Package 'jrc'

August 23, 2023

Type Package

Title Exchange Commands Between R and 'JavaScript'

Version 0.6.0

Date 2023-08-22

Description An 'httpuv' based bridge between R and 'JavaScript'. Provides an easy way to exchange commands and data between a web page and a currently running R session.

License GPL-3

Imports httpuv, jsonlite, utils, stringr, stringi, mime, R6, R.utils

RoxygenNote 7.2.3

URL https://github.com/anders-biostat/jrc

BugReports https://github.com/anders-biostat/jrc/issues

Suggests testthat

Language en-GB

NeedsCompilation no

Author Svetlana Ovchinnikova [aut, cre],
Simon Anders [aut]

Maintainer Svetlana Ovchinnikova <s.ovchinnikova@zmbh.uni-heidelberg.de>

Repository CRAN

Date/Publication 2023-08-23 13:50:05 UTC

R topics documented:

`allowDirectories` *Allow server to access files in a directory*

Description

This function adds paths to existing directories to the list of allowed directories, which can be accessed from the server. To any request for files from outside of the allowed directories the server will response with `403 Forbidden` error. `rootDirectory` (see `openPage`) can always be accessed. By default, when the app is initialized, current working directory is added to the list of allowed directories. Further changes of the working directory will not have any affect on this list or files accessibility.

Usage

```
allowDirectories(dirs = NULL)
```

Arguments

`dirs`	Vector of paths to existing directories. Can be absolute paths, or paths relative to the current working directory. If the specified directory doesn't exist, it will be ignored and a warning will be produced. If `NULL`, returns absolute paths to all currently allowed directories.

Details

This function is a wrapper around `allowDirectories` method of class `App`.

Value

Absolute paths to all currently allowed directories, if `dirs = NULL`.

See Also

`openPage` (check arguments `rootDirectory` and `allowedDirectories`).

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
# The directories must exist
allowDirectories(c("~/directory1", "../anotherDirectory"))
dirs <- allowDirectories()
closePage()
## End(Not run)
```

`allowFunctions` *Allow function calls without authorization*

Description

Adds R function names to the list of functions, that can be called from a web page without manual confirmation on the R side.

Usage

```
allowFunctions(funs = NULL)
```

Arguments

`funs`	Vector of function names to be added to the list. If `NULL`, returns names of all currently allowed R functions.

Details

This function is a wrapper around `allowFunctions` method of class `App`.

Value

Names of all currently allowed functions if `funs = NULL`.

See Also

`allowVariables`, `authorize`, `openPage` (check argument `allowedFunctions`), `callFunction`.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
allowFunctions(c("myFunction1", "print", "someObject$method"))
funs <- allowFunctions()
closePage()
## End(Not run)
```

`allowVariables` *Allow variable assignment without authorization*

Description

This function adds variable names to the list of variables, that can be modified from a web page without manual confirmation on the R side.

Usage

```
allowVariables(vars = NULL)
```

Arguments

`vars`	Vector of variable names to be added to the list. If `NULL`, returns names of all currently allowed variables.

Details

This function is a wrapper around `allowVariables` method of class `App`.

Value

Names of all currently allowed variables if `vars = NULL`.

