

# Package ‘webdeveloper’

August 8, 2022

**Type** Package

**Title** Functions for Web Development

**Version** 1.0.3

**Author** Timothy Conwell

**Maintainer** Timothy Conwell <timconwell@gmail.com>

**Description** Organizational framework for web development in R including functions to serve static and dynamic content via HTTP methods, includes the html5 package to create HTML pages, and offers other utility functions for common tasks related to web development.

**License** GPL (>= 2)

**Encoding** UTF-8

**Depends** httpuv, html5 (>= 1.0.0)

**RoxygenNote** 7.2.0

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-08-08 19:40:02 UTC

## R topics documented:

create_options . . . . .	2
endServer . . . . .	2
idAddAffixes . . . . .	3
idAddPrefix . . . . .	4
idAddSuffix . . . . .	4
idParseAffixes . . . . .	5
idParsePrefix . . . . .	5
idParseSuffix . . . . .	6
parseContentTypeHeader . . . . .	6
parseHTTP . . . . .	7
parseMultiPartFormData . . . . .	8
parseMultiPartFormParams . . . . .	9
parseQueryString . . . . .	9
serveHTTP . . . . .	10

**Index****13**


---

create_options	<i>Creates HTML option tags for each position of a list of values and labels by calling HTML5::option(), returning a string of HTML to pass to a select tag through HTML5::select().</i>
----------------	--

---

**Description**

Creates HTML option tags for each position of a list of values and labels by calling HTML5::option(), returning a string of HTML to pass to a select tag through HTML5::select().

**Usage**

```
create_options(x, selected = c(), add_blank = FALSE)
```

**Arguments**

x	A vector which will become the value/label for each option. If named, names become values.
selected	A value in the vector passed to mark as the initially selected option in the select tag.
add_blank	Boolean, If TRUE, adds a blank option to the top of x.

**Value**

A string, with an option tag each row of x.

**Examples**

```
create_options(
  x = c("New York", "Los Angeles", "Chicago"),
  selected = "Chicago"
)
```

---

endServer	<i>Stop HTTP server(s) by calling httpuv::stopServer() or httpuv::stopAllServers().</i>
-----------	---

---

**Description**

Stop HTTP server(s) by calling httpuv::stopServer() or httpuv::stopAllServers().

**Usage**

```
endServer(x = NULL, all = FALSE)
```

**Arguments**

x	A server object that was previously returned from serveHTTP.
all	TRUE/FALSE, if TRUE, calls httpuv::stopAllServers.

**Value**

Nothing.

**Examples**

```
endServer(all = TRUE)
```

---

idAddAffixes	<i>Add a prefix and suffix to an id</i>
--------------	---

---

**Description**

Add a prefix and suffix to an id

**Usage**

```
idAddAffixes(prefix, id, suffix, prefix_sep = "X", suffix_sep = "-")
```

**Arguments**

prefix	A string, the prefix to add.
id	A string to add a prefix and suffix to.
suffix	A string, the suffix to add.
prefix_sep	A string, the prefix separator to use. This should be different than suffix_sep.
suffix_sep	A string, the suffix separator to use. This should be different than prefix_sep.

**Value**

A string.

**Examples**

```
idAddAffixes("group1", "example", 1)
```

idAddPrefix                      *Add a prefix to an id*

---

**Description**

Add a prefix to an id

**Usage**

```
idAddPrefix(prefix, id, sep = "X")
```

**Arguments**

prefix	A string, the prefix to add.
id	A string to add a prefix to.
sep	A string, the separator to use.

**Value**

A string.

**Examples**

```
idAddSuffix("example", 1)
```

---

idAddSuffix                      *Add a suffix to an id*

---

**Description**

Add a suffix to an id

**Usage**

```
idAddSuffix(id, suffix, sep = "-")
```

**Arguments**

id	A string to add a suffix to.
suffix	A string, the suffix to add.
sep	A string, the separator to use.

**Value**

A string.

**Examples**

```
idAddSuffix("example", 1)
```

---

idParseAffixes	<i>Remove a prefix and suffix from an id</i>
----------------	--

---

**Description**

Remove a prefix and suffix from an id

**Usage**

```
idParseAffixes(id, split = "X|-")
```

**Arguments**

id	A string to remove a prefix and suffix from.
split	A regular expression to use for splitting the prefix and suffix from the id.

**Value**

A named vector, with prefix, id, and suffix returned in that order.

**Examples**

```
idParseAffixes(idAddAffixes("group1", "example", 1))
```

---

idParsePrefix	<i>Remove a prefix from an id</i>
---------------	-----------------------------------

---

**Description**

Remove a prefix from an id

**Usage**

```
idParsePrefix(id, split = "X", position = 2)
```

**Arguments**

id	A string to remove a prefix from.
split	A string, the separator to use for splitting the id.
position	A integer vector, the position of the split string to return.

