Package: RSclient (via r-universe)

October 29, 2026

Version 0.7-12

Title Client for Rserve

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

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

Depends R (>= 2.7.0)

Description Client for Rserve, allowing to connect to Rserve instances and issue commands.

License GPL-2 | file LICENSE

URL https://www.rforge.net/RSclient/

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

NeedsCompilation yes

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

Date/Publication 2026-01-19 23:00:02 UTC

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

RemoteRef HEAD

RemoteSha 22144fab9ff44dc7a875e4b9e002387b916505ac

Contents

RC-methods 2
RCC 2
Rclient 7

Index **10**

RC-methods *Methods for the RserveConnection class*

Description

Basic methods (printing, comparison) for the RserveConnection class.

Usage

```
## S3 method for class 'RserveConnection'
print(x, ...)
## S3 method for class 'RserveConnection'
e1 == e2
## S3 method for class 'RserveConnection'
e1 != e2
```

Arguments

x	Rserve connection object
e1	Rserve connection object
e2	Rserve connection object
...	ignored

Value

`print` returns `x` invisibly

`==` and `!=` return a logical scalar

Author(s)

Simon Urbanek

RCC *Functions to talk to an Rserve instance (new version)*

Description

Rserve is a server providing R functionality via sockets. The following functions allow another R session to start new Rserve sessions and evaluate commands.

Usage

```
RS.connect(host = NULL, port = 6311L, tls = FALSE, verify = TRUE,
           proxy.target = NULL, proxy.wait = TRUE, chain, key, ca)
RS.login(rsc, user, password, pubkey, authkey)
RS.eval(rsc, x, wait = TRUE, lazy = TRUE)
RS.eval.qap(rsc, x, wait = TRUE)
RS.collect(rsc, timeout = Inf, detail = FALSE, qap = FALSE)
RS.close(rsc)
RS.assign(rsc, name, value, wait = TRUE)
RS.switch(rsc, protocol = "TLS", verify = TRUE, chain, key, ca)
RS.authkey(rsc, type = "rsa-authkey")
RS.server.eval(rsc, text)
RS.server.source(rsc, filename)
RS.server.shutdown(rsc)
RS.oobCallbacks(rsc, send, msg)
```

Arguments

host	host to connect to or socket path or NULL for local host
port	TCP port to connect to or 0 if unix socket is to be used
tls	if TRUE then SSL/TLS encrypted connection is started
verify	logical, if FALSE no verification of the server certificate is done, otherwise the certificate is verified and the function will fail with an error if it is not valid.
chain	string, optional, path to a file in PEM format that contains client certificate and its chain. The client certificate must be first in the chain.
key	string, optional, path to a file in PEM format containing the private key for the client certificate. If a client certificate is necessary for the connection, both chain and key must be set.
ca	string, optional, path to a file or directory holding any additional certificate authority (CA) certificates (including intermediate certificates) in PEM format that are required for the verification of the server certificate. Only relevant if verify=TRUE. If not set then it will be attempted to load the default trust store of the underlying library (which typically includes trusted root certificates).
proxy.target	proxy target (string) in the form <host>:<port> to be used when connecting to a non-transparent proxy that requires target designation. Not used when connected to transparent proxies or directly to Rserve instances. Note that literal IPv6 addresses must be quoted in [].
proxy.wait	if TRUE then the proxy will wait (indefinitely) if the target is unavailable due to too high load, if FALSE then the proxy is instructed to close the connection in such instance instead
rsc	Rserve connection as obtained from RS.connect
user	username for authentication (mandatory)
password	password for authentication
pubkey	public key for authentication

