Package: rzmq (via r-universe)

October 18, 2026

Title R Bindings for 'ZeroMQ'

Version 0.9.16

Description Interface to the 'ZeroMQ' lightweight messaging kernel (see <https://zeromq.org/> for more information).

License GPL-3

SystemRequirements ZeroMQ >= 3.0.0: libzmq3-dev (deb) or zeromq-devel (rpm)

URL https://docs.ropensci.org/rzmq/

https://ropensci.r-universe.dev/rzmq

BugReports https://github.com/ropensci/rzmq/issues

RoxygenNote 7.2.3

NeedsCompilation yes

Author Whit Armstrong [aut], Michael Schubert [ctb], Jeroen Ooms [aut, cre] (ORCID: <https://orcid.org/0000-0002-4035-0289>)

Maintainer Jeroen Ooms <jeroenooms@gmail.com>

Config/pak/sysreqs libzmq3-dev

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

Date/Publication 2026-03-21 20:40:02 UTC

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

RemoteRef HEAD

RemoteSha 4c8bf68614e530e13bd9a304162245239c505e78

Contents

bind.socket 2
connect.socket 3
init.context 4
init.message 5
poll.socket 6

receive.multipart 9
receive.socket 9
send.multipart 10
send.socket 11
socket.options 12
zmq.error 14
zmq.version 15

Index **16**

`bind.socket` *Create an endpoint for accepting connections and bind it to the socket referenced by the socket argument.*

Description

The zmq_bind() function shall create an endpoint for accepting connections and bind it to the socket referenced by the socket argument.

The endpoint argument is a string consisting of two parts as follows: transport ://address. The transport part specifies the underlying transport protocol to use. The meaning of the address part is specific to the underlying transport protocol selected.

The following transports are defined:

inproc local in-process (inter-thread) communication transport, see zmq_inproc(7) ipc local inter-process communication transport, see zmq_ipc(7) tcp unicast transport using TCP, see zmq_tcp(7) pgm, epgm reliable multicast transport using PGM, see zmq_pgm(7) With the exception of ZMQ_PAIR sockets, a single socket may be connected to multiple endpoints using zmq_connect(), while simultaneously accepting incoming connections from multiple endpoints bound to the socket using zmq_bind(). Refer to zmq_socket(3) for a description of the exact semantics involved when connecting or binding a socket to multiple endpoints.

Usage

```
bind.socket(socket, address)
```

Arguments

`socket`	a zmq socket object.
`address`	a transport as described above.

Value

TRUE if operation succeeds or FALSE if the operation fails

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

connect.socket,bind.socket,receive.socket,send.socket,poll.socket

Examples

```
## Not run:

library(rzmq)
context = init.context()
in.socket = init.socket(context,"ZMQ_PULL")
bind.socket(in.socket,"tcp://*:5557")

out.socket = init.socket(context,"ZMQ_PUSH")
bind.socket(out.socket,"tcp://*:5558")

## End(Not run)
```

---

connect.socket *Connect the socket referenced by the socket argument to the endpoint specified by the endpoint argument.*

Description

The zmq_connect() function shall connect the socket referenced by the socket argument to the endpoint specified by the endpoint argument.

The endpoint argument is a string consisting of two parts as follows: transport ://address. The transport part specifies the underlying transport protocol to use. The meaning of the address part is specific to the underlying transport protocol selected.

The following transports are defined:

inproc local in-process (inter-thread) communication transport, see zmq_inproc(7) ipc local inter-process communication transport, see zmq_ipc(7) tcp unicast transport using TCP, see zmq_tcp(7) pgm, epgm reliable multicast transport using PGM, see zmq_pgm(7) With the exception of ZMQ_PAIR sockets, a single socket may be connected to multiple endpoints using zmq_connect(), while simultaneously accepting incoming connections from multiple endpoints bound to the socket using zmq_bind(). Refer to zmq_socket(3) for a description of the exact semantics involved when connecting or binding a socket to multiple endpoints.

Usage

```
connect.socket(socket, address)
disconnect.socket(socket, address)
```

Arguments

socket	a zmq socket object.
address	a transport as described above.

Value

TRUE if operation succeeds or FALSE if the operation fails

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

connect.socket, bind.socket, receive.socket, send.socket, poll.socket

Examples

```
## Not run:
library(rzmq)
context = init.context()
in.socket = init.socket(context,"ZMQ_PULL")
bind.socket(in.socket,"tcp://*:5557")

out.socket = init.socket(context,"ZMQ_PUSH")
bind.socket(out.socket,"tcp://*:5558")

## End(Not run)
```

---

init.context *initailize zmq context and zmq socket*

Description

initialize zmq context and zmq socket for to be used for further zmq operations.