**Value**

A vector.

**Examples**

```
idParsePrefix(idAddPrefix("example", 1))
```

---

idParseSuffix	<i>Remove a suffix from an id</i>
---------------	-----------------------------------

---

**Description**

Remove a suffix from an id

**Usage**

```
idParseSuffix(id, split = "-", position = 1)
```

**Arguments**

id	A string to remove a suffix from.
split	A string, the separator to use for splitting the id.
position	A integer vector, the position of the split string to return.

**Value**

A vector.

**Examples**

```
idParseSuffix(idAddSuffix("example", 1))
```

---

parseContentTypeHeader	<i>Parse the content type header string to return the content type and boundary</i>
------------------------	---

---

**Description**

Parse the content type header string to return the content type and boundary

**Usage**

```
parseContentTypeHeader(x)
```

**Arguments**

x                    A string containing the content type header.

**Value**

A named list with "content\_type" and "boundary" if boundary is present.

**Examples**

```
parseContentTypeHeader("application/x-www-form-urlencoded")
```

---

<code>parseHTTP</code>	<i>Parse a HTTP request</i>
------------------------	-----------------------------

---

**Description**

Parse a HTTP request

**Usage**

```
parseHTTP(x, content_type_header = NULL, consolidate = TRUE)
```

**Arguments**

x                    The body of the HTTP request  
content\_type\_header    A string containing the content type header.  
consolidate          TRUE/FALSE, if TRUE, consolidates items with the same name.

**Value**

A named list.

**Examples**

```
parseHTTP("?form_id=example&col_name=Test+String", "application/x-www-form-urlencoded")
```

---

 parseMultiPartFormData

*Parse multi-part form data*


---

### Description

Parse multi-part form data

### Usage

```
parseMultiPartFormData(x, boundary)
```

### Arguments

x	A vector.
boundary	A string, the boundary used for the multi-part form data

### Value

A named list.

### Examples

```
parseMultiPartFormData(
  x = c(
    "-----WebKitFormBoundaryfBloeh49i0mYt05A",
    "Content-Disposition: form-data; name=\"form_name\"",
    "",
    "Example",
    "-----WebKitFormBoundaryfBloeh49i0mYt05A",
    "Content-Disposition: form-data; name=\"form_id\"",
    "",
    "test",
    "-----WebKitFormBoundaryfBloeh49i0mYt05A",
    "Content-Disposition: form-data; name=\"desktop_file\"; filename=\"limit_type.csv\"",
    "Content-Type: text/csv",
    "",
    "limit_type",
    "Aggregate",
    "Occurrence",
    "-----WebKitFormBoundaryfBloeh49i0mYt05A--"
  ),
  boundary = parseContentTypeHeader(
    "multipart/form-data; boundary=-----WebKitFormBoundaryfBloeh49i0mYt05A")[[ 'boundary' ]]
)
```



---

`parseMultiPartFormParams`*Helper function for parseMultiPartFormData*

---

**Description**

Helper function for parseMultiPartFormData

**Usage**

```
parseMultiPartFormParams(x)
```

**Arguments**

`x` A vector, a chunk of multi-part form data to parse.

**Value**

A named list.

**Examples**

```
parseMultiPartFormParams(c("Content-Disposition: form-data; name=\"form_name\"", "", "Example"))
```

---

`parseQueryString`*Parse a query string*

---

**Description**

Parse a query string

**Usage**

```
parseQueryString(x, split = "&", consolidate = TRUE)
```

**Arguments**

`x` A string containing the query string.  
`split` A string, the character to split by.  
`consolidate` TRUE/FALSE, if TRUE, consolidates items with the same name.

**Value**

A named list.

**Examples**

```
parseQueryString("?form_id=example&col_name=Test+String")
```

---

serveHTTP	<i>Conveniently create HTTP server using <code>httpuv::startServer()</code> or <code>httpuv::runServer()</code>.</i>
-----------	--

---

### Description

Conveniently create HTTP server using `httpuv::startServer()` or `httpuv::runServer()`.

