Package ‘svSocket’

October 14, 2022

Type Package
Version 1.1.0
Date 2022-05-04
Title 'SciViews' - Socket Server
Description A socket server allows to connect clients to R.
Maintainer Philippe Grosjean <phgrosjean@sciviews.org>
Depends R (>= 2.6.0)
Imports tcltk, svMisc (>= 0.9-68), utils
Suggests svHttp, spelling, covr, knitr, rmarkdown
License GPL-2

URL https://github.com/SciViews/svSocket,
      https://www.sciviews.org/svSocket/
RoxygenNote 7.1.1
VignetteBuilder knitr
Encoding UTF-8
Language en-US
NeedsCompilation no
Author Philippe Grosjean [aut, cre] (<https://orcid.org/0000-0002-2694-9471>),
       Matthew Dowle [ctb]
Repository CRAN
Date/Publication 2022-05-09 08:50:02 UTC

R topics documented:
close_socket_clients ............................................. 2
eval_socket_server ............................................. 2
get_socket_clients .............................................. 4
### close_socket_clients

Close one or more clients currently connected

**Description**

The socket servers asks to clients to nicely disconnect (possibly doing further process on their side). This function is used by `stop_socket_server()`, but it can also be invoked manually to ask for disconnection of a particular client. Note that, in this case, the client still can decide not to disconnect! The code send to ask for client disconnection is: `f`.

**Usage**

```r
close_socket_clients(sockets = "all", server_port = 8888)
closeSocketClients(sockets = "all", server_port = 8888)
```

**Arguments**

- `sockets` the list of socket client names (sockXXX) to close, or "all" (by default) to disconnect all currently connected clients.
- `server_port` the corresponding R socket server port.

**See Also**

`send_socket_clients()`
eval_socket_server

Usage

```
Usage

Usage

Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```