authkey	authkey (as obtained from RS.authkey) for secure authentication
x	expression to evaluate
wait	if TRUE then the result is delivered synchronously, if FALSE then NULL is returned instead and the result can be collected later with RS.collect
lazy	if TRUE then the passed expression is not evaluated locally but passed for remote evaluation (as if quoted, modulo substitution). Otherwise it is evaluated locally first and the result is passed for remote evaluation.
timeout	numeric, timeout (in seconds) to wait before giving up
detail	if TRUE then the result payload is returned in a list with elements value (unserialized result value of the command - where applicable) and rsc (connection which returned this result) which allows to identify the source of the result and to distinguish timeout from a NULL value. Otherwise the returned value is just the payload value of the result.
name	string, name of the symbol to assign to
value	value to assign – if missing name is assumed to be a symbol and its evaluated value will be used as value while the symbol name will be used as name
protocol	protocol to switch to (string)
type	type of the authentication to perform (string)
send	callback function for OOB_SEND
msg	callback function for OOB_MSG
text	string that will be parsed and evaluated on the server side
filename	name of the file (on the server!) to source
qap	logical, if TRUE then the result is assumed to be in QAP encoding (native Rserve protocol), otherwise it is assumed to be using R serialization.

Details

RS.connect creates a connection to a Rserve. The returned handle is to be used in all subsequent calls to client functions. The session associated witht he connection is alive until closed via RS.close.

RS.close closes the Rserve connection.

RS.login performs authentication with the Rserve. The user entry is mandatory and at least one of password, pubkey and authkey must be provided. Typical secure authentication is performed with RS.login(rsc, "username", "password", authkey=RS.authkey(rsc)) which ensures that the authentication request is encrypted and cannot be spoofed. When using TLS connections RS.authkey is not necessary as the connection is already encrypted.

RS.eval evaluates the supplied expression remotely.

RS.eval.qap behaves like RS.eval(..., lazy=FALSE), but uses the Rserve QAP serialization of R objects instead of the native R serialization.

RS.collect collects results from RS.eval(..., wait = FALSE) calls. Note that in this case rsc can be either one connection or a list of connections.

RS.assign assigns a value to the remote global workspace.

RS.switch attempts to switch the protocol currently used for communication with Rserve. Currently the only supported protocol switch is from plain QAP1 to TLS secured (encrypted) QAP1.

RS.oobCallbacks sets or retrieves the callback functions associated with OOB_SEND and OOB_MSG out-of-band commands. If neither send nor msg is specified then RS.oobCallbacks simply returns the current callback functions, otherwise it replaces the existing ones. Both functions have the form function(code, payload) where code is the OOB sub-code (scalar integer) and payload is the content passed in the OOB command. For OOB_SEND the result of the callback is disarded, for OOB_MSG the result is encoded and sent back to the server. Note that OOB commands in this client are only processed when waiting for the response to another command (typically RS.eval). OOB commands must be explicitly enabled in the server in order to be used (they are disabled by default).

RS.server.eval, RS.server.source and RS.server.shutdown are 'control commands' which are enqueued to be processed by the server asynchronously. They return TRUE on success which means the command was enqueued - it does not mean that the server has processed the command. All control commands affect only future connections, they do NOT affect any already established client connection (including the curretn one). RS.server.eval parses and evaluates the given code in the server instance, RS.server.source sources the given file in the server (the path is interpreted by the server, it is not the local path of the client!) and RS.server.shutdown attempts a clean shutdown of the server. Note that control commands are disabled by default and must be enabled in Rserve either in the configuration file with control enable or on the command line with --RS-enable-control (the latter only works with Rserve 1.7 and higher). If Rserve is configured with authentication enabled then only admin users can issues control commands (see Rserve documentation for details).

Parallel use

It is currently possible to use Rserve connections in parallel via mcparallel or mclapply if certain conditions are met. First, only clear connection (non-TLS) are eligible for parallel use and there may be no OOB commands. Then it is legal to use connections in forked process as long as both the request is sent and the result is collected in the same process while no other process uses the connection. However, connections can only be created in the parent session (except if the connection is created and subsequently closed in the child process).

