Package 'FastRWeb'

September 9, 2025

Version 1.2-3

Title Fast Interactive Framework for Web Scripting Using R

Author Simon Urbanek [aut, cre, cph] (https://urbanek.org, <https://orcid.org/0000-0003-2297-1732>), Jeffrey Horner [aut]

Maintainer Simon Urbanek <Simon.Urbanek@r-project.org>

Depends R (>= 2.0.0)

Imports base64enc, grDevices, stats, utils, Cairo

Suggests Rserve

Description Infrastrcture for creating rich, dynamic web content using R scripts while maintaining very fast response time.

License GPL-2

URL https://www.rforge.net/FastRWeb/

BugReports https://github.com/s-u/FastRWeb/issues

NeedsCompilation yes

Contents

add.header 2
done 3
FastRWeb 3
oinput 5
out 7
parse.multipart 8
requests 9
WebPlot 9
WebResult 11

Index **13**

`add.header` *Add HTML headers to FastRWeb response.*

Description

`add.header` appends additional headers to the HTML reponse when using `WebResult` with any other command than `"raw"`.

This is useful for handling of cookies (see `getCookies()` in the sample `common.R` script), cache-behavior, implementing URL redirection etc.

Usage

```
add.header(txt)
```

Arguments

`txt`	character vector of header entires. The string may NOT include any CR/LF characters, those will be automatically generated when the final response is constructed. Elements of the vector should represent lines. It is user's responsibility to ensure the entries are valid according to the HTTP standard. Also note that you should never add either `Content-type:` or `Content-length:` headers as those are always generated automatically from the `WebResult`.

Value

Character vector of the resulting headers

Author(s)

Simon Urbanek

See Also

`WebResult`

Examples

```
## main.css.R: serve a static file (main.css) with cache control
run <- function(...) {
  # last for at most an hour
  add.header("Cache-Control: max-age=3600")
  WebResult("file", "main.css", "text/css")
}
```

done *FastRWeb interface functions*

Description

done collects the entire contents created so far using output functions (such as out) and creates a WebResult object that can be returned from the run function

Usage

```
done(..., cmd = "html", type = "text/html; charset=utf-8")
```

Arguments

...	additional entries to appent to the output (or the actual content depending on the command)
cmd	interface command
type	content type

Details

Some objects can override the default command and type value. For example, if the document consits solely of a plot then the content type is taken from the plot and command changed to "tmpfile".

Value

Returns a WebResult object that can be returned from the run function.

See Also

out, WebResult

FastRWeb *FastRWeb - infratructure to serve web pages with R scripts efficiently*

Description

FastRWeb is not just a package, but an entire infrastructure allowing the use of R scripts to create web pages and graphics.

The basic idea is that an URL of the form http://server/cgi-bin/R/foo?bar=value will be processed by FastRWeb such as to result in sourcing of the foo.R script and running the function run(bar="value") which is expected to be defined in that script. The results of a script can be anything from HTML pages to bitmap graphics or PDF document.

FastRWeb uses CGI or PHP as front-end and Rserve server as the back-end. For details see *Urbanek, S. (2008)* FastRWeb: Fast Interactive Web Framework for Data Mining Using R, *IASC 2008*.

The R code in the package itself provides R-side tools that facilitate the delivery of results to a browser - such as WebResult, WebPlot, out, done - more in detail below.

Installation

The default configuration of FastRWeb assumes that the project root will be in `/var/FastRWeb` and that the server is a unix machine. It is possible to install FastRWeb in other settings, but it will require modification of the configuration.

First, the `FastRWeb` package should be installed (typically using `install.packages("FastRWeb")` in R). The installed package contains shell script that will setup the environment in `/var/FastRWeb`. To run the script, use

```
system(paste("cd",system.file(package="FastRWeb"),"&& install.sh"))
```

For the anatomy of the `/var/FastRWeb` project root see below.

Once created, you can inspect the Rserve configuration file `/var/FastRWeb/code/rserve.conf` and adjust it for your needs if necessary. You can also look a the Rserve initialization script located in `/var/FastRWeb/code/rserve.R` which is used to pre-load data, packages etc. into Rserve. If you are happy with it, you can start Rserve using `/var/FastRWeb/code/start`

In order to tell your webserver to use FastRWeb, you have two options: CGI script or PHP script. The former is more common as it works with any web server. The FastRWeb R package builds and installs the Rcgi script as part of its installation process into the `cgi-bin` directory of the package, but it has no way of knowing about the location of your server's `cgi-bin` directory, so it is left to the user to copy the script in the proper location. Use `system.file("cgi-bin", package="FastRWeb")` in R to locate the package directory - it will contain an executable `Rcgi` (or `Rcgi.exe` on Windows) and copy that executable into you server's `cgi-bin` directory (on Debian/Ubuntu this is typically `/usr/lib/cgi-bin`, on Mac OS X it is `/Library/WebServer/CGI-Execu` Most examples in FastRWeb assume that you have renamed the script to `R` instead of `Rcgi`, but you can choose any name.