Usage

```
init.context(threads=1L)
init.socket(context, socket.type)
```

Arguments

threads	number of threads for the context to use
context	a zmq context object.
socket.type	The ZMQ socket type requested e.g. ZMQ_REQ,ZMQ_REP,ZMQ_PULL,ZMQ_PUSH, etc.

Value

`init.context` returns a zmq context object. `init.socket` returns a zmq socket object.

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

`connect.socket`,`bind.socket`,`receive.socket`,`send.socket`,`poll.socket`

Examples

```
## Not run:

library(rzmq)
context = init.context()
in.socket = init.socket(context,"ZMQ_PULL")

## End(Not run)
```

---

init.message *create a message object.*

Description

Create a ZeroMQ message object that can be sent multiple times

Usage

```
init.message(data, serialize=TRUE, xdr=.Platform$endian=="big")
```

Arguments

data	the R object to be sent
serialize	whether to call serialize before sending the data
xdr	passed directly to serialize command if serialize is requested

Value

a ZeroMQ message object as external pointer

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

send.message.object

Examples

```
## Not run:

## remote execution server in rzmq
library(rzmq)
data = list(x=5)
msg = init.message(data)

## End(Not run)
```

---

poll.socket — *Polls a list of sockets, waiting for the presence of a nonblocking read, write, or error event.*

Description

The zmq_poll() function shall poll a list of a sockets for either read, write, or error conditions subject to a millisecond resolution timeout.

Usage

```
poll.socket(sockets, events, timeout=0L)
```

Arguments

sockets	a list of zmq socket objects.
events	a list of character vectors containing one or more events in {read, write, error}. The first element in the list corresponds to the first zmq socket, and so on...
timeout	the numbers of seconds to wait for events. Fractional seconds are supported. ZeroMQ guarantees at most millisecond resolution. A timeout of -1L blocks until an event occurs; a timeout of 0L is non-blocking.

Value

A list of pairlists corresponding to the polled zmq sockets. Each list has one of more tags from {read, write, error} with logical values indicating the results of the poll operation.

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

connect.socket,bind.socket,receive.socket,send.socket,poll.socket

Examples

```
## Not run:
library(rzmq)

# Create a set of REP-REQ sockets that
# have a Send, Receive, Send, Receive, ...
# pattern.
context = init.context()
in.socket = init.socket(context,"ZMQ_REP")
bind.socket(in.socket,"tcp://*:5557")

out.socket = init.socket(context,"ZMQ_REQ")
connect.socket(out.socket,"tcp://*:5557")

# Poll the REP and REQ sockets for all events.
events <- poll.socket(list(in.socket, out.socket),
                      list(c("read", "write", "error"),
                           c("read", "write", "error")),
                      timeout=0L)

# The REQ socket is writable without blocking.
paste("Is upstream REP socket readable without blocking?", events[[1]]$read)
paste("Is upstream REP socket writable without blocking?", events[[1]]$write)
```

```
paste("Is downstream REQ socket readable without blocking?", events[[2]]$read)
paste("Is downstream REQ socket writable without blocking?", events[[2]]$write)

# Send a message to the REP socket from the REQ socket. The
# REQ socket must respond before the REP socket can send
# another message.
send.socket(out.socket, "Hello World")

events <- poll.socket(list(in.socket, out.socket),
                      list(c("read", "write", "error"),
                           c("read", "write", "error")),
                      timeout=0L)

# The incoming message is readable on the REP socket.
paste("Is upstream REP socket readable without blocking?", events[[1]]$read)
paste("Is upstream REP socket writable without blocking?", events[[1]]$write)
paste("Is downstream REQ socket readable without blocking?", events[[2]]$read)
paste("Is downstream REQ socket writable without blocking?", events[[2]]$write)

receive.socket(in.socket)

events <- poll.socket(list(in.socket, out.socket),
                      list(c("read", "write", "error"),
                           c("read", "write", "error")),
                      timeout=0L)

# The REQ socket is waiting for a response from the REP socket.
paste("Is upstream REP socket readable without blocking?", events[[1]]$read)
paste("Is upstream REP socket writable without blocking?", events[[1]]$write)
paste("Is downstream REQ socket readable without blocking?", events[[2]]$read)
paste("Is downstream REQ socket writable without blocking?", events[[2]]$write)

send.socket(in.socket, "Greetings")

events <- poll.socket(list(in.socket, out.socket),
                      list(c("read", "write", "error"),
                           c("read", "write", "error")),
                      timeout=0L)

# The REP response is waiting to be read on the REQ socket.
paste("Is upstream REP socket readable without blocking?", events[[1]]$read)
paste("Is upstream REP socket writable without blocking?", events[[1]]$write)
paste("Is downstream REQ socket readable without blocking?", events[[2]]$read)
paste("Is downstream REQ socket writable without blocking?", events[[2]]$write)

# Complete the REP-REQ transaction cycle by reading
# the REP response.
receive.socket(out.socket)

## End(Not run)
```

receive.multipart *Receive multipart ZMQ message*

Description

Returns a list of raw vectors for the parts of a multipart message.