One possible use is to initiate connections to a cluster and perform operations in parallel. For example:

```
library(RSclient)
library(parallel)
## try to connect to 50 different nodes
## cannot parallelize this - must be in the parent process
c <- lapply(paste("machine", 1:50, sep=''),
            function(name) try(RS.connect(name), silent=TRUE))
## keep only successful connections
c <- c[sapply(c, class) == "RserveConnection"]
## login to all machines in parallel (using RSA secured login)
unlist(mclapply(c,
       function(c) RS.login(c, "user", "password",, RS.authkey(c)),
mc.cores=length(c)))
## do parallel work ...
## pre-load some "job" function to all nodes
```

```
unlist(mclapply(c, function(c) RS.assign(c, job), mc.cores=length(c)))
## etc. etc. then call it in parallel on all nodes ...
mclapply(c, function(c) RS.eval(c, job()), mc.cores=length(c))

## close all
sapply(c, RS.close)
```

Note

The current version of the RSclient package supplies two clients - one documented in Rclient which uses R connections and one documented in RCC which uses C code and is far more versatile and efficient. This is the documentation for the latter which is new and supports features that are not supported by R such as unix sockets, SSL/TLS connections, protocol switching, secure authentication and multi-server collection.

Note

The RSclient package can be compiled with TLS/SSL support based on OpenSSL. Therefore the following statements may be true if RSclient binaries are shipped together with OpenSSL: This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). They are not true otherwise.

Author(s)

Simon Urbanek

Examples

```
## Not run:
  c <- RS.connect()
  RS.eval(c, data(stackloss))
  RS.eval(c, library(MASS))
  RS.eval(c, rlm(stack.loss ~ ., stackloss)$coeff)
  RS.eval(c, getwd())
  x <- rnorm(1e5)
  ## this sends the contents of x to the remote side and runs `sum` on
  ## it without actually creating the binding x on the remote side
  RS.eval(c, as.call(list(quote(sum), x)), lazy=FALSE)
  RS.close(c)

## End(Not run)
```

Rclient *Functions to talk to an Rserve*

Description

Rserve is a server providing R functionality via sockets. The following functions allow another R session to start new Rserve sessions and evaluate commands. The support is very rudimentary and uses only a fraction of the funtionality provided by Rserve. The typical use of Rserve is to connect to other applications, not necessarily to connect two R processes. However, it is not uncommon to have a cluster of Rserve machines so the following functions provide a simple client access.

For more complete cilent implementation see `src/clients` directory of the Rserve distribution which show a C/C++ client. Also available from the Rserve pages is a Java client (`JRclient`). See `http://rosuda.org/Rserve` for details.

Usage

```
RSconnect(host = "localhost", port = 6311)
RSlogin(c, user, pwd, silent = FALSE)
RSeval(c, expr)
RSclose(c)
RSshutdown(c, pwd = NULL, ctrl = FALSE)
RSdetach(c)
RSevalDetach(c, cmd = "")
RSattach(session)
RSassign(c, obj, name = deparse(substitute(obj)) )
RSserverEval(c, expr)
RSserverSource(c, file)
```

Arguments

`host`	host to connect to
`port`	TCP port to connect to
`c`	Rserve connection
`user`	username for authentication
`pwd`	password for authentication
`cmd`	command (as string) to evaluate
`silent`	flag indicating whether a failure should raise an error or not
`session`	session object as returned by `RSdetach` or `RSevalDetach`
`obj`	value to assign
`name`	name to assign to on the remote side
`expr`	R expression to evaluate remotely
`file`	path to a file on the server(!) that will be sourced into the main instance
`ctrl`	logical, if `TRUE` then control command (`CMD_ctrlShutdown`) is used for shutdown, otherwise the legacy `CMD_shutdown` is used instead.

Details

RSconnect creates a connection to a Rserve. The returned handle is to be used in all subsequent calls to client functions. The session associated witht he connection is alive until closed via RSclose.

RSlogin performs authentication with the Rserve. Currently this simple client supports only plain text authentication, encryption is not supported.

RSclose closes the Rserve connection.

