Package: webserver (via r-universe)

November 20, 2026

Version 1.0-1

Title Web/HTTP Server for R

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

Description Provides a highly performant and flexible webserver for R. Supports HTTP and TLS/SSL.

License GPL-2 | GPL-3

SystemRequirements OpenSSL (>= 1.1.0, optional, for TLS)

Imports jsonlite

Suggests FastRWeb

URL http://www.rforge.net/webserver/

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

Encoding UTF-8

NeedsCompilation yes

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

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-11-20 08:38:07 UTC

RemoteUrl https://github.com/cran/webserver

RemoteRef HEAD

RemoteSha 00094ba5433017430cf3173773be8f3b81dc971e

Contents

handlers 2
R.run 5
simple 6

Index **9**

handlers *Webserver Content Handlers*

Description

addHandler adds a content handler to be used with the webserver.

rmAllHandlers removes all currently registered handlers.

defaultBodyParsers returns the request body parsers used by R function handlers with parse.body=TRUE.

Usage

```
addHandler(type, prefix, ...)
rmAllHandlers()
defaultBodyParsers()
```

Arguments

type	handler type
prefix	string, prefix at which the handler will be registered.
...	parameters for the handler, see details below.

Details

For each webserver request the content handlers are matched in sequence against the request's path and each matching handler gets a chance to provide the result for the request.

Most commonly used handlers are "static" which serves files from the file system, "rfun" which registers an R function to be called with the request details and "rewrite" which modifies the request path according to specified rules.

Value

undefined (currently returns TRUE invisibly, but that is not guaranteed for the future).

Static Handler

Syntax: addHandler("static", prefix, path, index=NULL, stop=FALSE, gzip=FALSE)

path must be an existing directory. It is expanded (see path.expand) and converted to an absolute path when the handler is added (on unix symbolic links are resolved), so later changes of the working directory have no effect.

When the prefix is matched, it is removed from the request path and the remaining path is appended to path to look up files. If the target matches a directory and index is a string, then index is appended to the path, unless the trailing slash is missing in which case a re-direct is returned to add the slash (this makes sure that relative paths in the index stay valid). If the resulting target is a file, it is served as the request result. Otherwise either 404 Not Found response is sent (if stop=TRUE) or the processing is passed to the next handler.

The static handler supports conditional GET requests based on last modified timestamp. It also supports HTTP range requests. If `gzip=TRUE` and the client has requested gzip compression then the handler checks if a file `<target>.gz` exists and is not older than `<target>` then the compressed file is served instead. This saves bandwidth for compressible static content.

Example: `addHandler("static", "/", path=".", index="index.html", stop=TRUE)`

Serves files from the current directory (at the time of the call) at the root of URLs. It will stop and not evaluate any further handlers even if the file is not found.

R Function Handler

This handler calls an R function with the same API as the internal R server and Rserve.

`addHandler("rfun", prefix, FUN, parse.query=TRUE, parse.headers=TRUE, parse.body=FALSE, fork=FALSE)`

FUN function to call for a request (see below for details)

parse.query logical, if `TRUE` then the query string is parsed into a named character vector, otherwise it is just one string (see below)

parse.headers logical, if `TRUE` then the headers are parsed (see below), otherwise just a raw vector

parse.body `FALSE` to pass the request body as-is, `TRUE` to decode it with the parsers returned by `defaultBodyParsers()` or a named list of parsers (see below)

fork logical, if `TRUE` then the R processing will be delegated to forked workers, one per each I/O thread (not available on Windows). Otherwise the R code is evaluated in the current session.