See Also

`allowFunctions`, `authorize`, `openPage` (check argument `allowedVariables`), `sendData`.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
allowVariables(c("myVariable", "anotherOne"))
vars <- allowVariables()
closePage()
## End(Not run)
```

App *App class*

Description

Object of this class represents the entire jrc-based app. It stores all the active connections, client-specific variables and all the global app settings.

You can create interactive apps by initializing new instances of this class and manage the apps with the methods that are described below. There are no limitations on the number of apps that can run simultaneously in one R session.

A wrapper function is also exported for almost each method (see links in the Methods section). This functions allow you to gain full control over the app without ever dealing with this class. However, in this case only a single app can run per R session. Attempt to create a new app (with `openPage` function) will force the existing one (if any) to stop. You can always get the `App` object for the currently running app with `getPage` function.

Methods

`new(rootDirectory = NULL, startPage = NULL, onStart = NULL, onClose = NULL, connectionNumber = Inf, allowed`
Creates a new instance of class `App`. Check `openPage` man page for information about arguments.

`startServer(port = NULL)` Starts a local server that listens to a given port. If `port = NULL`, picks a random available port. See also `openPage`.

`stopServer()` Closes all active sessions and stops a running server. See also `closePage`.

`openPage(useViewer = TRUE, browser = NULL)` Opens a new web page either in a browser, or in the R Studio viewer. If `useViewer = FALSE` and browser is not selected, a default installed browser is used. If browser is specified, `useViewer` is ignored. This method returns a new `Session` object, which should correspond to the page that has been just opened. However, if someone would start a new connection at the moment when `openPage` method is called, it may return a wrong session. See also `openPage`.

`getSession(sessionId = NULL)` Returns a session with the given ID or `NULL` if session with this ID doesn't exist. If `sessionId = NULL` and there is only one active session, returns it. See also `getSession`.

`closeSession(sessionId = NULL, inactive = NULL, old = NULL)` Closes WebSocket connection of one or multiple sessions and removes all the related data from the app. For more information on the arguments, please, check `closeSession` man page.

`getSessionIds()` Returns IDs of all currently active sessions. See also `getSessionIds`.

`setEnvironment(envir)` Specifies the outer environment of the app, in which all the messages from the web pages will be evaluated. For more information, please, check `setEnvironment`.

`allowFunctions(funs = NULL)` Adds function names to a list of allowed R functions. These functions can be called from a web page without authorization on the R side. If `funs = NULL`, returns a list of all currently allowed functions. For more information, please, check `allowFunctions`.

`allowVariables(vars = NULL)` Adds variable names to the list of allowed variables. These variables can be changed from a web page without authorization on the R side. If `vars = NULL`, then returns a vector of names of all currently allowed variables. For more information, please, check `allowVariables`.

`allowDirectories(dir = NULL)` Allows app to serve files from an existing directory. Files from the `rootDirectory` can always be accessed by the app. By default, the current working directory is added to the list of the allowed directories, when the app is initialized. All the subdirectories of the allowed directories can also be accessed. Attempt to request file from outside allowed directory will produce `403 Forbidden` error. If `dirs = NULL`, then returns a vector of names of all currently allowed directories. Also see `allowDirectories`.

`startPage(path = NULL)` Sets path to a starting web page of the app. Path can be full, relative to the app's root directory or relative to the current R working directory. If `path = NULL`, returns current path to the starting page.

`rootDirectory(path = NULL)` Sets path to the root directory for the server. Any file, requested by the server, will be looked for in this directory. Can be a full path or a path relative to the current R working directory. If `path = NULL`, returns path to the current root directory.

`setLimits(...)` Sets limits for memory usage, number of simultaneously active connections and amount of messages processed per second. These settings will apply for each new connection. To change memory usage for an existing session use method `setLimits` of class `Session`. For information about possible arguments, please, check `setLimits`.

`getPort()` Returns number of the port which the running server listens to. After the app has been initialized, the port number cannot be changed.

`authorize` *Authorize further message processing*

Description

`jrc` library allows one to get full control over the currently running R session from a web page. Therefore for security reasons one should manually authorize function calls, variable assignments or expression evaluations. All the received messages that are not processed automatically are given an ID and stored. This function allows a message with the given ID to be evaluated. It can also show a short description of the message and give user a choice between running it or discarding.

Usage

```
authorize(sessionId = NULL, messageId = NULL, show = FALSE)
```

Arguments

`sessionId`	ID of the session that received the message. If there is only one active session, this argument becomes optional.
`messageId`	ID of the message to be processed. If the session has only one stored message, this argument becomes optional.
`show`	If `TRUE` information about the message will be shown first. After that user gets a choice to go on with evaluation, to ignore the message (meaning it will be removed from memory) or to do nothing.

Details

Expressions has to be always authorized before evaluation. One can specify a list of variables that can be changed automatically and functions that can be called without authorization.

This function is a wrapper around `authorize` method of class `Session`.

See Also

`allowFunctions`, `allowVariables`, `setLimits`, `getSessionIds`, `getMessageIds`.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()

callFunction("jrc.sendCommand", list("k <<- 10"), wait = 1)
allowVariables("x")
callFunction("jrc.sendData", list("x", 15), wait = 1)
callFunction("jrc.sendData", list("y", 20), wait = 1)
msgId <- getMessageIds()
authorize(messageId = msgId[1])
#run that to first see some information about the message
#authorize(messageId = msgId[2], show = TRUE)

closePage()
## End(Not run)
```

---

callFunction *Trigger a function call*

Description

Calls a function in a web page by its name. It can also pass a list of arguments for the function and save the returned result to a variable.

