

Package: pbdZMQ (via r-universe)

October 18, 2024

Version 0.3-13

Date 2024-09-15

Title Programming with Big Data -- Interface to 'ZeroMQ'

Depends R (>= 3.5.0)

LazyLoad yes

Copyright See files AUTHORS, COPYING, and COPYING.LESSER in 'inst/zmq_copyright/' for the 'ZeroMQ' source files in 'src/zmq_src/' which are under GPL-3.

Description 'ZeroMQ' is a well-known library for high-performance asynchronous messaging in scalable, distributed applications. This package provides high level R wrapper functions to easily utilize 'ZeroMQ'. We mainly focus on interactive client/server programming frameworks. For convenience, a minimal 'ZeroMQ' library (4.2.2) is shipped with 'pbdZMQ', which can be used if no system installation of 'ZeroMQ' is available. A few wrapper functions compatible with 'rzmq' are also provided.

SystemRequirements Linux, Mac OSX, and Windows, or 'ZeroMQ' library >= 4.0.4. Solaris 10 needs 'ZeroMQ' library 4.0.7 and 'OpenCSW'.

StagedInstall TRUE

License GPL-3

URL <https://pbdr.org/>

BugReports <https://github.com/snoweye/pbdZMQ/issues>

NeedsCompilation yes

Maintainer Wei-Chen Chen <wccsnow@gmail.com>

RoxygenNote 7.2.3

Author Wei-Chen Chen [aut, cre], Drew Schmidt [aut], Christian Heckendorf [aut] (file transfer), George Ostrouchov [aut] (Mac OSX), Whit Armstrong [ctb] (some functions are modified from the rzmq package for backwards compatibility), Brian Ripley [ctb] (C code of shellexec, and Solaris), R Core team [ctb] (some functions and headers are copied or modified from the R

source code), Philipp A. [ctb] (Fedora), Elliott Sales de Andrade [ctb] (sprintf), Spencer Aiello [ctb] (windows conf), Paul Andrey [ctb] (Mac OSX conf), Panagiotis Cheilaris [ctb] (add serialize version), Jeroen Ooms [ctb] (clang++ on MacOS ARM64), ZeroMQ authors [aut, cph] (source files in 'src/zmq_src/')

Repository CRAN

Date/Publication 2024-09-17 08:40:02 UTC

Contents

| | |
|---|-----------|
| pbdZMQ-package | 2 |
| address | 3 |
| C-like Wrapper Functions for ZeroMQ | 4 |
| Context Functions | 5 |
| File Transfer Functions | 6 |
| Initial Control Functions | 8 |
| ls | 9 |
| Message Function | 10 |
| Overwrite shpkg | 12 |
| Poll Functions | 13 |
| random_port | 15 |
| Send Receive Functions | 16 |
| Send Receive Multiple Raw Buffers | 18 |
| Set Control Functions | 19 |
| Socket Functions | 21 |
| Transfer Functions for Files or Directories | 23 |
| Wrapper Functions for rzmq | 25 |
| ZMQ Control Environment | 27 |
| ZMQ Control Functions | 28 |
| ZMQ Flags | 32 |
| Index | 34 |

pbdZMQ-package

Programming with Big Data – Interface to ZeroMQ

Description

ZeroMQ is a well-known library for high-performance asynchronous messaging in scalable, distributed applications. This package provides high level R wrapper functions to easily utilize ZeroMQ. We mainly focus on interactive client/server programming frameworks. For convenience, a minimal ZeroMQ library (4.1.0 rc1) is shipped with pbdZMQ, which can be used if no system installation of ZeroMQ is available. A few wrapper functions compatible with rzmq are also provided.

Details

The install command using default **pbZMQ**'s internal ZeroMQ library is

```
> R CMD INSTALL pbdZMQ_0.1-0.tar.gz
--configure-args="--enable-internal-zmq"
```

Other available variables include

| Variable | Default |
|-------------|--------------------|
| ZMQ_INCLUDE | -I./zmqsrc/include |
| ZMQ_LDFLAGS | -L./ -lzmq |
| ZMQ_POLLER | select |

See the package source file `pbdZMQ/configure.ac` for details.

For installation using an external ZeroMQ library, see the package source file `pbdZMQ/INSTALL` for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.ctx.new\(\)](#), [zmq.socket\(\)](#).

address

Form an Address/Endpoint

Description

A notationally convenient function for forming addresses/endpoints. It's a simple wrapper around the `paste0()` function.

Usage

```
address(host, port, transport = "tcp")
```

Arguments

| | |
|-----------|--|
| host | The host ip address or url. |
| port | A port; necessary for all transports except ipc. |
| transport | The transport protocol. Choices are "inproc", "ipc", "tcp", and "pgm"/"epgm" for local in-process (inter-thread), local inter-process, tcp, and pgm, respectively. |

Value

An address, for use with pbdZMQ functions.

Author(s)

Drew Schmidt

See Also

[zmq.bind](#)

Examples

```
address("localhost", 55555)
```

C-like Wrapper Functions for ZeroMQ
The C-like ZeroMQ Interface

Description

The basic interface to ZeroMQ that somewhat models the C interface.

Details

A list of all functions for this interface is as follows:

| | | |
|-------------------|---------------|----------------|
| zmq.bind() | zmq.close() | zmqconnect() |
| zmq.ctx.destroy() | zmq.ctx.new() | zmq.msg.recv() |
| zmq.msg.send() | zmq.recv() | zmq.send() |
| zmq.socket() | | |

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

Context Functions *Context Functions*

Description

Context functions

Usage

```
zmq.ctx.new()
```

```
zmq.ctx.destroy(ctx)
```

Arguments

ctx a ZMQ context

Details

`zmq.ctx.new()` initializes a ZMQ context for starting communication.

`zmq.ctx.destroy()` terminates the context for stopping communication.

Value

`zmq.ctx.new()` returns an R external pointer (ctx) generated by ZMQ C API pointing to a context if successful, otherwise returns an R NULL.

`zmq.ctx.destroy()` returns 0 if successful, otherwise returns -1 and sets errno to either EFAULT or EINTR.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.socket\(\)](#), [zmq.close\(\)](#), [zmq.bind\(\)](#), [zmq.connect\(\)](#).

Examples

```
## Not run:
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
zmq.ctx.destroy(context)

## End(Not run)
```

File Transfer Functions*File Transfer Functions*

Description

High level functions calling `zmq_send()` and `zmq_recv()` to transfer a file in 200 KiB chunks.