The function `FUN` should have the following signature:

`function(path, query, body, headers, ...)`

path string, path of the request

query query string part of the request. If `parse.query=TRUE` (the default) then the query string is parsed into a named character vector, so, e.g., the query string `"a=1&b=foo"` will be parsed into the R object `c(a="1", b="foo")`. Otherwise it is just the string itself.

body body part of the request or `NULL` if there is no body. Unless it is decoded by a parser (see `parse.body` below), it is a raw vector with the `"content-type"` attribute (if specified by the client).

headers request headers. It is a raw vector unless `parse.headers=TRUE` in which case the headers are parsed into a named list where the header names are lowercase and the `"-"` character is replaced by `"."` so, for example, `User-Agent` is stored in the `user.agent` element of the list. Note that the parsing is experimental, since the interpretation can vary among header entries.

The server adds the following headers which are always present unless stated otherwise: `Request-Method` (`request.method` when parsed) is the request method name (`GET`, `POST`, etc.), `Request-Proto` (`request.proto`) is `http` or `https`, depending on the listener the request was received on, and `Request-Client-CN` (`request.client.cn`) is the common name of the client certificate. The latter is only present if the client has presented a valid certificate, i.e., on HTTPS listeners with `tls.client.ca` set (see `webstart`), and the certificate has a common name that can be passed in a header (without control characters and not excessively long), so handlers can use it to identify the client. Header names starting with `Request-` are reserved for the server, so such headers sent by the client are dropped. Behind a reverse proxy,

these describe the connection from the proxy, the proxy may pass the original protocol on in another header such as `X-Forwarded-Proto`.

... reserved for future extensions

The result is expected to have the following form:

`list(result, content.type, headers, http.code)`

Only the first list element is mandatory, all others are optional.

result Can be either a string or a raw vector containing the body of the result to return to the client. As a special case, the result can be a named character vector of length one with either `file` or `tempfile` name in which case the result is interpreted as a path to a file whose contents will be served. In the case of `tempfile` the file will be removed once read.

content.type string, MIME type of the result, defaults to `"text/html"`

headers character vector (unnamed) of any additional headers to send. Note that it may not contain `Content-Type` nor `Content-Length`.

http.code integer, valid HTTP code (100..999), defaults to 200L

One exception is that returning `FALSE` indicates that the handler has refused to process the request and so the search should continue to the next handler. Note that due to the relatively expensive nature of some evaluation methods (such as worker communication) it is recommended to consolidate R handlers instead of relying on each to signal its unwillingness to process the request.

If the returned object has any other form (e.g. if it is a character vector), it is assumed to be an error and it will result in 500 HTTP error code.

Example: `addHandler("rfun", "/api/v1/add", function(path, query, ...) list(as.numeric(query["a"]) + as.numeric(query["b"])))`

Body parsers. A parser is a function `function(body, content.type)` which receives the request body (raw vector) and the value of the `Content-Type` header (including parameters such as `charset`) and returns the decoded body which is then passed to `FUN`. The names of the list of parsers are media types (matched case-insensitively, parameters are ignored). A name can also start with `*` to match any media type ending with the rest (e.g., `"*+json"`) or end with `*` to match any media type starting with the rest (e.g., `"text/*"`). An exact match takes precedence, otherwise the first matching name is used. If no parser matches, the body is passed as-is. If the parser fails, the request is answered with 400 Bad Request and `FUN` is not called.

`defaultBodyParsers()` returns a parser for URL encoded forms (`application/x-www-form-urlencoded`) which produces a named character vector like the parsed query and parsers for JSON (`application/json` and `*+json`) using `jsonlite::fromJSON()`. The list can be modified or extended to define the parsers of a handler, e.g.,

`addHandler("rfun", "/api", FUN, parse.body=c(defaultBodyParsers(), list("text/csv"=function(body, content.type) read.csv(text=rawToChar(body)))))`

Rewrite Handler

This handler modifies the request based on simple substitution rules.

`addHandler("rewrite", prefix, replacement, stop=FALSE)`

The `prefix` can contain the `?` character as a wildcard which will match any number of characters other than `/`. The matched content is then available in the form of `$n` variables in the `replacement`

where n is one of the integers 1..9. If the request prefix is matched then it is replaced by the replacement.