Usage

```
callFunction(
  name,
  arguments = NULL,
  assignTo = NULL,
  wait = 0,
  sessionId = NULL,
  thisArg = NULL,
  ...
)
```

Arguments

name	Name of the function. If the function is a method of some object its name must contain the full chain of calls (e.g. myArray.sort or Math.rand).
arguments	List of arguments for the function. Note that in JavaScript arguments must be given in a fixed order, naming is not necessary and will be ignored.
assignTo	Name of a variable to which will be assigned the returned value of the called function.
wait	If wait > 0, after sending the message, R will wait for a reply for a given number of seconds. For this time (or until the reply is received), execution of other commands will be halted. Any incoming message from the session will be considered as a reply.
sessionId	An ID of the session to which the function call should be sent. Can also be a vector of multiple session IDs. If NULL, the function call will be sent to all currently active sessions.
thisArg	JavaScript functions (methods) can belong to some object, which is referred to as this inside the function (e.g. in someObject.myFunction() function myFunction is a method of someObject). thisArg specifies object that will be known as this inside the function. If NULL, the function will use its parent as this object (as it happens in JavaScript by default).
...	further arguments passed to sendData. It is used to send arguments to the web page.

Details

JavaScript counterpart is jrc.callFunction(name, arguments, assignTo, package, internal). Its arguments are:

name Name of an R function. If function name hasn't been previously added to the list of allowed functions (see allowFunctions or allowedFunctions argument of openPage), attempt to call it from a web page will require manual authorization on the R side.

arguments **(optional)** arguments for the function. This should be an Array (for unnamed arguments) or an Object with argument names as keys (for named arguments).

assignTo **(optional)** Name of the variable to which the returned value of the function will be assigned in the R session. If the variable name hasn't been previously added to the list of allowed variables (see allowVariables or allowedVariables argument of openPage), attempt to assign it from a web page will require manual authorization on the R side.

package **(optional)** If the function needs to be imported from an installed package, name of this package.

internal **(optional)** Whether assignment of the function returned value should happen internally or not. If true, result will be stored in the session environment and can be accessed from the outside with getSessionVariable function. If false, result will be saved to the outer environment of the app (see setEnvironment). By default, uses true for variables that already exist in the session environment (see setSessionVariables or sessionVariables argument of the openPage function) and false otherwise.

This function is a wrapper around callFunction method of class Session.

See Also

authorize, allowFunctions, allowVariables, setEnvironment, getSessionIds.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
callFunction("alert", list("Some alertText"))
callFunction("Math.random", assignTo = "randomNumber")
sendCommand("alert(randomNumber)")
closePage()

## End(Not run)
```

---

closePage *Stop server*

Description

Stops the server and closes all currently opened pages (if any). This function is a wrapper of `stopServer` method of class App.

Usage

```
closePage()
```

See Also

openPage

---

closeSession *Close one or several client sessions*

Description

Closes WebSocket connections for the selected client sessions and removes all the related information from memory. If no arguments are provided and there is only one active session, closes it. This function is a wrapper around method `closeSession` of class App.

Usage

```
closeSession(sessionId = NULL, inactive = NULL, old = NULL)
```

Arguments

sessionId	IDs of the sessions to close. Can be a vector of multiple IDs.
inactive	All sessions that were inactive (didn't receive any messages) for the specified amount of time (in seconds) will be closed.
old	All sessions that were opened for at least specified amount of time (in seconds) will be closed.

Examples

```
## Not run:
# to run this example an installed web browser is required
start <- Sys.time()
openPage()

app <- getPage()
time <- Sys.time()

app$openPage(FALSE)
app$openPage(FALSE)

print(getSessionIds())

# No sessions will be closed
closeSession(old = Sys.time() - start)
print(getSessionIds())

# One session (the one that has been opened first) will be closed
closeSession(old = Sys.time() - time)
print(getSessionIds())

time <- Sys.time()
sendCommand("jrc.sendCommand('print(\"Hi!\")')", sessionId = getSessionIds()[1], wait = 3)

# this will close all sessions except for the one, that has just send a command to R session
closeSession(inactive = Sys.time() - time)

# if there is only one active session, sessionId becomes an optional argument
closeSession()

closePage()
## End(Not run)
```

---

getMessageIds *Get IDs of all stored messages*

Description

Returns IDs of all currently stored messages.

Usage