Usage

```
zmq.sendfile(
  port,
  filename,
  verbose = FALSE,
  flags = ZMQ.SR()$BLOCK,
  forcebin = FALSE,
  ctx = NULL,
  socket = NULL
)

zmq.recvfile(
  port,
  endpoint,
  filename,
  verbose = FALSE,
  flags = ZMQ.SR()$BLOCK,
  forcebin = FALSE,
  ctx = NULL,
  socket = NULL
)
```

Arguments

| | |
|----------|---|
| port | A valid tcp port. |
| filename | The name (as a string) of the in/out files. The in and out file names can be different. |
| verbose | Logical; determines if a progress bar should be shown. |

| | |
|----------|--|
| flags | A flag for the method used by <code>zmq_sendfile</code> and <code>zmq_recvfile</code> |
| forcebin | Force to read/send/recv/write in binary form. Typically for a Windows system, text (ASCII) and binary files are processed differently. If TRUE, "r+b" and "w+b" will be enforced in the C code. This option is mainly for Windows. |
| ctx | A ZMQ ctx. If NULL (default), the function will initial one at the beginning and destroy it after finishing file transfer. |
| socket | A ZMQ socket based on ctx. If NULL (default), the function will create one at the beginning and close it after finishing file transfer. |
| endpoint | A ZMQ socket endpoint. |

Details

If no socket is passed, then by default `zmq_sendfile()` binds a ZMQ_PUSH socket, and `zmq_recvfile()` connects to this with a ZMQ_PULL socket. On the other hand, a PUSH/PULL, REQ/REP, or REP/REQ socket pairing may be passed. In that case, the socket should already be connected to the desired endpoint. Be careful not to pass the wrong socket combination (e.g., do not do REQ/REQ), as this can put the processes in an un-recoverable state.

Value

`zmq_sendfile()` and `zmq_recvfile()` return number of bytes (invisible) in the sent message if successful, otherwise returns -1 (invisible) and sets `errno` to the error value, see ZeroMQ manual for details.

Author(s)

Drew Schmidt and Christian Heckendorf

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>
Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.msg.send\(\)](#), [zmq.msg.recv\(\)](#).

Examples

```
## Not run:
### Run the sender and receiver code in separate R sessions.

# Receiver
library(pbdZMQ, quietly = TRUE)
zmq.recvfile(55555, "localhost", "/tmp/outfile", verbose=TRUE)

# Sender
library(pbdZMQ, quietly = TRUE)
zmq.sendfile(55555, "/tmp/infile", verbose=TRUE)
```

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

Initial Control Functions

Initial controls in pbdZMQ

Description

Initial control functions

Usage

```
.zmqopt_get(main, sub = NULL, envir = .GlobalEnv)  
.zmqopt_set(val, main, sub = NULL, envir = .GlobalEnv)  
.zmqopt_init(envir = .GlobalEnv)
```

Arguments

| | |
|-------|--|
| main | a variable to be get from or set to |
| sub | a subvariable to be get from or set to |
| envir | an environment where ZMQ controls locate |
| val | a value to be set |

Details

`.zmqopt_init()` initials default ZMQ controls. `.zmqopt_get()` gets a ZMQ control. `.zmqopt_set()` sets a ZMQ control.

Value

`.zmqopt_init()` initial the ZMQ control at `envir`.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>
Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[.pbd_env](#).

Examples

```
## Not run:
library(pbdZMQ, quietly = TRUE)

ls(.pbd_env)
rm(.pbd_env)
.zmqopt_init()
ls(.pbd_env)

.pbd_env$ZMQ.SR$BLOCK
pbd_opt(bytext = "ZMQ.SR$BLOCK = 0L")

## End(Not run)
```

ls

A wrapper function for base::ls

Description

The `ls()` function with modification to avoid listing hidden pbd objects.

Usage

```
ls(
  name,
  pos = -1L,
  envir = as.environment(pos),
  all.names = FALSE,
  pattern,
  sorted = TRUE
)
```

Arguments

`name`, `pos`, `envir`, `all.names`, `pattern`, `sorted`
as the original `base::ls()`.

Details

As the original `base::ls()`, it returns the names of the objects.

Value

As the original `base::ls()` except when `all.names` is `TRUE` and `envir` is `.GlobalEnv`, hidden pbd objects such as `.pbd_env` and `.pbdenv` will not be returned.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

Examples

```
## Not run:
library(pbdRPC, quietly = TRUE)
ls(all.names = TRUE)
base::ls(all.names = TRUE)

## End(Not run)
```

| | |
|------------------|--------------------------|
| Message Function | <i>Message Functions</i> |
|------------------|--------------------------|

Description

Message functions

Usage

```
zmq.msg.send(
  rmsg,
  socket,
  flags = ZMQ.SR()$BLOCK,
  serialize = TRUE,
  serialversion = NULL
)

zmq.msg.recv(socket, flags = ZMQ.SR()$BLOCK, unserialize = TRUE)
```

Arguments

| | |
|---------------|--|
| rmsg | an R message |
| socket | a ZMQ socket |
| flags | a flag for method of send and receive |
| serialize | if serialize the rmsg |
| serialversion | NULL or numeric; the workspace format version to use when serializing. NULL specifies the current default version. The only other supported values are 2 and 3 |
| unserialize | if unserialize the received R message |

Details

zmq.msg.send() sends an R message.
zmq.msg.recv() receives an R message.

Value

`zmq.msg.send()` returns 0 if successful, otherwise returns -1 and sets `errno` to `EFAULT`.

`zmq.msg.recv()` returns the message if successful, otherwise returns -1 and sets `errno` to `EFAULT`.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.send\(\)](#), [zmq.recv\(\)](#).