Usage

```
receive.multipart(socket)
```

Arguments

socket	The ZMQ socket from which to receive data

receive.socket *Receive a message from the socket referenced by the socket argument.*

Description

The zmq_recv() function shall receive a message from the socket referenced by the socket argument. If there are no messages available on the specified socket, by default the function shall block until the request can be satisfied. A non-blocking receive can be obtained by setting dont.wait to TRUE If there are no messages available on the specified socket, the receive.socket() call will return NULL immediately.

Usage

```
receive.socket(socket, unserialize=TRUE, dont.wait=FALSE)
receive.null.msg(socket)
receive.string(socket)
receive.int(socket)
receive.double(socket)
```

Arguments

socket	a zmq socket object
unserialize	whether to call unserialize on the received data
dont.wait	defaults to false, for blocking receive. Set to TRUE for non-blocking receive.

Value

the value sent from the remote server or NULL on failure. If dont.wait was TRUE and a message was not immediately available for receipt, NULL is returned and get.zmq.errno() is set to 11 or get.zmq.strerror() is set to EAGAIN.

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

connect.socket, bind.socket, receive.socket, send.socket, poll.socket

Examples

```
## Not run:
library(rzmq)

remote.exec <- function(out.socket,in.socket,fun,...) {
    send.socket(out.socket,data=list(fun=fun,args=list(...)))
    receive.socket(in.socket)
}

context = init.context()
out.socket = init.socket(context,"ZMQ_PUSH")
bind.socket(out.socket,"tcp://*:5557")

in.socket = init.socket(context,"ZMQ_PULL")
bind.socket(in.socket,"tcp://*:5558")


myfun <- function(x) {
    sum(abs(x))
}

remote.exec(out.socket,in.socket,myfun,rnorm(1e3))


## End(Not run)
```

---

send.multipart *Send multipart ZMQ message.*

Description

Queue a list of raw vectors to be sent as a series of ZMQ message parts. Each part before the last will be sent with the SNDMORE flag.

Usage

```
send.multipart(socket, parts)
```

Arguments

socket	The ZMQ socket on which to send data
parts	A list of raw vectors; each component will be sent as one part of the message, in the order of the list

---

`send.socket` *send a message.*

Description

Queue the message referenced by the msg argument to be sent to the socket referenced by the socket argument.

A successful invocation of send.socket does not indicate that the message has been transmitted to the network, only that it has been queued on the socket and ZMQ has assumed responsibility for the message.

Usage

```
send.socket(socket, data, send.more=FALSE, serialize=TRUE, xdr=.Platform$endian=="big")
send.null.msg(socket, send.more=FALSE)
send.raw.string(socket,data,send.more=FALSE)
```

Arguments

socket	a zmq socket object
data	the R object to be sent
send.more	whether this message has more frames to be sent
serialize	whether to call serialize before sending the data
xdr	passed directly to serialize command if serialize is requested

Value

a boolean indicating success or failure of the operation.

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

`connect.socket`,`bind.socket`,`receive.socket`,`send.socket`,`poll.socket`

Examples

```
## Not run:

## remote execution server in rzmq
library(rzmq)
context = init.context()
in.socket = init.socket(context,"ZMQ_PULL")
bind.socket(in.socket,"tcp://*:5557")

out.socket = init.socket(context,"ZMQ_PUSH")
bind.socket(out.socket,"tcp://*:5558")

while(1) {
   msg = receive.socket(in.socket)
   fun <- msg$fun
   args <- msg$args
   print(args)
   ans <- do.call(fun,args)
   send.socket(out.socket,ans)
}

## End(Not run)
```

---

socket.options *set a socket option.*

Description

The zmq_setsockopt() function shall set the option specified by the option_name argument to the value pointed to by the option_value argument for the ZMQ socket pointed to by the socket argument.