With Rserve started and the CGI script in place, you should be able to open a browser and run your first script, the URL will probably look something like `http://my.server/cgi-bin/R/main`. This will invoke the script `/var/FastRWeb/web.R/main.R` by sourcing it and running the `run()` function.

For advanced topics, please see `Rserve` documentation. For production systems we encourage the use of `gid`, `uid`, `sockmod` and `umask` configuration directives to secure the access to the Rserve according to your web server configuration.

Project root anatomy

The project root (typically `var/FastRWeb`) contains various directories:

- `web.R` - this directory contains the R scripts that will be served by FastRWeb. The URL is parsed such that the path part after the CGI binary is taken, `.R` appended and serves to locate the file in the `web.R` directory. Once located, it is sourced and the `run()` function is called with query strang parsed into its arguments. The default installation also sources `common.R` in addition to the specified script (see `code/rserve.R` and the `init()` function for details on how this is achieved - you can modify the behavior as you please).
- `web` - this directory can contain static content that can be referenced using the `"file"` command in `WebResult`.
- `code` - this directory contains supporting infrastructure and configurations files in association with the Rserve back-end. If the `start` script in this directory is used, it loads the `rserve.conf` configuration file and sources `rserve.R` as initialization of the Rserve master. The `init()` function (if present, e.g., defined in `rserve.R`) is run on every request.
- `tmp` - this directory is used for temporary files. It should be purged occasionally to prevent accumulation of temporary files. FastRWeb provides ways of cleanup (e.g., see `"tmpfile"`

command in WebResult), but crashed or aborted requests may still leave temporary files around. Onyl files from this directory can be served using the "tmpfile" WebResult command.

- logs - this directory is optional and if present, the Rcgi script will log requests in the cgi.log file in this directory. It records the request time, duration, IP address, WebResult command, payload, optional cookie filter and the user-agent. If you want to enable logging, simply create the logs directory with sufficient permissions to allow the Rcgi script to write in it.
- run - this directory is optional as well and used for run-time systems such as global login authorization etc. It is not populated or used in the CRAN version of FastRWeb, but we encourage this structure for any user-defined subsystems.

In addition, the default configuration uses a local socket of the name socket to communicate with the Rserve instance. Note that you can use regular unix permissions to limit the access to Rserve this way.

See Also

WebResult, WebPlot, out, done, add.header

---

oinput *Functions aiding in creating HTML form elements.*

Description

oinput creates an input element (text input, button, checkbox, file, hidden value, image, password, radio button or reset/submit button)

oselection creates a drop-down list of items

osubmit is a convenience wrapper for oinput(type='submit', ...) to create a submit button

Usage

```
oinput(name, value, size, type="text", checked=FALSE, ...)
osubmit(name="submit", ...)
oselection(name, text, values = text, sel.index, sel.value, size, ...)
```

Arguments

name	name of the element in the HTML form. This argument is mandatory and should be unique in the form.
value	optional, value that will be pre-populated in the text field and/or the caption of the button.
size	optional, size of the element. For text input the number of visible characters, for selection the number of visible items.
type	type of the element. Valid entries are "text", "password", "button", "checkbox", "radio", "file", "hidden", "image", "reset" and "submit".

checked	boolean, if set to TRUE then the checked attribute is set in the element (valid for checkboxes only).
text	character vector of the items that will be shown to the user.
values	values that will represent the text items in the form and thus submitted. Typically IDs are used here instead of the actual text to avoid issues with encoding and size.
sel.index	index (integer or a logical vector) specifying which value will be selected. If missing, none will be marked as selected.
sel.value	value (one of the values elements) which will be selected. Only one of sel.index and set.value may be specified.
...	Additional HTML attributes and their values. The actual range of supported attibutes is browser- and element-specific. Some commonly supported attributes include disabled (must be boolean), class, id, style, onChange, onClick, onSelect, onFocus, onBlur. It is possible to pass objects as long as they implement as.character method to generate valid values that can be used in the item="value" form, i.e. assuming double quotes around the value in HTML.

Value

The functions are called for their side-effect (see out). They return the current HTML buffer.

Note

All form-level functions assume the existence of an enclosing form. The actual behavior (other than custom JavaScript callback attributes) is defined by the enclosing form.