Examples

```
## Not run:
### Using request-reply pattern.

### At the server, run next in background or the other window.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
responder <- zmq.socket(context, ZMQ.ST())$REP
zmq.bind(responder, "tcp://*:5555")
buf <- zmq.msg.recv(responder)
set.seed(1234)
ret <- rnorm(5)
print(ret)
zmq.msg.send(ret, responder)
zmq.close(responder)
zmq.ctx.destroy(context)

### At a client, run next in foreground.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
requester <- zmq.socket(context, ZMQ.ST())$REQ
zmq.connect(requester, "tcp://localhost:5555")
zmq.msg.send(NULL, requester)
ret <- zmq.msg.recv(requester)
print(ret)
zmq.close(requester)
zmq.ctx.destroy(context)

## End(Not run)
```

`Overwrite shpkg`*Overwrite rpath of linked shared library in osx*

Description

Overwrite rpath of linked shared library (e.g. `JuniperKernel/libs/JuniperKernel.so` in `osx` only. Typically, it is called by `.onLoad()` to update rpath if `pbzMQ` or `pbzMQ/libs/libzmq*.dylib` was moved to a personal directory (e.g. the binary package was installed to a none default path). The commands `otool` and `install_name_tool` are required. Permission may be needed (e.g. `sudo`) to overwrite the shared library.

Usage

```
overwrite.shpkg.rpath(  
  mylib = NULL,  
  mypkg = "JuniperKernel",  
  linkingto = "pbzMQ",  
  shlib = "zmq"  
)
```

Arguments

| | |
|------------------------|--|
| <code>mylib</code> | the path where <code>mypkg</code> was installed (default <code>NULL</code> that will search from R's path) |
| <code>mypkg</code> | the package for where <code>mypkg.so</code> will be checked or updated |
| <code>linkingto</code> | the package for where <code>libshpkg*.dylib</code> is located |
| <code>shlib</code> | name of shlib to be searched for |

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

Programming with Big Data in R Website: <https://pbdr.org/>

Examples

```
## Not run:  
### Called by .onLoad() within "JuniperKernel/R/zzz.R"  
overwrite.shpkg.rpath(mypkg = "JuniperKernel",  
                      linkingto = "pbzMQ",  
                      shlib = "zmq")  
  
## End(Not run)
```

Poll Functions

Poll Functions

Description

Poll functions

Usage

```
zmq.poll(socket, type, timeout = -1L, MC = ZMQ.MC())
```

```
zmq.poll.free()
```

```
zmq.poll.length()
```

```
zmq.poll.get.revents(index = 1L)
```

Arguments

| | |
|---------|---|
| socket | a vector of ZMQ sockets |
| type | a vector of socket types corresponding to socket argument |
| timeout | timeout for poll, see ZeroMQ manual for details |
| MC | a message control, see ZMQ.MC() for details |
| index | an index of ZMQ poll items to obtain revents |

Details

`zmq.poll()` initials ZMQ poll items given ZMQ socket's and ZMQ poll type's. Both socket and type are in vectors of the same length, while socket contains socket pointers and type contains types of poll. See [ZMQ.PO\(\)](#) for the possible values of type. ZMQ defines several poll types and utilize them to poll multiple sockets.

`zmq.poll.free()` frees ZMQ poll structure memory internally.

`zmq.poll.length()` obtains total numbers of ZMQ poll items.

`zmq.poll.get.revents()` obtains revent types from ZMQ poll item by the input index.

Value

`zmq.poll()` returns a ZMQ code and an errno, see ZeroMQ manual for details, no error/warning/interrupt in this R function, but some error/warning/interrupt may catch by the C function `zmq_poll()`.

`zmq.poll.length()` returns the total number of poll items

`zmq.poll.get.revents()` returns the revent type

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>
 Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.recv\(\)](#), [zmq.send\(\)](#).

Examples

```
## Not run:
### Using poll pattern.
### See demo/mspoller.r for details.

### Run next in background or the other window.
SHELL> Rscript wuserver.r &
SHELL> Rscript taskvent.r &
SHELL> Rscript mspoller.r

### The mspoller.r has next.
library(pbdZMQ, quietly = TRUE)

### Initial.
context <- zmq.ctx.new()
receiver <- zmq.socket(context, ZMQ.ST()$PULL)
zmq.connect(receiver, "tcp://localhost:5557")
subscriber <- zmq.socket(context, ZMQ.ST()$SUB)
zmq.connect(subscriber, "tcp://localhost:5556")
zmq.setsockopt(subscriber, ZMQ.SO()$SUBSCRIBE, "20993")

### Process messages from both sockets.
cat("Press Ctrl+C or Esc to stop mspoller.\n")
i.rec <- 0
i.sub <- 0
while(TRUE){
  ### Set poller.
  zmq.poll(c(receiver, subscriber),
           c(ZMQ.PO()$POLLIN, ZMQ.PO()$POLLIN))

  ### Check receiver.
  if(bitwAnd(zmq.poll.get.revents(1), ZMQ.PO()$POLLIN)){
    ret <- zmq.recv(receiver)
    if(ret$len != -1){
      cat("task ventilator:", ret$buf, "at", i.rec, "\n")
      i.rec <- i.rec + 1
    }
  }
}

### Check subscriber.
if(bitwAnd(zmq.poll.get.revents(2), ZMQ.PO()$POLLIN)){
  ret <- zmq.recv(subscriber)
  if(ret$len != -1){
```

```

        cat("weather update:", ret$buf, "at", i.sub, "\n")
        i.sub <- i.sub + 1
    }
}

if(i.rec >= 5 & i.sub >= 5){
    break
}

Sys.sleep(runif(1, 0.5, 1))
}

### Finish.
zmq.poll.free()
zmq.close(receiver)
zmq.close(subscriber)
zmq.ctx.destroy(context)

## End(Not run)

```

random_port

Random Port

Description

Generate a valid, random TCP port.

Usage

```
random_port(min_port = 49152, max_port = 65536)
```

```
random_open_port(min_port = 49152, max_port = 65536, max_tries = 100)
```

Arguments

min_port, max_port

The minimum/maximum value to be generated. The minimum should not be below 49152 and the maximum should not exceed 65536 (see details).

max_tries

The maximum number of times a random port will be searched for.

Details

By definition, a TCP port is an unsigned short, and so it can not exceed 65535. Additionally, ports in the range 1024 to 49151 are (possibly) registered by ICANN for specific uses.

random_port() will simply generate a valid, non-registered tcp port. random_unused_port() will generate a port that is available for socket connections.

random_open_port() finds a random port not already bound to an endpoint.