```
getMessageIds(sessionId = NULL, simplify = TRUE)
```

Arguments

sessionId	ID of the session for which to return message IDs. Can also be a vector of multiple session IDs. If NULL, returns message IDs for all currently active sessions.
simplify	If TRUE and only one session ID is provided (or there is only one active session), returns a vector of message IDs. Otherwise returns a named list with one vector for each requested session.

Details

For security reasons, most of the messages that are received from web pages require manual authorization in the R session with authorize function. Until that happens, messages are given randomly generated IDs and are stored in memory.

This function is a wrapper around method getMessageIds of class Session.

Value

Either a named list or a vector with message IDs.

See Also

authorize, getSessionIds.

---

getPage *Get the currently running app*

Description

jrc offers two ways to control an interactive app. One is by using methods of classes App and Session. This allows one to have any number of apps within one R session, but requires some understanding of object oriented programming. Another way is to use provided wrapper functions that are exported by the package. These functions internally work with the App object, which is stored in the package namespace upon initialization with openPage function. getPage returns this object if any.

Usage

```
getPage()
```

Value

Object of class App or NULL if there is no active app.

getPort *Get number of the port on which the local server is running*

Description

This function returns port number for the running server. By default, a random available port is used. One can also set a port number as an argument of the openPage function. The port number can't be changed after the app was initialized.This function is a wrapper around method getPort of the class App.

Usage

```
getPort()
```

See Also

openPage

getSession *Get a session*

Description

Returns Session by its ID. This function is a wrapper around method getSession of class App.

Usage

```
getSession(sessionId = NULL)
```

Arguments

sessionId	ID of the session. If there is only one active session, this argument becomes optional.

Value

Object of class Session.

getSessionIds *Get IDs of all active sessions*

Description

Returns IDs of all currently active sessions. An ID is a randomly generated combination of 6 letters and numbers that is assigned to each session upon opening. This function is a wrapper around method `getSessionIds` of class `App`.

Usage

```
getSessionIds()
```

Value

Vector of session IDs.

getSessionVariable *Get a variable from a client session environment*

Description

This function returns a variable, how it is seen from a session, e.g. for all the received function calls and commands. It searches for the variable in the session environment first, and then, if variable is not found, checks enclosing frames of the environment, starting from the outer environment of the app (see `setEnvironment`). If the variable doesn't exist, throws an error.

Usage

```
getSessionVariable(varName, sessionId = NULL)
```

Arguments

`varName`	Name of the variable to search for. Must be a character.
`sessionId`	ID of the session. If there is only one active session, this argument becomes optional.

Details

This function is a wrapper around method `sessionVariables` of the class `Session`.

Value

Requested variable

See Also

setSessionVariables

Examples

```
## Not run:
# to run this example an installed web browser is required
f <- function(x) {x * 3}
openPage(allowedFunctions = "f", allowedVariables = "k", sessionVars = list(k = 0))
k <- getSessionVariable("k")
getPage()$openPage(FALSE)
id1 <- getSessionIds()[1]
id2 <- getSessionIds()[2]
sendCommand("jrc.callFunction('f', [10], 'k')", sessionId = id1, wait = 3)
sendCommand("jrc.callFunction('f', [20], 'k')", sessionId = id2, wait = 3)
k1 <- getSessionVariable("k", id1)
k2 <- getSessionVariable("k", id2)

closePage()
## End(Not run)
```

---

listen *Listen to the server*

Description

When R session is not interactive, messages from the server are not processed automatically. In this case, one needs to keep this function running. This function, is a wrapper around run_now or service. It runs the service in a loop with a specified condition.

Usage

```
listen(time = Inf, activeSessions = NULL, condition = NULL)
```

Arguments

time	Time (in seconds), during which the R session should listen to the server. By default, the function runs until it is not interrupted (time = Inf).
activeSessions	The function runs, until there is at least one active session in the provided app. If there is only one active app, this argument can be set to TRUE for the same effect.
condition	Custom condition. This argument must be a function that returns TRUE or FALSE. R session will listen to the server, while the condition function returns TRUE.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
listen(time = 3)
## End(Not run)
```

---

openPage *Create a server*

Description

openPage starts a server and opens a new page with a WebSocket connection between it and the current R session. After that, messages can be exchanged between R session and the web page to generate content on the web page and to trigger calculations in R as a response to user activity on the page.

