Package ‘svSocket’

February 20, 2015

Type Package
Version 0.9-57
Date 2014-03-02
Title SciViews GUI API - R Socket Server
Author Philippe Grosjean [aut, cre], Matthew Dowle [ctb]
Maintainer Philippe Grosjean <phgrosjean@sciviews.org>
Depends R (>= 2.6.0)
Imports tcltk, svMisc (>= 0.9-68)
Description Implements a simple socket server allowing to connect GUI clients to R
License GPL-2
URL http://www.sciviews.org/SciViews-R
BugReports https://r-forge.r-project.org/tracker/?group_id=194
NeedsCompilation no
Repository CRAN
Date/Publication 2014-03-02 13:20:12

R topics documented:

svSocket-package .................................................. 2
closeSocketClients .............................................. 3
evalServer ......................................................... 3
getSocketClients ................................................ 5
getSocketServerName .......................................... 6
getSocketServers ............................................... 7
parSocket ......................................................... 7
processSocket .................................................. 10
sendSocketClients .............................................. 11
socketClientConnection ................................. 13
startSocketServer ........................................... 14

Index 16
Description

The SciViews svSocket package provides a stateful, multi-client and preemptive socket server. Socket transaction are operational even when R is busy in its main event loop (calculation done at the prompt). This R socket server uses the excellent asynchronous socket ports management by Tcl, and thus, it needs a working version of Tcl/Tk (>= 8.4) and of the tcltk R package.

A particular effort has been made to handle requests the same way as if they were introduced at the command prompt, including presentation of the output. However, the server sends results back to the client only at the end of the computations. It means that any interaction during computation (for instance, using scan(), browser(), or par(ask = TRUE) is not echoed in the client on due time. If you parameterize the socket server to echo commands in the R console, such interaction would be possible from there. Another option is to run R in non-interactive mode (switching to non-interactive mode during the R session is possible by using the interactivity R package available on CRAN).

Although initially designed to server GUI clients, the R socket server can also be used to exchange data between separate R processes. The evalServer() function is particularly useful for this. Note, however, that R objects are serialized into a text (i.e., using dump()) format, currently. It means that the transfer of large object is not as efficient as, say Rserv (Rserv exchanges R objects in binary format, but Rserv is not stateful, clients do not share the same global workspace and it does not allow concurrent use of the command prompt).

See startSocketServer and processSocket for further implementation details.

Details

Package: svSocket
Type: Package
Version: 0.9-57
Date: 2014-03-02
License: GPL 2 or above, at your convenience

Author(s)

Philippe Grosjean & Matthew Doyle.

Maintainer: Ph. Grosjean <phgrosjean@sciviews.org>
closeSocketClients

Close one or more clients currently connected

Description
The socket server asks to clients to nicely disconnect (possibly doing further process on their side). This function is used by stopSocketServer, 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

closeSocketClients(sockets = "all", serverport = 8888)

Arguments

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

Author(s)
Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also
sendSocketClients

evalServer

Evaluate R code in a server process

Description
This function is designed to connect two R processes together using the socket server. It allows for piloting the server R process from a client R process, to evaluate R code in the server and return its results to the client.

Usage
evalServer(con, expr, send = NULL)

Arguments

e con: a socket connection with the server (see examples).
expr: an R expression to evaluate in the server.
send: optional data to send to the server.
Details

The function serializes R objects using `dump()` on the server, and it `source()`s the data on the client side. It has, thus, the same limitations as `dump`, (see `?dump`), and in particular, environments, external pointers, weak references and objects of type S4 are not serializable with `dump()` and will raise an error, or will produce unusable objects on the client side. Note also that lists or attributes of accepted objects may contain external pointers or environments, and thus, the whole object becomes unserializable. In that case, try to coerce your object, or extract a part of it on the server side to make sure you send just the part that is transferable between the two R processes.

Value

The object returned by the last evaluation in the server.