Author(s)

Drew Schmidt

References

"The Ephemeral Port Range" by Mike Gleason. https://www.ncftpd.com/ncftpd/doc/misc/ephemeral_ports.html

Examples

```
random_port()
```

Send Receive Functions

*Send Receive Functions***Description**

Send and receive functions

Usage

```
zmq.send(socket, buf, flags = ZMQ.SR())$BLOCK)
```

```
zmq.recv(
  socket,
  len = 1024L,
  flags = ZMQ.SR())$BLOCK,
  buf.type = c("char", "raw")
)
```

Arguments

| | |
|----------|---|
| socket | a ZMQ socket |
| buf | a buffer to be sent |
| flags | a flag for the method using by zmq_send and zmq_recv |
| len | a length of buffer to be received, default 1024 bytes |
| buf.type | buffer type to be received |

Details

zmq.send() is a high level R function calling ZMQ C API zmq_send() sending buf out.

zmq.recv() is a high level R function calling ZMQ C API zmq_recv() receiving buffers of length len according to the buf.type.

flags see [ZMQ.SR\(\)](#) for detail options of send and receive functions.

buf.type currently supports char and raw which are both in R object format.

Value

`zmq.send()` returns number of bytes (invisible) in the sent message if successful, otherwise returns -1 (invisible) and sets `errno` to the error value, see ZeroMQ manual for details.

`zmq.recv()` returns a list (`ret`) containing the received buffer `ret$buf` and the length of received buffer (`ret$len` which is less or equal to the input `len`) if successful, otherwise returns -1 and sets `errno` to the error value, see ZeroMQ manual for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.msg.send\(\)](#), [zmq.msg.recv\(\)](#).

Examples

```
## Not run:
### Using request-reply pattern.

### At the server, run next in background or the other window.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
responder <- zmq.socket(context, ZMQ.ST())$REP
zmq.bind(responder, "tcp://*:5555")
for(i.res in 1:5){
  buf <- zmq.recv(responder, 10L)
  cat(buf$buf, "\n")
  Sys.sleep(0.5)
  zmq.send(responder, "World")
}
zmq.close(responder)
zmq.ctx.destroy(context)

### At a client, run next in foreground.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
requester <- zmq.socket(context, ZMQ.ST())$REQ
zmq.connect(requester, "tcp://localhost:5555")
for(i.req in 1:5){
  cat("Sending Hello ", i.req, "\n")
  zmq.send(requester, "Hello")
  buf <- zmq.recv(requester, 10L)
```

```
    cat("Received World ", i.req, "\n")
  }
  zmq.close(requester)
  zmq.ctx.destroy(context)

## End(Not run)
```

Send Receive Multiple Raw Buffers

Send Receive Multiple Raw Buffers

Description

Send and receive functions for multiple raw buffers

Usage

```
zmq.send.multipart(socket, parts, serialize = TRUE)
```

```
zmq.recv.multipart(socket, unserialize = TRUE)
```

Arguments

| | |
|------------------------|--|
| socket | a ZMQ socket |
| parts | a vector of multiple buffers to be sent |
| serialize, unserialize | if serialize/unserialize the received multiple buffers |

Details

`zmq.send.multipart()` is a high level R function to send multiple raw messages parts at once.

`zmq.recv.multipart()` is a high level R function to receive multiple raw messages at once.

Value

`zmq.send.multipart()` returns.

`zmq.recv.multipart()` returns.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

`zmq.msg.send()`, `zmq.msg.recv()`.

Examples

```
## Not run:
### Using request-reply pattern.

### At the server, run next in background or the other window.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
responder <- zmq.socket(context, ZMQ.ST()$REP)
zmq.bind(responder, "tcp://*:5555")

ret <- zmq.recv.multipart(responder, unserialize = TRUE)
parts <- as.list(rep("World", 5))
zmq.send.multipart(responder, parts)
for(i in 1:5) cat(ret[[i]])

zmq.close(responder)
zmq.ctx.destroy(context)

### At a client, run next in foreground.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
requester <- zmq.socket(context, ZMQ.ST()$REQ)
zmq.connect(requester, "tcp://localhost:5555")

parts <- lapply(1:5, function(i.req){ paste("Sending Hello ", i.req, "\n") })
zmq.send.multipart(requester, parts)
ret <- zmq.recv.multipart(requester, unserialize = TRUE)
print(ret)

zmq.close(requester)
zmq.ctx.destroy(context)

## End(Not run)
```

Set Control Functions *Set controls in pbdZMQ*

Description

Set control functions

Usage

```
pbid_opt(..., bytext = "", envir = .GlobalEnv)
```

Arguments

| | |
|--------|--|
| ... | in argument format option = value to set .pbid_env\$option <- value inside the envir |
| bytext | in text format "option = value" to set .pbid_env\$option <- value inside the envir. |
| envir | by default the global environment is used. |

Details

pbid_opt() sets pbid options for ZMQ controls.

... allows multiple options in envir\$.pbid_env, but only in a simple way.

bytext allows to assign options by text in envir\$.pbid_env, but can assign advanced objects. For example, "option\$suboption <- value" will set envir\$.pbid_env\$option\$suboption <- value.

Value

No value is returned.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and Drew Schmidt.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbidr.org/>

See Also

[.pbid_env](#).

Examples

```
## Not run:
library(pbidZMQ, quietly = TRUE)

ls(.pbid_env)
rm(.pbid_env)
.zmqopt_init()
ls(.pbid_env)

.pbid_env$ZMQ.SR$BLOCK
pbid_opt(bytext = "ZMQ.SR$BLOCK <- 0L")
```

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

Socket Functions

Socket Functions

Description

Socket functions

Usage

```
zmq.socket(ctx, type = ZMQ.ST())$REP)
```

```
zmq.close(socket)
```

```
zmq.bind(socket, endpoint, MC = ZMQ.MC())
```

```
zmq.connect(socket, endpoint, MC = ZMQ.MC())
```

```
zmq.disconnect(socket, endpoint, MC = ZMQ.MC())
```

```
zmq.setsockopt(socket, option.name, option.value, MC = ZMQ.MC())
```

```
zmq.getsockopt(socket, option.name, option.value, MC = ZMQ.MC())
```

Arguments

| | |
|--------------|---|
| ctx | a ZMQ context |
| type | a socket type |
| socket | a ZMQ socket |
| endpoint | a ZMQ socket endpoint |
| MC | a message control, see ZMQ.MC() for details |
| option.name | an option name to the socket |
| option.value | an option value to the option name |

Details

`zmq.socket()` initials a ZMQ socket given a ZMQ context `ctx` and a socket type. See [ZMQ.ST\(\)](#) for the possible values of `type`. ZMQ defines several patterns for the socket type and utilize them to communicate in different ways including request-reply, publish-subscribe, pipeline, exclusive pair, and naive patterns.