Usage

```
openPage(
  useViewer = TRUE,
  rootDirectory = NULL,
  startPage = NULL,
  port = NULL,
  browser = NULL,
  allowedFunctions = NULL,
  allowedVariables = NULL,
  allowedDirectories = getwd(),
  connectionNumber = Inf,
  sessionVars = NULL,
  onStart = NULL,
  onClose = NULL,
  onlyServer = FALSE
)
```

Arguments

useViewer	If TRUE, the new web page will be opened in the RStudio Viewer. If FALSE, a default web browser will be used (if other is not specified with the browser argument).
rootDirectory	A path to the root directory for the server. Any file, requested by the server will be searched for in this directory. If rootDirectory is not defined, the http_root in the package directory will be used as a root directory.
startPage	A path to an HTML file that should be used as a starting page of the app. It can be an absolute path to a local file, or it can be relative to the rootDirectory or to the current R working directory. If startPage is not defined, an empty page will be used. The file must have *.html* extension.

`port` Defines which TCP port the server will listen to. If not defined, random available port will be used (see `randomPort`).

`browser` A browser in which to open a new web page. If not defined, default browser will be used. For more information check `browseURL`. If this argument is specified, `useViewer` will be ignored.

`allowedFunctions`
List of functions that can be called from a web page without any additional actions on the R side. All other functions will require authorization in the current R session before they are called. This argument should be a vector of R function names. Check `authorize` and `allowFunctions` for more information.

`allowedVariables`
List of variables that can be modified from a web page without any additional actions on the R side. All other variable reassignments must be confirmed in the current R session. This argument should be a vector of variable names. Check `authorize` and `allowVariables` for more information.

`allowedDirectories`
List of directories that can be accessed by the server. This argument should be a vector of paths (absolute or relative to the current working directory) to existing directories. Check `allowDirectories` for more information.

`connectionNumber`
Maximum number of connections that is allowed to be active simultaneously.

`sessionVars` Named list of variables, that will be declared for each session, when a new connection is opened. Any changes to these variables will affect only a certain session. Thus they can be used, for instance, to store a state of each session. For more information, please, check `setSessionVariables`.

`onStart` A callback function that will be executed, when a new connection is opened. This function gets a single argument, which is an object of class `Session`. General purpose of the function is to populate each new web page with some default content.

`onClose` A callback function that will be executed, when a connection is closed. This function gets a single argument, which is an object of class `Session`. General purpose of the function is to store session variables if needed or in any other form to finalize user's interaction with the app.

`onlyServer` If `TRUE`, then an app will initialise without trying to open a new page in a browser.

Details

`jrc` supports four types of messages:

- Commands are pieces of R or JavaScript code that will be evaluated on the receiving side. Note, that any command from a web page must be authorized in the R session for security reasons. A message with information about how to do that is printed in the console each time a command is received. For more information, please, check `sendCommand`.
- Data is any variable that is sent to or from the R session. It must always come with a name of the variable to which it should be assigned on the receiving side. For more information, please, check `sendData`.

- Function calls can be triggered on each side of the WebSocket connection. Alongside the function name, one can also send a list of arguments and name of a variable to which the returned value of the function will be assigned. For more information, please, check `callFunction`.
- Unlike other types of messages, HTML code can be sent only from the R session to a web page. This code will be added to the body of the page.

`openPage` function is a wrapper around several methods of class `App`. First, it creates an instance of this class. Then it starts a server that listens to the given port. And finally, it attempts to open a new web page. It also stores a new app object in the package namespace, which allows other wrapper functions to access it.

Value

Object of class `App`.

See Also

`closePage`, `setEnvironment`, `setLimits`, `allowVariables`, `allowFunctions`, `setSessionVariables`.

`removeMessage` *Removes a stored message*

Description

Removes a message from the storage of a session. This function is a wrapper around method `removeMessage` of class `Session`.

Usage

```
removeMessage(sessionId = NULL, messageId = NULL)
```

Arguments

`sessionId`	ID of the session from where to remove a message. If there is only one active session, this argument becomes optional.
`messageId`	ID of the message to remove. If there is only one stored message, this argument becomes optional.

See Also

`authorize`, `getMessageIds`.

removeSessionVariables *Remove variables from a client session environment*

Description

This function removes variables from the environment of a client session. It allows, for instance, to unmask a variable with the same name from the outer app environment (see setEnvironment) for the session (check the example below). This function is a wrapper around method sessionVariables of the class Session.