Author(s)

Matthew Dowle (<mdowle@mdowle.plus.com>)

See Also

`sendSocketClients`

Examples

```r
## Not run:
## Start an R process and make it a server
require(svSocket)
startSocketServer()

## Start a second R process and run this code in it (the R client):
require(svSocket)

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

L <- 10:20
L
evalServer(con, L) # L is not an the server, hence the error
evalServer(con, L, L) # Send it to the server
evalServer(con, L) # Now it is there
evalServer(con, L, L + 2)
L
evalServer(con, L)

## More examples
evalServer(con, "x <- 42") # Set x
evalServer(con, "y <- 10") # Set y
evalServer(con, x + y) # Don't need quotes
evalServer(con, "x + y") # but you can put quotes if you like
evalServer(con, x) # Same as get x
evalServer(con, "x + Y") # Return server side-error to the client
evalServer(con, x) # Keep working after an error
```
getSocketClients

# Embedded quotes are OK

evalServer(con, "x <- 'a'")

## Examples of sending data

evalServer(con, X - 42)  # Alternative way to assign to X

evalServer(con, Y, 1:10)

evalServer(con, X + Y)

X  # Generates an error, X is not here in the client, only on the server

evalServer(con, X)

evalServer(con, "Z <- X + 3")  # Send an assignment to execute remotely

evalServer(con, X + Z)

evalServer(con, "Z <- X + 1:1000; NULL")  # Same but prevents Y being returned

evalServer(con, length(Z))

Z <- evalServer(con, Z)  # Bring it back to client

Z

## Close connection with the R socket server

close(con)

## Now, switch back to the R server process and check
## that the created variables are there

d

x

y

X

Y

Z

## Stop the socket server

stopSocketServer()

## End (Not run)

---

getSocketClients  Get infos about socket clients

### Description

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

### Usage

```r
getSocketClients(port = 8888)
gETCH_SOCKET_CLIENTS_NAMES(port = 8888)
```

### Arguments

- **port** the port of the R socket server.
getSocketServerName

Value

getSocketClients() 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.

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

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

getSocketServers

Description

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

Usage

getSocketServerName(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.

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

getSocketServers
**getSocketServers**

*Get the ports of current R socket servers*

**Description**

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

**Usage**

```
getSocketServers()
```

**Value**

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

**Author(s)**

Philippe Grosjean (<phgrosjean@sciviews.org>)

**See Also**

`getSocketClients`, `getSocketServerName`, `startSocketServer`

---

**parSocket**

*Get or set parameters specific to Sciviews socket clients*

**Description**

This function manages 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**

```
parSocket(client, serverport = 8888, clientsocket = client, ...)
```

**Arguments**

- **client**: the client identification. By default, it is the socket identifier as it appears in `getSocketClients()`. 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.

- **serverport**: the port on which the server is running, 8888 by default. Not important for fake socket client configurations.
parSocket

clientsocket  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 clientsocket string if you want to be able to activate a redirection to it (see socketClientConnection()).

...  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.

Value

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

Note

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(parSocket(client, port)), or ls(parSocket(client, port), all.names = TRUE), but not names(parSocket(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(parSocket(client, port))

There is a convenient placeholder for code send by the client to insert automatically the right socket and serverport in parSocket(): <<$s>>>. Hence, code that the client send to access or change its environment is just parSocket(<<$s>>>, bare = FALSE) or parSocket(<<$s>>>)$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 multiline; be 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 completting 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 «qq» 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 «qq»), 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 immediatelly 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 identifier (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 «$h>>>>. These commands are intended for R editors/IDE. The first code «$e>>> sets the server into a mode that is suitable to evaluated R code (including in a multi-line way). The other code temporarily configure
parSocket

the server to run the command (in single line mode only) in a hidden way. They can be used to ex-

execute 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 "<<<" and "<<<" is that the former waits for com-
mand 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.

**Author(s)**

Philippe Grosjean (<phgrosjean@sciviews.org>)

**See Also**

`startSocketServer`, `sendSocketClients`, `getSocketClients`, `socketClientConnection`.