`zmq.close()` destroys the ZMQ socket.

`zmq.bind()` binds the socket to a local endpoint and then accepts incoming connections on that endpoint. See `endpoint` next for details.

`zmq.connect()` connects the socket to a remote endpoint and then accepts outgoing connections on that endpoint. See `endpoint` next for details.

`endpoint` is a string consisting of a transport `://` followed by an address. The transport specifies the underlying protocol to use. The address specifies the transport-specific address to bind to. `pbdrZMQ/ZMQ` provides the following transports:

| Transport | Usage |
|------------------------|---|
| <code>tcp</code> | unicast transport using TCP |
| <code>ipc</code> | local inter-process communication transport |
| <code>inproc</code> | local in-process (inter-thread) communication transport |
| <code>pgm, epgm</code> | reliable multicast transport using PGM |

*** warning: `epgm` is not turned on by default in the `pbdrZMQ`'s internal ZeroMQ library.

*** warning: `ipc` is not supported in Windows system.

`zmq.setsockopt()` is to set/change socket options.

`zmq.getsockopt()` is to get socket options and returns `option.value`.

Value

`zmq.socket()` returns an R external pointer (`socket`) generated by ZMQ C API pointing to a socket if successful, otherwise returns an R NULL and sets `errno` to the error value, see ZeroMQ manual for details.

`zmq.close()` destroys the socket reference/pointer (`socket`) and returns 0 if successful, otherwise returns -1 and sets `errno` to the error value, see ZeroMQ manual for details.

`zmq.bind()` binds the socket to specific endpoint and returns 0 if successful, otherwise returns -1 and sets `errno` to the error value, see ZeroMQ manual for details.

`zmq.connect()` connects the socket to specific endpoint and returns 0 if successful, otherwise returns -1 and sets `errno` to the error value, see ZeroMQ manual for details.

`zmq.setsockopt()` sets/changes the socket option and returns 0 if successful, otherwise returns -1 and sets `errno` to the error value, see ZeroMQ manual for details.

`zmq.getsockopt()` returns the value of socket option, see ZeroMQ manual for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.ctx.new\(\)](#), [zmq.ctx.destroy\(\)](#).

Examples

```
## Not run:
### Using request-reply pattern.

### At the server, run next in background or the other windows.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
responder <- zmq.socket(context, ZMQ.ST()$REP)
zmq.bind(responder, "tcp://*:5555")
zmq.close(responder)
zmq.ctx.destroy(context)

### At a client, run next in foreground.
library(pbdZMQ, quietly = TRUE)

context <- zmq.ctx.new()
requester <- zmq.socket(context, ZMQ.ST()$REQ)
zmq.connect(requester, "tcp://localhost:5555")
zmq.close(requester)
zmq.ctx.destroy(context)

## End(Not run)
```

Transfer Functions for Files or Directories

Transfer Functions for Files or Directories

Description

High level functions calling `zmq.sendfile()` and `zmq.recvfile()` to zip, transfer, and unzip small files or directories contains small files.

Usage

```
zmq.senddir(  
  port,  
  infiles,  
  verbose = FALSE,  
  flags = ZMQ.SR()$BLOCK,  
  ctx = NULL,  
  socket = NULL  
)  
  
zmq.recvdir(  
  port,
```

```

    endpoint,
    outfile = NULL,
    exdir = NULL,
    verbose = FALSE,
    flags = ZMQ.SR()$BLOCK,
    ctx = NULL,
    socket = NULL
  )

```

Arguments

| | |
|----------|--|
| port | A valid tcp port to be passed to <code>zmq.sendfile()</code> and <code>zmq.recvfile()</code> . |
| infiles | The name (as a string) vector of the in files to be zipped and to be sent away. |
| verbose | Logical; determines if a progress bar should be shown. |
| flags | A flag for the method used by <code>zmq_sendfile</code> and <code>zmq_recvfile</code> |
| ctx | A ZMQ ctx. If NULL (default), the function will initial one at the beginning and destroy it after finishing file transfer. |
| socket | A ZMQ socket based on ctx. If NULL (default), the function will create one at the beginning and close it after finishing file transfer. |
| endpoint | A ZMQ socket endpoint to be passed to <code>zmq.sendfile()</code> and <code>zmq.recvfile()</code> . |
| outfile | The name (as a string) of the out file to be saved on the disk. If <code>outfile = NULL</code> and <code>exdir = NULL</code> , a tempfile will be used and the tempfile name will be returned. |
| exdir | The name (as a string) of the out directory to save the unzip files unzipped from the received outfile. |

Details

`zmq.senddir()` calls `zmq.senddir()`, and `zmq.recvdir()` calls `zmq.recvdir()`.

Value

`zmq.senddir()` and `zmq.recvdir()` return number of bytes (invisible) in the sent message if successful, otherwise returns -1 (invisible) and sets `errno` to the error value, see ZeroMQ manual for details. In addition, `zmq.recvdir()` returns a zipped file name in a list.

Author(s)

Wei-Chen Chen

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[zmq.sendfile\(\)](#), [zmq.recvfile\(\)](#).

Examples

```
## Not run:
### Run the sender and receiver code in separate R sessions.

### Receiver
library(pbdZMQ, quietly = TRUE)
zmq.recvdir(55555, "localhost", outfile = "./backup_2019.zip",
            verbose = TRUE)
### or unzip to exdir
# zmq.recvdir(55555, "localhost", exdir = "./backup_2019", verbose = TRUE)

### Sender
library(pbdZMQ, quietly = TRUE)
zmq.senddir(55555, c("./pbdZMQ/R", "./pbdZMQ/src"), verbose = TRUE)

## End(Not run)
```

Wrapper Functions for rzmq

All Wrapper Functions for rzmq

Description

Wrapper functions for backwards compatibility with rzmq. See vignette for examples.

