

Package ‘ambiorix’

July 22, 2025

Title Web Framework Inspired by 'Express.js'

Version 2.2.0

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 7.3.2

Imports fs, log, cli, glue, httpuv, methods, assertthat, webutils (>= 1.2.0), yyjsonr (>= 0.1.20)

Suggests mime, readr, readxl, ggplot2, promises, 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, cre] (ORCID: <<https://orcid.org/0000-0002-6637-4107>>),
Opifex [fnd],
Kennedy Mwavu [ctb] (ORCID: <<https://orcid.org/0009-0006-3157-7234>>)

Maintainer John Coene <jcoenep@gmail.com>

Repository CRAN

Date/Publication 2025-03-15 16:40:02 UTC

Contents

Ambiorix	2
as_cookie_parser	9
as_cookie_preprocessor	10
as_path_to_pattern	10
as_renderer	11

content	12
create_dockerfile	12
default_cookie_parser	13
forward	14
import	15
jobj	15
mockRequest	16
new_log	16
parse_form_urlencoded	17
parse_json	21
parse_multipart	25
pre_hook	29
Request	30
Response	33
responses	42
robject	43
Router	44
Routing	45
serialise	51
set_log	52
stop_all	53
token_create	53
use_html_template	54
Websocket	54
websocket_client	56
Index	58

Ambiorix	<i>Ambiorix</i>
----------	-----------------

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 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 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 = "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 `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	<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()
```

Value

Length 1 character vector.

Examples

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

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

create_dockerfile	<i>Dockerfile</i>
-------------------	-------------------

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", file_path)
```

Arguments

port, host	Port and host to serve the application.
file_path	String. Path to file to write to.

Details

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

Value

NULL (invisibly)

Examples

```
if (interactive()) {  
  create_dockerfile(port = 5000L, host = "0.0.0.0", file_path = tempfile())  
  # create_dockerfile(port = 5000L, host = "0.0.0.0", file_path = "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.

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	<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.

Value

A Request object.

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")
```

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$input(id = "email", name = "email", value = "john@mail.com"),
        tags$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_multipart	<i>Parse multipart form data</i>
-----------------	----------------------------------

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()
}

```

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	<i>Request</i>
---------	----------------

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 Directives 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	<i>Response</i>
----------	-----------------

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$print()`
- `Response$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 named 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 String. Name of the variable.

value Value of the variable.

Details: Set Data

Returns: Invisible returns self.

Method get():

Usage:

Response\$get(name)

Arguments:

name String. Name of the variable to get.

Details: Get data

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 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 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()
```

roby

R Object

Description

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

Usage

```
roby(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 replaced like `x <- hello` (missing quote): breaking the template. Using `roby` one would obtain `x <- "hello"`.

Value

Object of class "roby".

Examples

```
roby(1:10)
```

Router

Router

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.

Public fields

error Error handler.

Active bindings

basepath Basepath, read-only.

websocket WebSocket 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$get_middleware()`
- `Routing$prepare()`
- `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)

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_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.

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 an Object to JSON

Description

Serialise an Object to JSON

Usage

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

Arguments

data	Data to serialise.
...	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

Customise logs

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	<i>HTML Template</i>
-------------------	----------------------

Description

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

Usage

```
use_html_template()
```

Value

A renderer function.

Examples

```
use_html_template()
```

Websocket	<i>Websocket</i>
-----------	------------------

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

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, [45](#)
- Ambiorix, [2](#), [45](#)
- ambiorix::Routing, [2](#), [44](#)
- 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](#)
- create_dockerfile, [12](#)
- default_cookie_parser, [13](#)
- dput(), [43](#)
- forward, [14](#)
- get_websocket_client_path
 - (websocket_client), [56](#)
- get_websocket_clients
 - (websocket_client), [56](#)
- ggplot2::ggsave(), [37](#)
- htmltools::htmlTemplate(), [54](#)
- htmlwidgets::saveWidget(), [36](#)
- import, [15](#)
- jobj, [15](#)
- log::Logger, [52](#)
- mockRequest, [16](#)
- new_log, [16](#)
- parse_form_urlencoded, [17](#)
- parse_form_urlencoded(), [22](#), [26](#)
- parse_json, [21](#)
- parse_json(), [18](#), [26](#), [32](#)
- parse_multipart, [25](#)
- parse_multipart(), [18](#), [22](#)
- pre_hook, [29](#)
- pre_hook(), [40](#)
- readr::format_csv(), [36](#)
- readr::format_tsv(), [36](#)
- Request, [9](#), [13](#), [30](#)
- Response, [10](#), [33](#)
- response (responses), [42](#)
- response(), [4](#), [46–48](#)
- response_404 (responses), [42](#)
- response_404(), [3](#)
- response_500 (responses), [42](#)
- response_500(), [3](#), [44](#)
- responses, [42](#)
- robjects, [43](#)
- Router, [44](#), [45](#), [49](#)
- Routing, [45](#)
- serialise, [51](#)
- set_log, [52](#)
- set_log_error (set_log), [52](#)
- set_log_info (set_log), [52](#)
- set_log_success (set_log), [52](#)
- stop_all, [53](#)
- token_create, [53](#)
- use_html_template, [54](#)
- Websocket, [54](#)
- websocket_client, [56](#)
- webutils::parse_http(), [17](#), [26](#)

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