Author(s)

Simon Urbanek

See Also

out, oprint, done

Examples

```
run <- function(foo, fruit, ...) {
  fruits <- c("apples", "oranges", "pears")
  if (!missing(fruit))
    out("Thank you for choosing ", fruits[as.integer(fruit)],"!<p>")

  out("<form>")
  out("Foo:")
  oinput("foo", foo)
  out("<br>Select fruit:")
  oselection("fruit", fruits, seq.int(fruits), , fruit)
  out("<br>")
  osubmit()
  out("</form>")
  done()
}
```

out *FastRWeb HTML output functions*

Description

`out` outputs the argument as-is (also works for objects that are intended for web output)

`oprint` outputs the result of verbatim `print` call

`otable` constructs a table

`ohead` creates a header

`oclear` clears (by discarding existing content) the output buffer and/or headers

Usage

```
out(..., sep = "", eol = "\n")
oprint(..., sep = "\n", escape = TRUE)
otable(..., tab = "", tr = "", cs = "</td><td>", escape = TRUE)
ohead(..., level = 3, escape = TRUE)
oclear(output=TRUE, headers=FALSE)
```

Arguments

`...`	entries to output or print
`sep`	separator string
`eol`	end of line separator
`escape`	if `TRUE` special HTML characters are escaped in inner text (via 'FastRWeb:::htmlEscape'), if `FALSE` the strings are passed without modification. It can also be a function taking exactly one argument that is expected to perform the escaping.
`tab`	additional attributes for `table` HTML tag
`tr`	additional attibutes for table row (`tr`) HTML tag
`cs`	column separator
`level`	level of the header (1 is the topmost)
`output`	logical, if `TRUE` then the output is cleared
`headers`	logical, if `TRUE` then the headers are cleared

Details

The output functions enable the `run` function to build the result object gradually as opposed to returing just one `WebResult` object at the end.

The output functions above manipulate an internal buffer that collects output and uses `done` to contruct the final `WebResult` object. It is analogous to using `print` to create output in R scripts as they proceed. However, due to the fact that `print` output is generally unsuitable as HTML output, the output function here process the output such that the result is a HTML document. Special HTML characters '<', '>' and '&' are escaped in the inner text (not in tags) if `escape=TRUE` in functions that provide that argument.

NOTE: It is important to remember that the output is collected in a buffer, so in order to actually create the output, do not forget to use `return(done())` when leaving the `run` function to use that content!

Value

All functions returns the full document as constructed so far

See Also

done, WebResult

Examples

```
run <- function(...) {
  ohead("My Table", level=2)
  d <- data.frame(a = 1:3, b = c("foo", "bar", "foobar"))
  otable(d)
  out("<p><b>Verbatim R output:</b><br>")
  oprint(str(d))
  done()
}
```

---

parse.multipart *Parsing of POST request multi-part body.*

Description

parse.multipart parses the result of a POST request that is in a multi-part encoding. This is typically the case when a form is submitted with "enctype='multipart/form-data'" property and "file" input types.

Usage

```
parse.multipart(request = .GlobalEnv$request)
```

Arguments

request	Request interface object as defined by the FastRWeb interface. parse.multipart will use c.type, c.length and body elements of the object.

Value

On success a named list of values in the form. Scalar values are passed literally as strings, files (multi-part chunks) are passed as lists with named elements content_type, tempfile (file containing the content), filename (name of the file as specified in the encoding,if present) and head (character vector of content headers).

On failure NULL with a warning.

Note

The typical use is along the lines of:

```
if (grepl("^multipart", request$s.type)) pars <- parse.multipart()
```

The function uses warnings to communicate parsing issues. While debugging, it may be usedful to convert then to errors via options(warn=2) so they will be visigle on the client side.

Author(s)

The original parser code was written by Jeffrey Horner for the Rook package.

`requests` *FastRWeb asynchronous (AJAX) requests*

Description

`arequests` creates an anchor object representing AJAX request to load elements of the document dynamically

Usage

```
arequest(txt, target, where, ..., attr = "")
```

Arguments

`txt`	text (or any HTML content) inside the anchor
`target`	URI to load
`where`	name of the element (usuall a `div` tag) load the new content into
`...`	additional parameters to the request
`attr`	additional attributes for the anchor

Value

Returns an object that can be added to the HTML document.

`WebPlot` *Graphics device for inclusion of plots in FastRWeb results.*

Description

`WebPlot` opens a new graphics device (currently based on `Cairo`) and returns an object that can be used as a result of FastRWeb functions or in web output.

Usage

```
WebPlot(width = 640, height = 480, type = "png", ...)
```

Arguments

`width`	width of the resulting plot (normally in pixels)
`height`	height of the resulting plot (normally in pixels)
`type`	type of the output
`...`	further arguments to be passed to `Cairo`

Details

WebPlot generates a temporary file name that is accessible using the "tmpfile" command of WebResult and opens a new Cairo device with the specified parameteres. It returns a WebPlot object that can be either retured directly from the run() function (and thus resulting in one image) or used with the out() function to reference the image in an HTML page (see examples below).