Usage

```
send.socket(
  socket,
  data,
  send.more = FALSE,
  serialize = TRUE,
  serialversion = NULL
)

receive.socket(socket, unserialize = TRUE, dont.wait = FALSE)

init.context()

init.socket(context, socket.type)

bind.socket(socket, address)

connect.socket(socket, address)
```

Arguments

| | |
|-------------------------------------|---|
| <code>socket</code> | A ZMQ socket. |
| <code>data</code> | An R object. |
| <code>send.more</code> | Logical; will more messages be sent? |
| <code>serialize, unserialize</code> | Logical; determines if <code>serialize/unserialize</code> should be called on the sent/received data. |
| <code>serialversion</code> | NULL or numeric; the workspace format version to use when serializing. NULL specifies the current default version. The only other supported values are 2 and 3. |
| <code>dont.wait</code> | Logical; determines if reception is blocking. |
| <code>context</code> | A ZMQ context. |
| <code>socket.type</code> | The type of ZMQ socket as a string, of the form "ZMQ_type". Valid 'type' values are PAIR, PUB, SUB, REQ, REP, DEALER, PULL, PUSH, XPUB, XSUB, and STERAM. |
| <code>address</code> | A valid address. See details. |

Details

`send.socket()/receive.socket()` send/receive messages over a socket. These are simple wrappers around `zmq.msg.send()` and `zmq.msg.receive()`, respectively.

`init.context()` creates a new ZeroMQ context. A useful wrapper around `zmq.ctx.new()` which handles freeing memory for you, i.e. `zmq.ctx.destroy()` will automatically be called for you.

`init.socket()` creates a ZeroMQ socket; serves as a high-level binding for `zmq.socket()`, including handling freeing memory automatically. See also `ZMQ.ST()`.

`bind.socket()`: see `zmq.bind()`.

`connect.socket()`: see `zmq.connect()`

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

 ZMQ Control Environment

Sets of controls in pbdZMQ.

Description

These sets of controls are used to provide default values in this package.

Format

Objects contain several parameters for communicators and methods.

Details

The elements of `.pbd_env$ZMQ.ST` are default values for socket types as defined in ‘zmq.h’ including

| Elements | Value | Usage |
|----------|-------|--------------------|
| PAIR | 0L | socket type PAIR |
| PUB | 1L | socket type PUB |
| SUB | 2L | socket type SUB |
| REQ | 3L | socket type REQ |
| REP | 4L | socket type REP |
| DEALER | 5L | socket type DEALER |
| ROUTER | 6L | socket type ROUTER |
| PULL | 7L | socket type PULL |
| PUSH | 8L | socket type PUSH |
| XPUB | 9L | socket type XPUB |
| XSUB | 10L | socket type XSUB |
| STREAM | 11L | socket type STREAM |

The elements of `.pbd_env$ZMQ.SO` are default values for socket options as defined in ‘zmq.h’ including 60 different values, see `.pbd_env$ZMQ.SO` and ‘zmq.h’ for details.

The elements of `.pbd_env$ZMQ.SR` are default values for send/rcv options as defined in ‘zmq.h’ including

| Elements | Value | Usage |
|----------|-------|---|
| BLOCK | 0L | send/rcv option BLOCK |
| DONTWAIT | 1L | send/rcv option DONTWAIT |
| NOBLOCK | 1L | send/rcv option NOBLOCK |
| SNDMORE | 2L | send/rcv option SNDMORE (not supported) |

The elements of `.pbd_env$ZMQ.MC` are default values for warning and stop controls in R. These are not the ZeroMQ’s internal default values. They are defined as

| Elements | Value | Usage |
|------------------|-------|------------------|
| warning.at.error | TRUE | if warn at error |
| stop.at.error | TRUE | if stop at error |

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[.zmqopt_init\(\)](#).

ZMQ Control Functions *Sets of controls in pbdZMQ.*

Description

These sets of controls are used to provide default values in this package.

Usage

```
ZMQ.MC(warning.at.error = TRUE, stop.at.error = FALSE, check.eintr = FALSE)
```

```
ZMQ.PO(POLLIN = 1L, POLLOUT = 2L, POLLERR = 4L)
```

```
ZMQ.SR(BLOCK = 0L, DONTWAIT = 1L, NOBLOCK = 1L, SNDMORE = 2L)
```

```
ZMQ.SO(
  AFFINITY = 4L,
  IDENTITY = 5L,
  SUBSCRIBE = 6L,
  UNSUBSCRIBE = 7L,
  RATE = 8L,
  RECOVERY_IVL = 9L,
  SNDBUF = 11L,
  RCVBUF = 12L,
  RCVMORE = 13L,
  FD = 14L,
  EVENTS = 15L,
  TYPE = 16L,
  LINGER = 17L,
  RECONNECT_IVL = 18L,
```