Example: addHandler("rewrite", "/api/?/?", "/r?name=$1&par=$2", stop=TRUE)

Will match request /api/copy/this?a=1 and rewrite it to /r?a=1&name=copy&par=this. Note that if the replacement contains a query string, it will be appended to the request query string.

The stop parameter only applies to subsequent rewrite requests, because the rewrite handler is not an actual content handler, it will still proceed to other content handlers regardless of the value of stop, but setting stop=TRUE will make sure that no more rewrite handlers will be consulted.

Author(s)

Simon Urbanek

R.run *Run R*

Description

R.run evaluates expr in a new R process.

Usage

```
R.run(expr, out, where=parent.frame(), bquote=TRUE, redirect=FALSE)
```

Arguments

expr	expression to evaluate
out	string, filename to re-direct stdout/stderr to or NULL if not re-direction is desired.
where	environment to evaluate dot-quoted expressions in
bquote	logical, if TRUE then expr is treated like bquote() which means only expressions inside .() are evaluated locally and everyting else remotely. Otherwise expr is passed as-is.
redirect	logical, if TRUE then the original stdout/stderr are closed such that the output only goes to out. If out is NULL then the output is discarded.

Author(s)

Simon Urbanek

simple *Web Server*

Description

`simple` starts a simple webserver (HTTP) serving static files and (optionally) using an R function as a handler.

`webstart` starts a webserver based on currently registered content handlers, listening on one or more ports or sockets using HTTP or HTTPS.

`hasTLS` returns `TRUE` if the package was built with TLS support, i.e., HTTPS can be used.

Usage

```
simple(listen = 8080L, path = ".", prefix = "/", index = "index.html",
       start = TRUE, Rfun = get0(".http.request"), Rfun.prefix = "/",
       parse.body = TRUE, ...)

webstart(listen = 8080L, threads = 4L, fork = FALSE,
         tls.cert = NULL, tls.key = tls.cert, tls.ca = NULL,
         tls.client.ca = NULL, tls.client.cn = NULL)

hasTLS()
```

Arguments

`listen`	where to listen for connections: a port number or a character vector, each element specifying one listener (see Details).
`path`	string, path to serve static content from (must exist and be a directory) or `NULL` if no static content should be served.
`prefix`	string, URL path prefix to match for the static handler.
`index`	string, name of the file to look for if the URL path matches a directory or `NULL`.
`start`	logical, if `TRUE` then the server is started immediately and all handlers are removed upon exit.
`Rfun`	function to call as R handler or `NULL` if no R handler is desired.
`Rfun.prefix`	string, URL path prefix to match for the R handler. If both prefixes match, the static handler is evaluated first and will proceed to the R handler if no static file was found.
`parse.body`	passed to `addHandler` for the R handler: `TRUE` (default) decodes URL encoded forms and JSON request bodies (see `defaultBodyParsers`), `FALSE` passes the body as a raw vector.
`threads`	integer, number of threads for the server to use.

fork	logical, if TRUE then a new process is forked for each connection. Mutually exclusive with threads > 0, so threads defaults to 0 in that case (with a warning if threads > 0 is specified explicitly). Not supported on Windows. Forked children inherit the RNG state of the parent, so remove .Random.seed from the global environment before starting the server if independent random streams are needed.
tls.cert	file name of the server certificate (PEM format), required for HTTPS. The file may also contain intermediate certificates following the server certificate, they are sent to the clients along with it (making tls.ca below unnecessary).
tls.key	file name of the private key of the server certificate (PEM format). Defaults to tls.cert (also for NA values), i.e., the key can be stored in the same file as the certificate. Encrypted keys are not supported.
tls.ca	optional file name of further certificates (PEM format) sent to the clients along with the server certificate, typically the intermediate certificate(s) of the CA that issued it, and only needed if not already included in tls.cert. They are sent after those in tls.cert, so the intermediate certificates can also be split between the two files.
tls.client.ca	optional file name of CA certificates (PEM format). If specified, clients must present a valid certificate issued by one of these CAs, otherwise the connection is rejected.
tls.client.cn	optional character vector of allowed common names of client certificates (or a list of such vectors, see Details). A name can start with * to match any common name ending with the rest (e.g., "*.example.com") or end with * to match any common name starting with the rest. Requires tls.client.ca.
...	additional parameters passed to webstart