### Usage

```
serveHTTP(
  host = "127.0.0.1",
  port = 5001,
  persistent = FALSE,
  static = list(),
  dynamic = list(),
  lapply_staticPath = TRUE,
  static_path_options = list(indexhtml = TRUE, fallthrough = FALSE, html_charset =
    "utf-8", headers = list(), validation = character(0), exclude = FALSE)
)
```

### Arguments

host	A string that is a valid IPv4 or IPv6 address that is owned by this server, which the application will listen on. "0.0.0.0" represents all IPv4 addresses and "::/0" represents all IPv6 addresses. Refer to host parameter of <code>httpuv::startServer()</code> for more details.
port	The port number to listen on. Refer to port parameter of <code>httpuv::startServer()</code> for more details.
persistent	TRUE/FALSE. If FALSE, calls <code>httpuv::startServer()</code> , which returns back to the R session (and would therefore not work with launching a persistent server through a system service as the R session would continue and likely exit/end). If TRUE, calls <code>httpuv::runServer()</code> , which does not return to the R session unless an error or interruption occurs and is suitable for use with system services to start or stop a server.
static	A named list, names should be URL paths, values should be paths to the files to be served statically (such as a HTML file saved somewhere) or <code>staticPath</code> objects if <code>lapply_staticPath</code> is FALSE.
dynamic	A named list, names should be URL paths, values should be named alists (use <code>alist</code> instead of <code>list</code> ) with alist names equaling a HTTP method (such as "GET" or "POST") and the values being expressions that when evaluated return a named list with valid entries for status, headers, and body as specified by <code>httpuv::startServer()</code> . Refer to <code>httpuv::startServer()</code> for more details on what can be returned as the response. ex. <code>list("/") = alist("GET" = get_function(req), "POST" = post_function(req))</code>

lapply\_staticPath

TRUE/FALSE, if TRUE, httpuv::staticPath will be applied to each element of static to create staticPath objects.

static\_path\_options

A named list, passed to httpuv::staticPathOptions.

## Details

serveHTTP is a convenient way to start a HTTP server that works for both static and dynamically created pages. It offers a simplified and organized interface to httpuv::startServer()/httpuv::runServer() that makes serving static and dynamic pages easier. For dynamic pages, the expression evaluated when a browser requests a dynamically served path should likely be an expression/function that has "req" as a parameter. Per the Rook specification implemented by httpuv, "req" is the R environment in which browser request information is collected. Therefore, to access HTTP request headers, inputs, etc. in a function served by a dynamic path, "req" should be a parameter of that function. For the dynamic parameter of serveHTTP, list("/") = alist("GET" = get\_homepage(req)) would be a suitable way to call the function get\_homepage(req) when the root path of a website is requested with the GET method. The req environment has the following variables: request\_method = req\$request\_method, script\_name = req\$script\_name, path\_info = req\$path\_info, query\_string = req\$query\_string, server\_name = req\$server\_name, server\_port = req\$server\_port, headers = req\$headers, rook\_input = req[["rook.input"]]\$read\_lines(), rook\_version = req[["rook.version"]]\$read\_lines(), rook\_url\_scheme = req[["rook.url\_scheme"]]\$read\_lines(), rook\_error\_stream = req[["rook.errors"]]\$read\_lines()

## Value

A HTTP web server on the specified host and port.

## Examples

```
# Run both functions and go to http://127.0.0.1:5001/ in a web browser
get_example <- function(req){

  html <- doctype(
    html(
      head(),
      body(
        h1("Hello"),
        p("Here is a list of some of the variables included in the req environment
that were associated with this request:"),
        ul(
          li(paste0("req$request_method = ", req$request_method)),
          li(paste0("req$script_name = ", req$script_name)),
          li(paste0("req$path_info = ", req$path_info)),
          li(paste0("req$query_string = ", req$query_string)),
          li(paste0("req$server_name = ", req$server_name)),
          li(paste0("req$server_port = ", req$server_port))
        ),
        p("You can use parseQueryString to deal with inputs passed through query strings as
well as passed through the input stream."),
        p("params <- parseQueryString(req[["rook.input"]]$read_lines()) will give you a
named list of parameters. See also parseHTTP.")
      )
    )
  }
```

```
)  
)  
)  
return(  
list(  
status = 200L,  
headers = list('Content-Type' = 'text/html'),  
body = html  
)  
)  
}
```

```
serveHTTP(  
host = "127.0.0.1",  
port = 5001,  
persistent = FALSE,  
static = list(),  
dynamic = list(  
"/" = alist(  
"GET" = get_example(req)  
)  
)  
)
```

# Index

[create\\_options](#), 2

[endServer](#), 2

[idAddAffixes](#), 3

[idAddPrefix](#), 4

[idAddSuffix](#), 4

[idParseAffixes](#), 5

[idParsePrefix](#), 5

[idParseSuffix](#), 6

[parseContentTypeHeader](#), 6

[parseHTTP](#), 7

[parseMultiPartFormData](#), 8

[parseMultiPartFormParams](#), 9

[parseQueryString](#), 9

[serveHTTP](#), 10