BACKLOG = 19L,
RECONNECT_IVL_MAX = 21L,
MAXMSGSIZE = 22L,
SNDBWM = 23L,
RCVBWM = 24L,
MULTICAST_HOPS = 25L,
RCVTIMEO = 27L,
SNDTIMEO = 28L,
LAST_ENDPOINT = 32L,
ROUTER_MANDATORY = 33L,
TCP_KEEPALIVE = 34L,
TCP_KEEPALIVE_CNT = 35L,
TCP_KEEPALIVE_IDLE = 36L,
TCP_KEEPALIVE_INTVL = 37L,
TCP_ACCEPT_FILTER = 38L,
IMMEDIATE = 39L,
XPUB_VERBOSE = 40L,
ROUTER_RAW = 41L,
IPV6 = 42L,
MECHANISM = 43L,
PLAIN_SERVER = 44L,
PLAIN_USERNAME = 45L,
PLAIN_PASSWORD = 46L,
CURVE_SERVER = 47L,
CURVE_PUBLICKEY = 48L,
CURVE_SECRETKEY = 49L,
CURVE_SERVERKEY = 50L,
PROBE_ROUTER = 51L,
REQ_CORRELATE = 52L,
REQ_RELAXED = 53L,
CONFLATE = 54L,
ZAP_DOMAIN = 55L,
ROUTER_HANDOVER = 56L,
TOS = 57L,
IPC_FILTER_PID = 58L,
IPC_FILTER_UID = 59L,
IPC_FILTER_GID = 60L,
CONNECT_RID = 61L,
GSSAPI_SERVER = 62L,
GSSAPI_PRINCIPAL = 63L,
GSSAPI_SERVICE_PRINCIPAL = 64L,
GSSAPI_PLAINTEXT = 65L,
HANDSHAKE_IVL = 66L,
IDENTITY_FD = 67L,
SOCKS_PROXY = 68L,
XPUB_NODROP = 69L,
BLOCKY = 70L,
XPUB_MANUAL = 71L,

```
XPUB_WELCOME_MSG = 72L,  
STREAM_NOTIFY = 73L,  
INVERT_MATCHING = 74L,  
HEARTBEAT_IVL = 75L,  
HEARTBEAT_TTL = 76L,  
HEARTBEAT_TIMEOUT = 77L,  
XPUB_VERBOSE = 78L,  
CONNECT_TIMEOUT = 79L,  
TCP_MAXRT = 80L,  
THREAD_SAFE = 81L,  
MULTICAST_MAXTPDU = 84L,  
VMCI_BUFFER_SIZE = 85L,  
VMCI_BUFFER_MIN_SIZE = 86L,  
VMCI_BUFFER_MAX_SIZE = 87L,  
VMCI_CONNECT_TIMEOUT = 88L,  
USE_FD = 89L,  
GSSAPI_PRINCIPAL_NAME_TYPE = 90L,  
GSSAPI_SERVICE_PRINCIPAL_NAME_TYPE = 91L,  
BINDTODEVICE = 92L,  
ZAP_ENFORCE_DOMAIN = 93L,  
LOOPBACK_FASTPATH = 94L,  
METADATA = 95L,  
MULTICAST_LOOP = 96L,  
ROUTER_NOTIFY = 97L,  
XPUB_MANUAL_LAST_VALUE = 98L,  
SOCKS_USERNAME = 99L,  
SOCKS_PASSWORD = 100L,  
IN_BATCH_SIZE = 101L,  
OUT_BATCH_SIZE = 102L,  
WSS_KEY_PEM = 103L,  
WSS_CERT_PEM = 104L,  
WSS_TRUST_PEM = 105L,  
WSS_HOSTNAME = 106L,  
WSS_TRUST_SYSTEM = 107L,  
ONLY_FIRST_SUBSCRIBE = 108L,  
RECONNECT_STOP = 109L,  
HELLO_MSG = 110L,  
DISCONNECT_MSG = 111L,  
PRIORITY = 112L  
)
```

```
ZMQ.ST(  
PAIR = 0L,  
PUB = 1L,  
SUB = 2L,  
REQ = 3L,  
REP = 4L,  
DEALER = 5L,
```

```

ROUTER = 6L,
PULL = 7L,
PUSH = 8L,
XPUB = 9L,
XSUB = 10L,
STREAM = 11L
)

```

Arguments

warning.at.error, stop.at.error, check.eintr

Logical; if there is a messaging error, should there be an R warning/error, or check user interrupt events.

POLLIN, POLLOUT, POLLERR

ZMQ poll options; see zmq.h for details.

BLOCK, DONTWAIT, NOBLOCK, SNDMORE

ZMQ socket options; see zmq.h for details.

AFFINITY, IDENTITY, SUBSCRIBE, UNSUBSCRIBE, RATE, RECOVERY_IVL, SNDBUF, RCVBUF, RCVMORE, FD, EVENTS, TYPE, LINGER, RECONNECT_IVL, BACKLOG, RECONNECT_IVL_MAX, MAXMSGSIZE, SNDHWM, RCVHWM, MULTICAST_HOPS, RCVTIMEO, SNDTIMEO, LAST_ENDPOINT, ROUTER_MANDATORY, TCP_KEEPALIVE, TCP_KEEPALIVE_CNT, TCP_KEEPALIVE_IDLE, TCP_KEEPALIVE_INTVL, TCP_ACCEPT_FILTER, IMMEDIATE, XPUB_VERBOSE, ROUTER_RAW, IPV6, MECHANISM, PLAIN_SERVER, PLAIN_USERNAME, PLAIN_PASSWORD, CURVE_SERVER, CURVE_PUBLICKEY, CURVE_SECRETKEY, CURVE_SERVERKEY, PROBE_ROUTER, REQ_CORRELATE, REQ_RELAXED, CONFLATE, ZAP_DOMAIN, ROUTER_HANDOVER, TOS, IPC_FILTER_PID, IPC_FILTER_UID, IPC_FILTER_GID, CONNECT_RID, GSSAPI_SERVER, GSSAPI_PRINCIPAL, GSSAPI_SERVICE_PRINCIPAL, GSSAPI_PLAINTEXT, HANDSHAKE_IVL, IDENTITY_FD, SOCKS_PROXY, XPUB_NODROP, BLOCKY, XPUB_MANUAL, XPUB_WELCOME_MSG, STREAM_NOTIFY, INVERT_MATCHING, HEARTBEAT_IVL, HEARTBEAT_TTL, HEARTBEAT_TIMEOUT, XPUB_VERBOSE, CONNECT_TIMEOUT, TCP_MAXRT, THREAD_SAFE, MULTICAST_MAXTPDU, VMCI_BUFFER_SIZE, VMCI_BUFFER_MIN_SIZE, VMCI_BUFFER_MAX_SIZE, VMCI_CONNECT_TIMEOUT, USE_FD, GSSAPI_PRINCIPAL_NAMETYPE, GSSAPI_SERVICE_PRINCIPAL_NAMETYPE, BINDTODEVICE, ZAP_ENFORCE_DOMAIN, LOOPBACK_FASTPATH, METADATA, MULTICAST_LOOP, ROUTER_NOTIFY, XPUB_MANUAL_LAST_VALUE, SOCKS_USERNAME, SOCKS_PASSWORD, IN_BATCH_SIZE, OUT_BATCH_SIZE, WSS_KEY_PEM, WSS_CERT_PEM, WSS_TRUST_PEM, WSS_HOSTNAME, WSS_TRUST_SYSTEM, ONLY_FIRST_SUBSCRIBE, RECONNECT_STOP, HELLO_MSG, DISCONNECT_MSG, PRIORITY

ZMQ socket options; see zmq.h for details.

PAIR, PUB, SUB, REQ, REP, DEALER, ROUTER, PULL, PUSH, XPUB, XSUB, STREAM

ZMQ socket types; see zmq.h for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>

Programming with Big Data in R Website: <https://pbdr.org/>

See Also

[.pbd_env.](#)

ZMQ Flags

ZMQ Flags

Description

ZMQ Flags

Usage

```
get.zmq.ldflags(arch = "", package = "pbdZMQ")
```

```
get.zmq.cppflags(arch = "", package = "pbdZMQ")
```

```
test.load.zmq(arch = "", package = "pbdZMQ")
```

```
get.pbdZMQ.ldflags(arch = "", package = "pbdZMQ")
```

Arguments

arch " (default) for non-windows or `'/i386'` and `'/ix64'` for windows

package the pbdZMQ package

Details

`get.zmq.cppflags()` gets CFLAGS or CPPFLAGS

`get.zmq.ldflags()` gets LDFLAGS for libzmq.so, libzmq.dll, or libzmq.*.dylib

`get.pbdZMQ.ldflags()` gets LDFLAGS for pbdZMQ.so or pbdZMQ.dll

`test.load.zmq()` tests load libzmq and pbdZMQ shared libraries

Value

flags to compile and link with ZMQ.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com>.