Usage

```
removeSessionVariables(varNames, sessionId = NULL)
```

Arguments

varNames	Names of variables to remove.
sessionId	ID of the session. If there is only one active session, this argument becomes optional.

See Also

setSessionVariables

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage(allowedVariables = "k", sessionVars = list(k = 10))

k <- -1
getPage()$openPage(FALSE)
id1 <- getSessionIds()[1]
id2 <- getSessionIds()[2]
removeSessionVariables("k", id1)
#this changes global 'k', since the variable is no longer masked
sendCommand("jrc.sendData('k', 1)", sessionId = id1, wait = 3)
#this doesn't affect global 'k'
sendCommand("jrc.sendData('k', 5)", sessionId = id2, wait = 3)
local_k <- getSessionVariable("k", id2)

closePage()
## End(Not run)
```

sendCommand *Send a command to a web page*

Description

sendCommand sends JavaScript code through the selected WebSocket connection and evaluates it on the specified web page. Use JavaScript function jrc.sendCommand to send R code from the web page and evaluate it in the current R session. All commands send to R from the server will be evaluated only after authorization in the currently running R session (see authorize).

Usage

```
sendCommand(command, sessionId = NULL, wait = 0)
```

Arguments

command	A line (or several lines separated by \n) of JavaScript code. This code will be directly evaluated on the web page. No R-side syntax check is performed.
sessionId	An ID of the session to which the command should be sent. Can also be a vector of multiple session IDs. If NULL, the command will be sent to all currently active sessions.
wait	If wait > 0, after sending the message, R will wait for a reply for a given number of seconds. For this time (or until the reply is received), execution of other commands will be halted. Any incoming message from the session will be considered as a reply.

Details

Each opened page gets its own environment, where all the commands are evaluated. Any changes made with the usual assignment operator <- will be limited to this page-specific environment. The changes are still saved, but can be accessed only with getSessionVariable function. To make changes outside of the page-specific environment use <<- instead.

In JavaScript one should use windows.varibleName = "SomeValue" instead of varibleName = "SomeValue", in order to make the variable accessible outside of the current sendCommand call.

This function is a wrapper around sendCommand method of class Session.

See Also

authorize, sendData, sendHTML, callFunction, openPage, getSessionIds.

Examples

```
## Not run:
# to run this example an installed web browser is required
k <- 0
openPage()
```

```
sendCommand(paste0("button = document.createElement('input');",
              "button.type = 'button';",
         "button.addEventListener('click', function() {jrc.sendCommand('k <<- k + 1')});",
              "button.value = '+1';",
              "document.body.appendChild(button);", collapse = "\n"))
closePage()
## End(Not run)
```

`sendData` *Send data to a web page*

Description

Sends a variable to a web page, where it is saved under a specified name. This function is a wrapper around `sendData` method of class `Session`.

Usage

```
sendData(
  variableName,
  variable,
  keepAsVector = FALSE,
  rowwise = TRUE,
  sessionId = NULL,
  wait = 0
)
```

Arguments

`variableName`	Name that the variable will have on the web page.
`variable`	Variable to send.
`keepAsVector`	If `TRUE`, variables with length 1 will be saved as arrays on the web page, otherwise they will be converted to atomic types.
`rowwise`	If `TRUE`, matrices and data.frames will be transformed into JavaScript objects or arrays row wise (e.g. a matrix will become an Array of its rows).
`sessionId`	An ID of the session to which the data should be sent. Can also be a vector of multiple session IDs. If `NULL`, the data will be sent to all currently active sessions.
`wait`	If `wait > 0`, after sending the message, R will wait for a reply for a given number of seconds. For this time (or until the reply is received), execution of other commands will be halted. Any incoming message from the session will be considered as a reply.

Details

To send data back from the web page to the current R session one should use`jrc.sendData(variableName, variable, internal)`. Its arguments are:

`variableName` Name that the variable will have in the R session. If variable name hasn't been previously added to the list of allowed variables (see `allowVariables` or `allowedVariables` argument of the `openPage` function), attempt to assign it from a web page will require manual authorization on the R side.

`variable` Variable to send.

`internal` **(optional)** Whether this variable should be used only by the session that sent it. If `true`, variable will be stored in the session-specific environment and can be accessed from the outside with `getSessionVariable` function. If `false`, variable will be saved to the outer environment of the app (see `setEnvironment`). By default, uses `true` for variables that already exist in the session specific environment (see `setSessionVariables` or `sessionVariables` argument of the `openPage` function.) and `false` otherwise.

See Also

`authorize`, `allowVariables`, `sendCommand`, `callFunction`, `sendHTML`, `openPage`, `getSessionIds`.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
x <- 1:100
sendData("x", x)
sendCommand("console.log(x);")
sendCommand("jrc.sendData('x', x.filter(function(e) {return e % 2 == 0}))")
closePage()
## End(Not run)
```

`sendHTML` *Send HTML to a web page*

Description

Sends a piece of HTML code to a web page and adds it at the end or the body element. This function is a wrapper around `sendHTML` method of class `Session`.