```
Usage

Usage
```
get_socket_clients

Description

List all clients currently connected to a given R socket server, or their names (sockXXX).
Usage

get_socket_clients(port = 8888)
getSocketClients(port = 8888)
get_socket_clients_names(port = 8888)
getSocketClientsNames(port = 8888)
getSocketServerName(port = 8888)

Arguments

port the port of the R socket server.

Value

`get_socket_clients()` returns a vector of character string with the address of clients in the form XXX.XXX.XXX.XXX:YYY where XXX.XXX.XXX.XXX is their ip address and YYY is their port. For security reasons, only localhost clients (on the same machine) can connect to the socket server. Thus, XXX.XXX.XXX.XXX is ALWAYS 127.0.0.1. However, the function returns the full IP address, just in case of further extensions in the future. The name of these items equals the corresponding Tcl socket name.

`get_socket_clients_names()` returns only a list of the socket client names.

See Also

`get_socket_servers()`

---

Description

Returns a list with all the ports of currently running R socket servers.

Usage

get_socket_servers()
getSocketServers()

Value

A character string vector, or NULL if no R socket server is currently running.
See Also

get_socket_clients(), get_socket_server_name(), start_socket_server

get_socket_server_name

Get the name of a R socket server

Description

Get the internal name given to a particular R socket server.

Usage

get_socket_server_name(port = 8888)

Arguments

port the port of the R socket server.

Value

A string with the server name, or NULL if it does not exist.

See Also

get_socket_servers()

par_socket_server

Get or set parameters specific to SciViews socket clients

Description

This function manage to persistently store sensible parameters for configuring communication between the server and the client, as well as, any other persistent data you may need. Parameters remain set even if the client disconnects and then reconnects to R, as long R was not restarted.

Usage

par_socket_server(client, server_port = 8888, client_socket = client, ...)

parSocket(client, server_port = 8888, client_socket = client, ...)
Arguments

client the client identification. By default, it is the socket identifier as it appears in get_socket_clients(). Since no attempt is made to check if the client really exists and is connected, you can create fake ones, outside of the socket server, to test your code for instance.

server_port the port on which the server is running, 8888 by default. Not important for fake socket client configurations.

client_socket the Tcl name of the socket where the client is connected. By default, it is the same as client name, but in case it was modified, do provide a correct client_socket string if you want to be able to activate a redirection to it (see socket_client_connection()).

... the parameters you want to change as named arguments. Non named arguments are ignored with a warning. If you specify arg = NULL, the corresponding variable is deleted from the environment.

Details

You can assign the environment to a variable, and then, access its content like if it was a list (e$var or e$var <- "new value"). To get a list of the content, use ls(par_socket_server(client, port)) or ls(par_socket_server(client, port), all.names = TRUE), but not names(par_socket_server(client, port)). As long as you keep a variable pointing on that environment alive, you have access to last values (i.e., changes done elsewhere are taken into account). If you want a frozen snapshot of the parameters, you should use myvar <- as.list(par_socket_server(client, port)).

There is a convenient placeholder for code send by the client to insert automatically the right socket and server_port in par_socket_server(): <<<s>>>. Hence, code that the client send to access or change its environment is just par_socket_server(<<<s>>>, bare = FALSE) or par_socket_server(<<<s>>>)$bare to set or get one parameter. Note that you can set or change many parameters at once.

Currently, parameters are:

- bare = TRUE|FALSE for "bare" mode (no prompt, no echo, no multilime; by default, bare = TRUE).
- multiline = TRUE|FALSE: does the server accept code spread on multiple lines and send in several steps (by default, yes, but works only if bare = FALSE).
- echo = TRUE|FALSE is the command echoed to the regular R console (by default echo = FALSE).
- last = "" string to append to each output (for instance to indicate that processing is done),
- prompt = ""> "", the prompt to use (if not in bare mode) and
- continue = ""+ "" the continuation prompt to use, when multiline mode is active. You can only cancel a multiline mode by completing the R code you are sending to the server, but you can break it too by sending <<<esc>>> before the next instruction. You can indicate <<<q>>> or <<<Q>>> at the very beginning of an instruction to tell R to disconnect the connection after the command is processed and result is returned (with <<<q>>>), or when the instructions are received but before they are processed (with <<<Q>>>). This is useful for "one shot" clients (clients that connect, send code and want to disconnect immediately after that). The code send
by the server to the client to tell him to disconnect gracefully (and do some housekeeping) is `\f`
send at the beginning of one line. So, clients should detect this and perform the necessary
actions to gracefully disconnect from the server as soon as possible, and he cannot send further
instructions from this moment on.

For clients that repeatedly connect and disconnect, but want persistent data, the default client iden-
tifier (the socket name) cannot be used, because that socket name would change from connection to
connection. The client must then provide its own identifier. This is done by sending `<<<id=myID>>>
at the very beginning of a command. This must be done for all commands! `myID` must use only
characters or digits. This code could be followed by `<<<e>>`, `<<<h>>` or `<<<H>>`. These com-
mands are intended for R editors/IDE. The first code `<<<e>>` sets the server into a mode that is
suitable to evaluate R code (including in a multi-line way). The other code temporarily configure
the server to run the command (in single line mode only) in a hidden way. They can be used to ex-
ecute R code without displaying it in the console (for instance, to start context help, to get a calltip,
or a completion list, etc.). The differences between `<<<h>>` and `<<<H>>` is that the former waits
for command completion and returns results of the command to the client before disconnecting,
while the latter disconnects from the client before executing the command.

There is a simple client (written in Tcl) available in the `/etc` subdirectory of this package installation.
Please, read the `ReadMe.txt` file in the same directory to learn how to use it. You can use this
simple client to experiment with the communication using these sockets, but it does not provide
advanced command line edition, no command history, and avoid pasting more than one line of code
into it.

Value

Returns the environment where parameters and data for the client are stored. To access those data,
see examples below.

See Also

`start_socket_server()`, `send_socket_clients()`, `get_socket_clients()`, `socket_client_connection()`

Examples

```r
# We use a fake socket client configuration environment
e <- par_socket_server("fake")
# Look at what it contains
ls(e)
# Get one data
e$bare
# ... or
par_socket_server("fake")$bare

# Change it
par_socket_server("fake", bare = FALSE)$bare
# Note it is changed too for e
e$bare

# You can change it too with
e$bare <- TRUE
e$bare
```
par_socket_server("fake")$bare

# Create a new entry
e$foo <- "test"
ls(e)
par_socket_server("fake")$foo
# Now delete it
par_socket_server("fake", foo = NULL)
ls(e)

# Our fake socket config is in SciViews:TempEnv environment
s <- search()
l <- length(s)
pos <- (1:l)[s == "SciViews:TempEnv"]
ls(pos = pos)  # It is named 'socket_client_fake'
# Delete it
rm(socket_client_fake, pos = pos)
# Do some house keeping
rm(list = c("s", "l", "pos"))

---

process_socket_server  The function that processes a command coming from the socket

**Description**

This is the default R function called each time data is send by a client through a socket. It is possible to customize this function and to use customized versions for particular R socket servers.

**Usage**