RSeval evaluates the supplied expression remotely. expr can be either a string or any R expression. Use quote to use unevaluated expressions. The implementation of RSeval is very efficient in that it does not require any buffer on the remote side and uses native R serialization as the protocol. See exmples below for correct use.

RSdetach detaches from the current Rserve connection. The connection is closed but can be restored by using RSattach with the value returned by RSdetach. Technically the R on the other end is still running and waiting to be atached.

RSshutdown terminates the server gracefully. It should be immediately followed by RSclose since the server closes the connection. It can be issued only on a valid (authenticated) connection. The password parameter is currently ignored since password-protected shutdown is not yet supported. Please note that you should not terminate servers that you did not start. More recent Rserve installation can disable regular shutdown and only allow control shutdown (avaiable to control users only) which is invoked by specifying ctrl=TRUE.

RSevalDetach same as RSdetach but allows asynchronous evaluation of the command. The remote Rserve is instructed to evaluate the command after the connection is detached. Please note that the session cannot be attached until the evaluation finished. Therefore it is advisable to use another session when attaching to verify the status of the detached session where necessary.

RSattach resume connection to an existing session in Rserve. The session argument must have been previously returned from the RSdetach or RSevalDetach comment.

RSassign pushes an object to Rserve and assigns it to the given name. Note that the name can be an (unevaluated) R expression itself thus allowing constructs such as RSassign(c, 1:5, quote(a$foo)) which will result in a$foo <- 1:5 remotely. However, character names are interpreted literarly.

RSserverEval and RSserverSource enqueue commands in the server instance of Rserve, i.e. their effect will be visible for all subsequent client connections. The Rserve instance must have control commands enabled (not the default) in order to allow those commands. RSserverEval evaluates the supplied expression and RSserverSource sources the specified file - it must be a valid path to a file on the server, not the client machine! Both commands are executed asynchronously in the server, so the success of those commands only means that they were queued on the server - they will be executed between subsequent client connections. Note that only subsequent connections will be affected, not the one issuing those commands.

Author(s)

Simon Urbanek

Examples

```
## Not run:
  c <- RSconnect()
  data(stackloss)
```

```
  RSassign(c, stackloss)
  RSeval(c, quote(library(MASS)))
  RSeval(c, quote(rlm(stack.loss ~ ., stackloss)$coeff))
  RSeval(c, "getwd()")

  image <- RSeval(c, quote(try({
    attach(stackloss)
    library(Cairo)
    Cairo(file="plot.png")
    plot(Air.Flow,stack.loss,col=2,pch=19,cex=2)
    dev.off()
    readBin("plot.png", "raw", 999999)})))
  if (inherits(image, "try-error"))
    stop(image)

## End(Not run)
```

Index

!=.RserveConnection *(RC-methods)*, 2
∗ **interface**
    RC-methods, 2
    RCC, 2
    Rclient, 7
==.RserveConnection *(RC-methods)*, 2

print.RserveConnection *(RC-methods)*, 2

quote, *8*

RC-methods, 2
RCC, 2, *6*
Rclient, *6*, 7
RS.assign *(RCC)*, 2
RS.authkey *(RCC)*, 2
RS.close *(RCC)*, 2
RS.collect *(RCC)*, 2
RS.connect *(RCC)*, 2
RS.eval *(RCC)*, 2
RS.login *(RCC)*, 2
RS.oobCallbacks *(RCC)*, 2
RS.server.eval *(RCC)*, 2
RS.server.shutdown *(RCC)*, 2
RS.server.source *(RCC)*, 2
RS.switch *(RCC)*, 2
RSassign *(Rclient)*, 7
RSattach *(Rclient)*, 7
RSclose *(Rclient)*, 7
RSconnect *(Rclient)*, 7
RSdetach *(Rclient)*, 7
RSeval *(Rclient)*, 7
RSevalDetach *(Rclient)*, 7
RSlogin *(Rclient)*, 7
RSserverEval *(Rclient)*, 7
RSserverSource *(Rclient)*, 7
RSshutdown *(Rclient)*, 7