Usage

```
sendHTML(html = "", sessionId = NULL, wait = 0)
```

Arguments

`html`	HTML code that will be added to the web page.
`sessionId`	An ID of the session to which the HTML should be sent. Can also be a vector of multiple session IDs. If `NULL`, the HTML will be sent to all currently active sessions.
`wait`	If `wait > 0`, after sending the message, R will wait for a reply for a given number of seconds. For this time (or until the reply is received), execution of other commands will be halted. Any incoming message from the session will be considered as a reply.

See Also

`sendData`, `sendCommand`, `callFunction`, `openPage`.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage(FALSE)

sendHTML("Test...")
sendHTML("This is <b>bold</b>")
sendHTML("<table><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>")
## End(Not run)
```

`Session` *Session class*

Description

Objects of this class handle all the incoming and outgoing messages for one active connection. Please, avoid creating instances of this class manually. Each `Session` object is created when a WebSocket is opened and serves as a wrapper around it. A manually created object will not have a WebSocket connection and thus are not functional.

All sessions are stored within an object of class `App` and cannot exist and function without it. One can manipulate a session directly, using its methods described below, via methods of the corresponding `App` object or the provided wrapper function (links to them can be found in the Methods section).

Fields

`id` Automatically generated ID for this session. ID is a random combination of 6 letters or numbers. Please, do not change the value of this field.

`lastActive` Time of the last received message from the session's WebSocket. The timestamp is generated by the `Sys.time` function.

`startTime` Time when this session has been started (generated by the `Sys.time` function).

Methods

`getMessageIds()` Returns IDs of all currently stored messages. ID is a combination of 6 random letters and numbers generated when the message is stored. See also `getMessageIds`.

`authorize(messageId = NULL, show = FALSE)` Authorizes evaluation of a message. Check `authorize` for more information.

`removeMessage(messageId = NULL)` Removes a stored message. This can also be done with the `authorize` function (set `show = TRUE` and then select the "Ignore message" option). See also `removeMessage`.

`sendCommand(command, wait = 0)` Sends a JavaScript command to be evaluated on the web page. Check `sendCommand` for more information.

`callFunction(name, arguments = NULL, assignTo = NULL, wait = 0, thisArg = NULL, ...)` Calls an existing JavaScript function on the web page. Check `callFunction` for more information.

`sendData(variableName, variable, wait = 0, keepAsVector = FALSE, rowwise = TRUE)` Sends data and assigns it to a variable on the web page. Check `sendData` for more information.

`sendHTML(html, wait = 0)` Sends HTML code that will be appended to the web page. Check `sendHTML` for more information.

`sessionVariables(vars = NULL, varName = NULL, remove = NULL)` Sets or returns variables that are used (read or modified) only by this session. If both arguments are `NULL`, returns environment for this session. If `vars` is a named list, adds this variables to the session environment. If `varName` is a character, returns a variable with this name how it is seen from the session. If the variable doesn't exist, throws an error. If `remove` is a vector of characters, removes variables with these names from the session environment. One can add variables to the session environment, get one back and remove variables with a single function call. Check `setSessionVariables`, `getSessionVariable`, `removeSessionVariables` for more information.

`setLimits(limits)` Sets limits for memory usage, number of simultaneously active connections and amount of messages processed per second. For information about possible arguments, please, check `setLimits`. This method accepts all the same arguments, but they should be supplied in a form of list.

Note, that `Session` class has some other public methods that are not mentioned in this list. These methods are intended to be used only by other functions of `jrc` package and therefore are not documented.

`setEnvironment` *Set Environment*

Description

Defines the outer environment of the app. Outer environment is a parent for all session environments. It is used to store variables that are common for all the client sessions. The only way to make changes outside of the outer environment is to use the global assignment operator `<<-` if and only if changes are made to the variable that does not exist in the outer environment.

Usage

```
setEnvironment(envir)
```

Arguments

envir	Environment to be used as outer environment.

Details

By default, an environment where app was initialized (via openPage function or with App$new() call) is used.

This function is a wrapper around setEnvironment method of class App.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
e <- new.env()
setEnvironment(e)

sendCommand("jrc.sendData('x', 10)", wait = 3)
print(e$x)
closePage()

## End(Not run)
```

---

setLimits *Set security limits*

Description

This function allows to control memory usage and limit number of messages processed per second or simultaneously active connections to the app.