```r
process_socket_server(msg, socket, server_port, ...) processSocket(msg, socket, server_port, ...)
```

**Arguments**

- `msg`  
  the message send by the client, to be processed.

- `socket`  
  the client socket identifier, as in `get_socket_clients()`. This is passed by the calling function and can be used internally.

- `server_port`  
  the port on which the server is running, this is passed by the calling function and can be used internally.

- `...`  
  anything you want to pass to `process_socket_server()`, but it needs to rework `start_socket_server()` to use it.
Details

There are special code that one can send to R to easily turn the server (possibly temporarily) into a given configuration. First, if you want to persistently store parameters for your client in the R server and make sure you retrieve the same parameters the next time you reconnect, you should specify your own identifier. This is done by sending <<<id=myID>>> at the very beginning of each of your commands. Always remember that, if you do not specify an identifier, the name of your socket will be used. Since socket names can be reused, you should always reinitialize the configuration of your server the first time you connect to it.

Then, sending <<<esc>>> breaks current multiline code submission and flushes the multiline buffer. The sequence <<<q>>> at the beginning of a command indicates that the server wants to disconnect once the command is fully treated by R. Similarly, the sequence <<<Q>>> tells the server to disconnect the client before processing the command (no error message is returned to the client!).

It is easy to turn the server to evaluate R code (including multiline code) and return the result and disconnect by using the <<<e>>> sequence at the beginning of a command. Using <<<h>>> or <<<H>>> configures that server to process a (single-line code only) command silently and disconnect before (uppercase H) or after (lowercase h) processing that command. It is the less intrusive mode that is very useful for all commands that should be executed behind the scene between R and a R editor or IDE, like contextual help, calltips, completion lists, etc.). Note that using these modes in a server that is, otherwise, configured as a multi-line server does not break current multi-line buffer.

The other sequences that can be used are: <<<s>>> for a placeholder to configure the current server (with configuration parameters after it), and <<<n>>> to indicate a newline in your code (submitting two lines of code as a single one; also works with servers configured as single-line evaluators).

To debug the R socket server and inspect how commands send by a client are interpreted by this function, use options(debug.Socket = TRUE). This function uses svMisc::parse_text() and svMisc::capture_all() in order to evaluate R code in character string almost exactly the same way as if it was typed at the command line of a R console.

Value

The results of processing msg in a character string vector.

See Also

start_socket_server(), send_socket_clients(), par_socket_server(), svMisc::parse_text(), svMisc::capture_all()

Examples

```r
## Not run:
# A simple REPL (R eval/process loop) using basic features of processSocket()
repl <- function() {
  pars <- par_socket_server("repl", ",", bare = FALSE) # Parameterize the loop
cat("Enter R code, hit <CTRL-C> or <ESC> to exit\n") # First prompt
  repeat {
    entry <- readlines(n = 1) # Read a line of entry
    if (entry == ") entry <- "<<<esc>>>" # Exit from multiline mode
    cat(process_socket_server(entry, "repl", ",") # Process the entry
  }
```

send_socket_clients

Description

The text is send to one or more clients of the R socket server currently connected.

Usage

send_socket_clients(text, sockets = "all", server_port = 8888)

Arguments

text: the text to send to the client(s).
sockets: the Tcl name of the client(s) socket(s) currently connected (sockXXX), or "all" (by default) to send the same text to all connected clients.
server_port: the port of the server considered.

Examples

## Not run:
# Start an R process (R#1) and make it a server
library(svSocket)
server_port <- 8888 # Port 8888 by default, but you can change it
start_socket_server(port = server_port)

# Connect with the R socket server
con <- socketConnection(host = "localhost", port = 8888, blocking = FALSE)

# Now, go back to the server R#1
get_socket_clients() # You should have one client registered
# Send something to all clients from R#1
send_socket_clients("Hi there!")
socket_client_connection

Description

A 'sockclientconn' object is created that opens a connection from R to a SciViews socket client (that must be currently connected). A timeout is defined by options(timeout = XX) where XX is a number of seconds. In R, its default value is 60 sec.

Usage

socket_client_connection(
  client,
  server_port = 8888,
  socket,
  blocking = FALSE,
  open = "a",
  encoding = getOption("encoding")
)

## S3 method for class 'sockclientconn'
summary(object, ...)

socketClientConnection(
  client,
  server_port = 8888,
  socket,
  blocking = FALSE,
  open = "a",
  encoding = getOption("encoding")
)
start_socket_server

Arguments

- **client**: the client identification. By default, it is the socket identifier as it appears in `get_socket_clients()`. The client must be currently connected.
- **server_port**: the port on which the server is running, 8888 by default. This server must be currently running.
- **socket**: the Tcl socket name where the targeted client is connected. If not provided, it will be guessed from `client`, otherwise, `client` is ignored.
- **blocking**: logical. Should the connection wait that the data is written before exiting?
- **open**: character. How the connection is opened. Currently, only "a" for append (default) or "w" for write access are usable.
- **encoding**: the name of the encoding to use.
- **object**: A 'sockclientconn' object as returned by `socket_client_connection()`.
- **...**: further arguments passed to the method (not used for the moment).

Value

`socket_client_connection()` creates a 'sockclientconn' object redirects text send to it to the SciViews socket server client. It is inherits from a 'sockconn' object (see `socketConnection()`), and the only difference is that output is redirected to a Tcl socket corresponding to a given SciViews socket client currently connected.

See Also

`socketConnection()`, `send_socket_clients()`

start_socket_server  Start and stop a R socket server

Description

A R socket server is listening for command send by clients to a TCP port. This server is implemented in Tcl/Tk, using the powerful 'socket' command. Since it runs in the separate tcltk event loop, it is not blocking R, and it runs in the background; the user can still enter commands at the R prompt while one or several R socket servers are running and even, possibly, processing socket clients requests.

Usage