Details

Each element of listen is one of

<port> a port number, the same as "http://localhost:<port>"

"http://<host>:<port>" HTTP on the given port

"https://<host>:<port>" HTTPS on the given port

"unix:<path>" HTTP on a local (unix) socket, not supported on Windows. An existing socket at <path> is replaced.

where <host> is localhost to only accept connections from the local machine, * to accept connections on all IPv4 interfaces or a numeric IPv4 address of the machine (such as 127.0.0.1, 0.0.0.0 (all IPv4 interfaces) or the address of one of its interfaces) to only accept connections on that address. IPv6 addresses are enclosed in brackets (such as [::1]) and require IPv6 support enabled at build time. IPv6 listeners only accept IPv6 connections, so c("https://*:8443", "https://[::]:8443") accepts connections on all IPv4 and IPv6 interfaces. Host names other than localhost are not supported. If the port is omitted, it is 80 for HTTP and 443 for HTTPS. Port 0 means any free port, which is reported when the server is started (e.g., Listening on http://localhost:54321).

The tls.* arguments apply to the HTTPS listeners and are recycled over them: each argument is either a single value used by all HTTPS listeners or has one value for each HTTPS listener (in the

order they appear in `listen`, HTTP listeners are not counted) where `NA` means not set. Since a single value of `tls.client.cn` is already a character vector (of allowed names), different values per listener are specified as a list of such vectors, with `NULL` for listeners that don't check names, e.g., `tls.client.cn = list(NULL, c("alice", "*.example.com"))`.

HTTPS requires TLS 1.2 or higher. All listeners are served by the same handlers. The certificates and keys are read when the server is started, so the server has to be restarted to use renewed certificates.

`webstart()` runs the server in the current R session and returns once it is stopped by a user interrupt (SIGINT, e.g., Ctrl-C) or, on unix, by SIGTERM or SIGHUP. The registered handlers are kept, so the server can be started again with another `webstart()` call.

Value

`simple`: `TRUE`, invisibly.

`webstart`: `NULL` once the server is stopped.

`hasTLS`: logical, `TRUE` if TLS (HTTPS) is supported.

Author(s)

Simon Urbanek

Examples

```
## Not run:
## webstart() runs the server until it is interrupted (e.g., by Ctrl-C
## or the Stop button), so each call below is a separate alternative

## local HTTP server on port 8080
webstart()

## HTTP for local clients and HTTPS for everyone
webstart(c(8080, "https://*:8443"), tls.cert="server.crt", tls.key="server.key")

## two HTTPS listeners with different certificates, the second
## one requires client certificates issued by the corporate CA
webstart(c("https://*:443", "https://*:8443"),
         tls.cert = c("public.pem", "internal.pem"),
         tls.client.ca = c(NA, "corp-ca.pem"),
         tls.client.cn = list(NULL, "*.corp.example.com"))

## End(Not run)
```

Index

∗ **IO**
    handlers, 2
    simple, 6
∗ **utilities**
    R.run, 5

addHandler, *6*
addHandler *(*handlers*)*, 2

defaultBodyParsers, *6*
defaultBodyParsers *(*handlers*)*, 2

handlers, 2
hasTLS *(*simple*)*, 6

path.expand, *2*

R.run, 5
rmAllHandlers *(*handlers*)*, 2

simple, 6

webstart, *3*
webstart *(*simple*)*, 6