References

ZeroMQ/4.1.0 API Reference: <https://libzmq.readthedocs.io/en/zeromq4-1/>
Programming with Big Data in R Website: <https://pbdr.org/>

Examples

```
## Not run:  
get.zmq.cppflags(arch = '/i386')  
get.zmq.ldflags(arch = '/x64')  
get.pbdZMQ.ldflags(arch = '/x64')  
test.load.zmq(arch = '/x64')  
  
## End(Not run)
```

Index

- * **compile**
 - Overwrite shpkg, [12](#)
 - ZMQ Flags, [32](#)
- * **global**
 - ZMQ Control Environment, [27](#)
 - ZMQ Control Functions, [28](#)
- * **package**
 - pbdZMQ-package, [2](#)
- * **programming**
 - Context Functions, [5](#)
 - File Transfer Functions, [6](#)
 - Initial Control Functions, [8](#)
 - Message Function, [10](#)
 - Poll Functions, [13](#)
 - Send Receive Functions, [16](#)
 - Send Receive Multiple Raw Buffers, [18](#)
 - Set Control Functions, [19](#)
 - Socket Functions, [21](#)
 - Transfer Functions for Files or Directories, [23](#)
- * **rmzmq**
 - Wrapper Functions for rmzmq, [25](#)
- * **variables**
 - ZMQ Control Environment, [27](#)
 - ZMQ Control Functions, [28](#)
- * **zmq**
 - C-like Wrapper Functions for ZeroMQ, [4](#)
 - .pbd_env, [8](#), [20](#), [32](#)
 - .pbd_env (ZMQ Control Environment), [27](#)
 - .zmqopt_get (Initial Control Functions), [8](#)
 - .zmqopt_init, [28](#)
 - .zmqopt_init (Initial Control Functions), [8](#)
 - .zmqopt_set (Initial Control Functions), [8](#)
- address, [3](#)
- bind.socket (Wrapper Functions for rmzmq), [25](#)
- C-like Wrapper Functions for ZeroMQ, [4](#)
- connect.socket (Wrapper Functions for rmzmq), [25](#)
- Context Functions, [5](#)
- File Transfer Functions, [6](#)
- get.pbdZMQ.ldflags (ZMQ Flags), [32](#)
- get.zmq.cppflags (ZMQ Flags), [32](#)
- get.zmq.ldflags (ZMQ Flags), [32](#)
- init.context (Wrapper Functions for rmzmq), [25](#)
- init.socket (Wrapper Functions for rmzmq), [25](#)
- Initial Control Functions, [8](#)
- ls, [9](#)
- Message Function, [10](#)
- Overwrite shpkg, [12](#)
- overwrite.shpkg.rpath (Overwrite shpkg), [12](#)
- pbd_opt (Set Control Functions), [19](#)
- pbdZMQ (pbzmq-package), [2](#)
- pbzmq-package, [2](#)
- Poll Functions, [13](#)
- random_open_port (random_port), [15](#)
- random_port, [15](#)
- receive.socket (Wrapper Functions for rmzmq), [25](#)
- Send Receive Functions, [16](#)
- Send Receive Multiple Raw Buffers, [18](#)
- send.socket (Wrapper Functions for rmzmq), [25](#)

- Set Control Functions, [19](#)
- Socket Functions, [21](#)
- test.load.zmq (ZMQ Flags), [32](#)
- Transfer Functions for Files or Directories, [23](#)
- Wrapper Functions for rzmq, [25](#)
- ZMQ Control Environment, [27](#)
- ZMQ Control Functions, [28](#)
- ZMQ Flags, [32](#)
- zmq.bind, [4](#), [5](#)
- zmq.bind (Socket Functions), [21](#)
- zmq.close, [5](#)
- zmq.close (Socket Functions), [21](#)
- zmq.connect, [5](#)
- zmq.connect (Socket Functions), [21](#)
- zmq.ctx.destroy, [22](#)
- zmq.ctx.destroy (Context Functions), [5](#)
- zmq.ctx.new, [3](#), [22](#)
- zmq.ctx.new (Context Functions), [5](#)
- zmq.disconnect (Socket Functions), [21](#)
- zmq.getsockopt (Socket Functions), [21](#)
- ZMQ.MC, [13](#), [21](#)
- ZMQ.MC (ZMQ Control Functions), [28](#)
- zmq.msg.recv, [7](#), [17](#), [19](#)
- zmq.msg.recv (Message Function), [10](#)
- zmq.msg.send, [7](#), [17](#), [19](#)
- zmq.msg.send (Message Function), [10](#)
- ZMQ.PO, [13](#)
- ZMQ.PO (ZMQ Control Functions), [28](#)
- zmq.poll (Poll Functions), [13](#)
- zmq.recv, [11](#), [14](#)
- zmq.recv (Send Receive Functions), [16](#)
- zmq.recv.multipart (Send Receive Multiple Raw Buffers), [18](#)
- zmq.recvdir (Transfer Functions for Files or Directories), [23](#)
- zmq.recvfile, [24](#)
- zmq.recvfile (File Transfer Functions), [6](#)
- zmq.send, [11](#), [14](#)
- zmq.send (Send Receive Functions), [16](#)
- zmq.send.multipart (Send Receive Multiple Raw Buffers), [18](#)
- zmq.senddir (Transfer Functions for Files or Directories), [23](#)
- zmq.sendfile, [24](#)
- zmq.sendfile (File Transfer Functions), [6](#)
- zmq.setsockopt (Socket Functions), [21](#)
- ZMQ.SO (ZMQ Control Functions), [28](#)
- zmq.socket, [3](#), [5](#)
- zmq.socket (Socket Functions), [21](#)
- ZMQ.SR, [16](#)
- ZMQ.SR (ZMQ Control Functions), [28](#)
- ZMQ.ST, [21](#)
- ZMQ.ST (ZMQ Control Functions), [28](#)