```r
start_socket_server(
  port = 8888,
  server_name = "Rserver",
  procfun = process_socket_server,
  secure = FALSE,
  local = !secure
)
```
startSocketServer(
    port = 8888,
    server_name = "Rserver",
    procfun = process_socket_server,
    secure = FALSE,
    local = !secure
)

stop_socket_server(port = 8888)
stopSocketServer(port = 8888)

Arguments

- **port**: the TCP port of the R socket server.
- **server_name**: the internal name of this server.
- **procfun**: the function to use to process client's commands. By default, it is `process_socket_server()`.
- **secure**: do we start a secure (TLS) server? (not implemented yet)
- **local**: if TRUE, accept only connections from local clients, i.e., from clients with IP address 127.0.0.1. Set by default if the server is not secure.

Details

This server is currently synchronous in the processing of the command. However, neither R, nor the client are blocked during exchange of data (communication is asynchronous).

Note also that socket numbers are reused, and corresponding configurations are not deleted from one connection to the other. So, it is possible for a client to connect/disconnect several times and continue to work with the same configuration (in particular, the multiline code submitted line by line) if every command starts with `<<<id=myID>>>` where `myID` is an alphanumeric (unique) identifier. This property is call a stateful server. Take care! The R server never checks uniqueness of this identifier. You are responsible to use one that would not interfere with other, concurrent, clients connected to the same server.

For trials and basic testings of the R socket server, you can use the Tcl script `SimpleClient.Tcl`. See the `ReadMe.txt` file in the `/etc/` subdirectory of the `svSocket` package folder. Also, in the source of the `svSocket` package you will find `testCLI.R`, a script to torture test CLI for R (console).

Note

One can write a different `procfun()` function than the default one for special servers. That function must accept one argument (a string with the command send by the client) and it must return a character string containing the result of the computation.

See Also

`process_socket_server()`, `send_socket_clients()`
Index

* IO
  close_socket_clients, 2
  eval_socket_server, 2
  get_socket_clients, 4
  get_socket_server_name, 6
  get_socket_servers, 5
  par_socket_server, 6
  process_socket_server, 9
  send_socket_clients, 11
  socket_client_connection, 12
  start_socket_server, 13

* stateful socket server interprocess communication
  close_socket_clients, 2
  eval_socket_server, 2
  get_socket_clients, 4
  get_socket_server_name, 6
  get_socket_servers, 5
  par_socket_server, 6
  process_socket_server, 9
  send_socket_clients, 11
  socket_client_connection, 12
  start_socket_server, 13

* utilities
  eval_socket_server, 2
  get_socket_clients, 4
  get_socket_server_name, 6
  get_socket_servers, 5
  par_socket_server, 6
  process_socket_server, 9
  send_socket_clients, 11
  socket_client_connection, 12
  start_socket_server, 13

  close_socket_clients, 2
  close_socket_clients(), 11
  closeSocketClients
    (close_socket_clients), 2

dump(), 3

eval_socket_server, 2
  evalServer (eval_socket_server), 2

  get_socket_clients, 4
  get_socket_clients(), 5–9, 13
  get_socket_clients_names
    (get_socket_clients), 4
  get_socket_clients_names(), 5
  get_socket_server_name, 6
  get_socket_server_name(), 6
  get_socket_servers, 5
  get_socket_servers(), 5, 6
  getSocketClients (get_socket_clients), 4
  getSocketClientsNames
    (get_socket_clients), 4
  getSocketServerName
    (get_socket_clients), 4
  getSocketServers (get_socket_servers), 5

  par_socket_server, 6
  par_socket_server(), 10
  parSocket (par_socket_server), 6
  process_socket_server, 9
  process_socket_server(), 9, 11, 14
  processSocket (process_socket_server), 9

  send_socket_clients, 11
  send_socket_clients(), 2, 3, 8, 10, 13, 14
  sendSocketClients
    (send_socket_clients), 11
  socket_client_connection, 12
  socket_client_connection(), 7, 8, 13
  socketClientConnection
    (socket_client_connection), 12
  socketConnection(), 13
  source(), 3
  start_socket_server, 6, 13
  start_socket_server(), 8–10
  startSocketServer
    (start_socket_server), 13
stop_socket_server
  (start_socket_server), 13
stop_socket_server(), 2
stopSocketServer (start_socket_server), 13
summary.sockclientconn
  (socket_client_connection), 12
svMisc::capture_all(), 10
svMisc::parse_text(), 10