send("greeting", "Hello from the server!")
  }

  app <- Ambiorix$new(port = 8080)
  app$get("/", home_get)
  app$receive("greeting", greeting_ws_handler)
  app$start()
}

# create websocket client from another R session:
if (interactive()) {
  client <- websocket::WebSocket$new("ws://127.0.0.1:8080", autoConnect = FALSE)

  client$onOpen(function(event) {
    cat("Connection opened\n")
  })
}
```

```
msg <- list(
  isAmbiorix = TRUE, # __MUST__ be set!
  name = "greeting",
  message = "Hello from the client!"
)

# serialise:
msg <- yyjsonr::write_json_str(msg, auto_unbox = TRUE)

client$send(msg)
})

client$onMessage(function(event) {
  cat("Received message from server:", event$data, "\n")
})

client$connect()
}
```

---

websocket\_client

*Websocket Client*

---

## Description

Handle ambiorix websocket client.

## Usage

```
copy_websocket_client(path)
```

```
get_websocket_client_path()
```

```
get_websocket_clients()
```

## Arguments

path                    Path to copy the file to.

## Value

- `copy_websocket_client`: String. The new path (invisibly).
- `get_websocket_client_path`: String. The path to the local websocket client.
- `get_websocket_clients`: List. Websocket clients.

**Functions**

- `copy_websocket_client` Copies the websocket client file, useful when ambiorix was not setup with the ambiorix generator.
- `get_websocket_client_path` Retrieves the full path to the local websocket client.
- `get_websocket_clients` Retrieves clients connected to the server.

**Examples**

```
chat_ws <- function(msg, ws) {  
  lapply(  
    X = get_websocket_clients(),  
    FUN = function(c) {  
      c$send("chat", msg)  
    }  
  )  
}
```

# Index

## \* export

- Routing, 44
- Ambiorix, 2, 44
- ambiorix::Routing, 2, 42
- as\_cookie\_parser, 9
- as\_cookie\_preprocessor, 10
- as\_path\_to\_pattern, 10
- as\_renderer, 11
- content, 12
- content\_csv (content), 12
- content\_html (content), 12
- content\_json (content), 12
- content\_plain (content), 12
- content\_protobuf (content), 12
- content\_tsv (content), 12
- copy\_websocket\_client
  - (websocket\_client), 56
- default\_cookie\_parser, 12
- dput(), 42
- forward, 13
- get\_websocket\_client\_path
  - (websocket\_client), 56
- get\_websocket\_clients
  - (websocket\_client), 56
- ggplot2::ggsave(), 37
- htmltools::htmlTemplate(), 54
- htmlwidgets::saveWidget(), 36
- import, 14
- jobj, 15
- log::Logger, 52
- mockRequest, 15
- new\_log, 16
- parse\_form\_urlencoded, 16
- parse\_form\_urlencoded(), 21, 25
- parse\_json, 20
- parse\_json(), 17, 25, 32
- parse\_multipart, 24
- parse\_multipart(), 17, 21
- pre\_hook, 29
- pre\_hook(), 38
- readr::format\_csv(), 35
- readr::format\_tsv(), 35
- Request, 9, 13, 30
- Response, 10, 33
- response (responses), 40
- response(), 4, 49
- response\_404 (responses), 40
- response\_404(), 3
- response\_500 (responses), 40
- response\_500(), 3, 43
- responses, 40
- robject, 42
- Router, 42, 44, 47
- Routing, 44, 49, 50
- routing-http-methods, 44, 45, 48, 49
- serialise, 50
- set\_log, 52
- set\_log\_error (set\_log), 52
- set\_log\_info (set\_log), 52
- set\_log\_success (set\_log), 52
- stop\_all, 52
- token\_create, 53
- use\_html\_template, 54
- Websocket, 54
- websocket\_client, 56
- webutils::parse\_http(), 17, 25

`yyjsonr::read_json_raw()`, [21](#)  
`yyjsonr::write_json_str()`, [51](#)