Note that as.WebResult coercion is used to finalize the result when returned directly and it will close the device, so dev.off() is optional and not needed in that case. Also WebPlot reserves the right to close any or all other active WebPlot devices - this ensures that dev.off() may not be neeed at all even when using multiple WebPlots.

Value

WebPlot object.

The structure of the WebPlot class is considered internal and should not be created directly. Current attributes include file (filename), type (output type), mime (MIME type), width, height.

Author(s)

Simon Urbanek

See Also

WebResult

Examples

```
## example 1: single image
## if saved as "plot.png.R"
## it can be served as http://server/cgi-bin/R/plot.png
run <- function(n = 100, ...) {
  n <- as.integer(n)
  # create the WebPlot device
  p <- WebPlot(800, 600)
  # plot ...
  plot(rnorm(n), rnorm(n), pch=19, col="#ff000080")
  # return the WebPlot result
  p
}

## example 2: page containing multiple images
## if saved as "plotex.html.R"
## it can be served as http://server/cgi-bin/R/plotex.html
run <- function(...) {
  out("<h2>Simple example<h2>")
  data(iris) ## ideally, you'll use data from the Rserve session
  attach(iris)
  p <- WebPlot(600, 600)
  plot(Sepal.Length, Petal.Length, pch=19, col=Species)
  out(p)
  p <- WebPlot(350, 600)
  barplot(table(Species), col=seq.int(levels(Species)))
  out(p)
  done()
}
```

`WebResult` *Result object of a FastRWeb script*

Description

`WebResult` is the class of the object that will be returned from the `run` function of a FastRWeb script back to the browser.

Using a separate class allows automatic conversion of other objects into the necessary representation - all that is needed is a `as.WebResult` method for that particular object.

`WebResult` function can be used to create such objects directly.

`as.WebResult` coerces an object into a `WebResult`, it is a generic. This allows methods to be defined for `as.WebResult` which act as convertors transforming R objects into web results.

Usage

```
WebResult(cmd = "html", payload = "", content.type = "text/html; charset=utf-8",
          headers = character(0))
as.WebResult(x, ...)
```

Arguments

`cmd`	string, command passed back to the FastRWeb interface. Currently supported commands are `"html"`, `"file"`, `"tmpfile"` and `"raw"`. See details below.
`payload`	string, the body (contents) that will be sent back or file name, depending on the command
`content.type`	MIME content type specification as it will be returned to the browser
`headers`	string vector, optional additional headers to be sent to the browser. Must not contain CR or LF!
`x`	object to convert into `WebResult`
`...`	additional arguments passed to the method

Details

There are four ways the results can be passed from R to the client (browser):

- `"html"` is the default mode and it simply sends the result contained in `payload` to the browser as the body of the HTTP response.
- `"file"` sends the content of the file with the name specified in `payload` from the `web` subdirectory of the FastRWeb project root as the body of the HTTP response.
- `"tmpfile"` sends the content of the file with the name specified in `payload` from the `tmp` subdirectory of the FastRWeb project root as the body of the HTTP response and removes the file once it was delivered.
- `"raw"` does not generate any HTTP headers but assumes that `payload` defines the entire HTTP response including headers. The use of this command is discouraged in favor of `"html"` with headers, since the payload must be properly formatted, which can be difficult.

All modes except `"raw"` cause FastRWeb to generate HTTP headers based on the content and any custom headers that were added using `add.header` or the `headers` argument. Note that the latter two may NOT contain `Content-length:` and `Content-type:` entries as those are generated automatically based on the content and the `content.type` argument.

Value

Object of the class `WebResult`

Author(s)

Simon Urbanek

See Also

`add.header`, `done`

Index

∗ **interface**
    add.header, 2
    done, 3
    FastRWeb, 3
    oinput, 5
    out, 7
    requests, 9
    WebPlot, 9
    WebResult, 11
∗ **utilities**
    parse.multipart, 8

add.header, 2, *5*, *11*, *12*
arequest *(requests)*, 9
as.WebResult, *10*
as.WebResult *(WebResult)*, 11

Cairo, *9*, *10*

done, *3*, 3, *5–8*, *12*

FastRWeb, 3
FastRWeb-package *(FastRWeb)*, 3

oclear *(out)*, 7
ohead *(out)*, 7
oinput, 5
oprint, *6*
oprint *(out)*, 7
oselection *(oinput)*, 5
osubmit *(oinput)*, 5
otable *(out)*, 7
out, *3*, *5*, *6*, 7, *10*

parse.multipart, 8

requests, 9

WebPlot, *3*, *5*, 9
WebResult, *2–5*, *7*, *8*, *10*, 11