**Examples**

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

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

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

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

## Our fake socket config is in SciViews:TempEnv environment
s <- search()
1 <- length(s)
```
processSocket

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

processSocket(msg, socket, serverport, ...)

Arguments

msg the message send by the client, to be processed.
socket the client socket identifier, as in getSocketClients(). This is passed by the calling function and can be used internally.
serverport 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 processSocket(), but it needs to rework startServerSocket() to use it).

Value

The results of processing msg in a character string vector.

Note

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 "<?w" 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 \\
 sequence at the beginning of a command. Using the \\
 configure 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: \\
 for a placeholder to configure the current server (with configuration parameters after it), and \\
 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 Parse() and captureAll() in order to evaluate R code in character string almost exactly the same way as if it was introduced at the command line of a R console.

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

startSocketServer, sendSocketClients, parSocket, parseText, captureAll

Examples

```r
## Not run:
## A simple REPL (R eval/process loop) using basic features of processSocket()
repl <- function ()
{
  pars <- parSocket("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 <- "\\e>>>") # Exit from multiline mode
  cat(processSocket(entry, "repl", "\") # Process the entry
}
repl()

## End(Not run)
```

sendSocketClients

Send data to one or more clients through a socket

Description

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

sendSocketClients(text, sockets = "all", serverport = 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.
serverport the port of the server considered.

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

closeSocketClients, processSocket

Examples

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

## Start a second R process (R#2) and run this code in it (the R client):
require(svSocket)
## Connect with the R socket server
con <- socketConnection(host = "localhost", port = 8888, blocking = FALSE)

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

## Switch back to client R#2
## Since the connection is not blocking, you have to read lines actively
readLines(con)
## Note the final empty string indicating there is no more data
close(con)  # Once done...

## Switch to the R#1 server and close the server
stopSocketServer(port = serverport)

## End(Not run)
socketClientConnection

Open a connection to a SciViews socket client for write access

Description

A `sockclientconn` object is created that opens a connection from R to a SciViews socket client (that must be currently connected).

Usage

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

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

Arguments

- `client` the client identification. By default, it is the socket identifier as it appears in `getSocketClients()`. The client must be currently connected.
- `serverport` the port on which the server is running, 8888 by default. This server must be currently running.
- `socket` the Tcl socket name where the targetted 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 `socketClientConnection()`.
- `...` further arguments passed to the method (not used for the moment).

Value

`socketClientConnection()` 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.

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

`socketConnection`, `sendSocketClients`
startSocketServer

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

startSocketServer(port = 8888, server.name = "Rserver", procfun = processSocket, secure = FALSE, local = !secure)
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 processSocket().
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

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.

Note

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).
startSocketServer

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

processSocket, startHttpServer
Index

+Topic **IO**
  - closeSocketClients, 3
  - evalServer, 3
  - getSocketClients, 5
  - getSocketServerName, 6
  - getSocketServers, 7
  - parSocket, 7
  - processSocket, 10
  - sendSocketClients, 11
  - socketClientConnection, 13
  - startSocketServer, 14
+Topic **package**
  - svSocket-package, 2
+Topic **utilities**
  - evalServer, 3
  - getSocketClients, 5
  - getSocketServerName, 6
  - getSocketServers, 7
  - svSocket-package, 2

captureAll, 11
closeSocketClients, 3, 12

evalServer, 3

gSocketClients, 5, 7, 9
getSocketClientsNames
  (getSocketClients, 5
getSocketServerName, 6, 7
getSocketServers, 6, 7

parText, 11
parSocket, 7, 11
processSocket, 2, 10, 12, 15

sendSocketClients, 3, 4, 9, 11, 11, 13
socketClientConnection, 9, 13
socketConnection, 13
startHttpServer, 15
startSocketServer, 2, 7, 9, 11, 14
stopSocketServer, 3

stopSocketServer (startSocketServer), 14
summary.sockclientconn
  (socketClientConnection), 13
svSocket (svSocket-package), 2
svSocket-package, 2