Usage

```
set.hwm(socket, option.value)
set.swap(socket, option.value)
set.affinity(socket, option.value)
set.identity(socket, option.value)
subscribe(socket, option.value)
unsubscribe(socket, option.value)
set.rate(socket, option.value)
set.recovery.ivl(socket, option.value)
set.recovery.ivl.msec(socket, option.value)
set.mcast.loop(socket, option.value)
set.sndbuf(socket, option.value)
set.rcvbuf(socket, option.value)
set.linger(socket, option.value)
set.reconnect.ivl(socket, option.value)
```

```
set.zmq.backlog(socket, option.value)
set.reconnect.ivl.max(socket, option.value)
get.rcvmore(socket)
get.last.endpoint(socket)
get.send.timeout(socket)
set.send.timeout(socket, option.value)
get.rcv.timeout(socket)
set.rcv.timeout(socket, option.value)
```

Arguments

socket	a zmq socket object
option.value	the new option value to bet set

Value

a boolean indicating success or failure of the operation or in the case of getsocketoptions, the value of the requsted option.

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

connect.socket,bind.socket,receive.socket,send.socket,poll.socket

Examples

```
## Not run:

library(rzmq)
context = init.context()
socket = init.socket(context,"ZMQ_REQ")

set.hwm(socket, 1L)
set.swap(socket, 100L)
set.identity(socket, "big.ass.socket")

## End(Not run)
```

`zmq.error` *get libzmq error numbers and error strings*

Description

return the error number or error description after a zmq call

Usage

```
zmq.errno()
zmq.strerror()
```

Value

an integer for zmq.errno or a string for zmq.strerror

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

`connect.socket`,`bind.socket`,`receive.socket`,`send.socket`

Examples

```
## Not run:

library(rzmq)
zmq.errno()
zmq.strerror()

## End(Not run)
```

zmq.version *get version of libzmq*

Description

return the version string of the system zmq library

Usage

```
zmq.version()
```

Value

a string of the following format: major.minor.patch

Author(s)

ZMQ was written by Martin Sustrik <sustrik@250bpm.com> and Martin Lucina <mato@kotelna.sk>. rzmq was written by Whit Armstrong.

References

http://www.zeromq.org http://api.zeromq.org http://zguide.zeromq.org/page:all

See Also

connect.socket, bind.socket, receive.socket, send.socket

Examples

```
## Not run:

library(rzmq)
zmq.version()

## End(Not run)
```

Index

∗ **utilities**
  bind.socket, 2
  connect.socket, 3
  init.context, 4
  init.message, 5
  poll.socket, 6
  receive.socket, 9
  send.socket, 11
  socket.options, 12
  zmq.error, 14
  zmq.version, 15

bind.socket, 2, *3–5*, *7*, *10, 11*, *13–15*

connect.socket, 3, *3–5*, *7*, *10, 11*, *13–15*

disconnect.socket *(*connect.socket*)*, 3

get.last.endpoint *(*socket.options*)*, 12
get.rcv.timeout *(*socket.options*)*, 12
get.rcvmore *(*socket.options*)*, 12
get.send.timeout *(*socket.options*)*, 12

init.context, 4
init.message, 5
init.socket *(*init.context*)*, 4

poll.socket, *3–5*, 6, *7*, *10, 11*, *13*

receive.double *(*receive.socket*)*, 9
receive.int *(*receive.socket*)*, 9
receive.multipart, 9
receive.null.msg *(*receive.socket*)*, 9
receive.socket, *3–5*, *7*, 9, *10, 11*, *13–15*
receive.string *(*receive.socket*)*, 9

send.message.object, *6*
send.message.object *(*send.socket*)*, 11
send.multipart, 10
send.null.msg *(*send.socket*)*, 11
send.raw.string *(*send.socket*)*, 11
send.socket, *3–5*, *7*, *10*, 11, *11*, *13–15*
set.affinity *(*socket.options*)*, 12
set.hwm *(*socket.options*)*, 12
set.identity *(*socket.options*)*, 12
set.linger *(*socket.options*)*, 12
set.mcast.loop *(*socket.options*)*, 12
set.rate *(*socket.options*)*, 12
set.rcv.timeout *(*socket.options*)*, 12
set.rcvbuf *(*socket.options*)*, 12
set.reconnect.ivl *(*socket.options*)*, 12
set.recovery.ivl *(*socket.options*)*, 12
set.send.timeout *(*socket.options*)*, 12
set.sndbuf *(*socket.options*)*, 12
set.swap *(*socket.options*)*, 12
set.zmq.backlog *(*socket.options*)*, 12
socket.options, 12
subscribe *(*socket.options*)*, 12

unsubscribe *(*socket.options*)*, 12

zmq.errno *(*zmq.error*)*, 14
zmq.error, 14
zmq.strerror *(*zmq.error*)*, 14
zmq.version, 15