If an app is deployed on a server and is publicly available, it may be useful to limit resources that are available to each user. There are various things that can be controlled by this function: storage size and number of stored messages, maximal variable size, number of messages processed per second and bytes received per second.

Messages are all the communication received via web socket from an opened web page. Each message contains a command that is to be evaluated in the R session, name of a function to call or variable to store. If number or size of messages exceeds the preset limit, they are completely ignored by the app.

Usage

```
setLimits(
  maxCon = NULL,
  storageSize = NULL,
  storedMsg = NULL,
  varSize = NULL,
  msgPerSec = NULL,
  msgSize = NULL,
  bytesPerSec = NULL
)
```

Arguments

maxCon	Maximal allowed number of web socket connections simultaneously. A new connection is established whenever someone requests an HTML page from the server or when the openPage method of class App is used. If number of the allowed connections is reached the newly opened web socked will be immediately closed and the user will see a warning.
storageSize	Maximal total size of all stored messages in bytes.
storedMsg	Maximal number of messages that can be stored simultaneously.
varSize	Maximal size of a variable that can be received from a web page. Attempt to assign data of larger size to a variable will be ignored.
msgPerSec	Maximal number of messages that can be received per second. All extra messages will be disposed of immediately without any attempt to process their content.
msgSize	Maximal allowed size of a message in bytes. Note, that here a size of character string that contains all the received information is estimated. All larger messages will be ignored.
bytesPerSec	Number of bytes that can be received per second. After the limit is reached, all the incoming messages will be ignored.

Details

For security reasons, some messages has to be first authorized by the authorize function, before they can be processed. Such messages are saved until they are manually removed or authorized. If number or total size of the stored messages exceeds the limits, new messages are still saved, but the older ones are removed from the memory. If storage size is set to zero no messages can be stored and every message that requires authorization will be automatically discarded.

Size of variables or messages is estimated in object.size and is always measured in byte.

The limits are set for the entire app and are applied for each new connection. One can also change security limits for any connection separately by using method setLimits of a corresponding object of class Session.

This function is a wrapper for method setLimits of class App.

See Also

authorize, allowFunctions, allowVariables.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage()
setLimits(maxCon = 10)
setLimits(varSize = 10 * 1024^2)
closePage()
## End(Not run)
```

`setSessionVariables` *Adds variables to a session environment*

Description

Each client session in `jrc`, gets its own environment that can be accessed only by this session (or from the outside with the `getSessionVariable` function). General purpose of these environments is to store some session-specific information such as state of the app for each user. It can also be used to mask variables from the user: if there are two variables with the same name in the session environment and outside of it, user will not be able to see the latter one. This function adds new variables to a session environment or changes values of some existing ones.

Usage

```
setSessionVariables(vars, sessionId = NULL, makeDefault = FALSE)
```

Arguments

`vars`	Named list of variables to be added to a session environment. Names are required and will be used as variable names.
`sessionId`	ID of the session to which variables should be added. Can also be a vector of multiple session IDs. If `NULL`, then variables will be added to all currently active sessions.
`makeDefault`	If `TRUE` then, in addition, the specified variables will be added to each new opened session as default ones.

Details

This function is a wrapper around method `sessionVariables` of class `Session`. If `makeDefault = TRUE`, it is also a wrapper around method `sessionVariables` of class `App`. The first one changes the current state of the session environment, while the second specifies default variables for each new session.

See Also

`getSessionVariable`.

Examples

```
## Not run:
# to run this example an installed web browser is required
openPage(allowedFunctions = "f", allowedVariables = "res")

m <- 1
f <- function() {v * m}
setSessionVariables(list(v = 1:10, m = 2))

sendCommand("jrc.callFunction('f', [], 'res')", wait = 1)
print(res)

closePage()
## End(Not run)
```

Index