Package 'socketR'

September 15, 2026

Title POSIX Socket Programming for R

Version 1.0.5

Description Provides a small POSIX sockets interface to R, enabling direct network communication from R for applications that need low-level socket control or lightweight client and server connections. This package provides IPv4 and IPv6 support with 'TCP/UDP' protocols. It functional API with 'socket_create' and an R6 interface through 'Socket' object.

License Apache License (>= 2)

Encoding UTF-8

SystemRequirements Linux operating system with POSIX socket APIs and C++17 compiler

Imports R6

Suggests arrow, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://sassoftware.github.io/socketr/

BugReports https://github.com/sassoftware/socketr/issues

Config/roxygen2/version 8.1.0

Depends R (>= 4.6.0)

NeedsCompilation yes

Author Eduardo Hellas [aut, cre], SAS [cph, fnd]

Maintainer Eduardo Hellas <ehellas@gmail.com>

Repository CRAN

Date/Publication 2026-09-15 06:30:02 UTC

Contents

`socketR-package` *POSIX socket programming for R*

Description

External-pointer sockets implemented with base R's C API for POSIX TCP, UDP, IPv4, and IPv6 and Unix-domain socket operations, including bind, listen, accept, connect, send, receive, polling, shutdown, close, socket names, and typed options.

Details

Socket handles are RAII-managed external pointers and should still be closed explicitly with `socket_close()` when possible. Most system-call failures include the POSIX errno name and number.

Author(s)

Maintainer: Eduardo Hellas <ehellas@gmail.com>

Authors:

- Eduardo Hellas <ehellas@gmail.com>

Other contributors:

- SAS <support@sas.com> [copyright holder, funder]

See Also

Useful links:

- https://sassoftware.github.io/socketr/

close.Socket *Close an R6 Socket with the standard R connection API.*

Description

Close an R6 Socket with the standard R connection API.

Usage

```
## S3 method for class 'Socket'
close(con, ...)
```

Arguments

con	A Socket R6 object.
...	Ignored.

close.socketr_socket *Close a socketR handle with the standard R connection API.*

Description

Close a socketR handle with the standard R connection API.

Usage

```
## S3 method for class 'socketr_socket'
close(con, ...)
```

Arguments

con	A socketR socket handle.
...	Ignored.

Socket *R6 convenience wrapper for socketR handles.*

Description

The public methods mirror the functional API: `is_open()`, `fd()`, `info()`, `set_blocking()`, `bind()`, `listen()`, `accept()`, `connect()`, `send()`, `receive()`, `send_to()`, `receive_from()`, `shutdown()`, `close()`, `local_name()`, `peer_name()`, `poll()`, `get_option()`, `set_option()`, and `set_options()`. They delegate to the corresponding documented functions while retaining the handle as object state.

Use `Socket$new_auto()` to create a connected client with IPv4/IPv6 fallback, or `Socket$new_listener()` to create a listening server with IPv4/IPv6 fallback.

Value

An R6 `Socket` object.

Public fields

`handle` The underlying `socketr_socket` external pointer.

Active bindings

`address` Local address, when bound.

`port` Local port, when bound.

`peer_address` Connected peer address, when connected.

`peer_port` Connected peer port, when connected.

`family` Address family (`"inet"`, `"inet6"`, or `"unix"`).

`type` Socket type (`"stream"` or `"dgram"`).

`protocol` Numeric socket protocol.

`blocking` Whether the socket is in blocking mode.

`open` Whether the underlying socket is open.

Methods

Public methods:

- `Socket$new()`
- `Socket$new_auto()`
- `Socket$new_listener()`
- `Socket$is_open()`
- `Socket$fd()`
- `Socket$info()`
- `Socket$set_blocking()`
- `Socket$bind()`

- Socket$listen()
- Socket$accept()
- Socket$connect()
- Socket$as_connection()
- Socket$write()
- Socket$read()
- Socket$send()
- Socket$receive()
- Socket$send_to()
- Socket$receive_from()
- Socket$shutdown()
- Socket$close()
- Socket$local_name()
- Socket$peer_name()
- Socket$poll()
- Socket$get_option()
- Socket$set_options()
- Socket$set_option()
- Socket$clone()

`Socket$new()`: Create a socket wrapper or wrap an existing handle.

Usage:

```
Socket$new(
  domain = c("inet", "inet6", "unix"),
  type = c("stream", "dgram"),
  protocol = 0L,
  nonblocking = FALSE,
  cloexec = TRUE,
  handle = NULL
)
```

Arguments:

`domain` Address family passed to `socket_create()`.

`type` Socket type passed to `socket_create()`.

`protocol` Numeric protocol passed to `socket_create()`.

`nonblocking` Whether the new socket is nonblocking.

`cloexec` Whether the new socket is close-on-exec.

`handle` An existing socketR handle to wrap.

`Socket$new_auto()`: Create a connected client using IPv4/IPv6 endpoint fallback.

Usage:

```
Socket$new_auto(
  address,
  port = NULL,
  type = c("stream", "dgram"),
```

```
  protocol = 0L,
  nonblocking = FALSE,
  cloexec = TRUE,
  prefer = c("inet6", "inet")
)
```

Arguments:

`address` Hostname, IP address, or endpoint such as `"[::1]:8080"`.

`port` Optional port when it is not embedded in `address`.

`type` Socket type, `"stream"` (TCP) or `"dgram"` (UDP).

`protocol` Numeric protocol, usually zero.

`nonblocking` Whether the socket should be nonblocking.

`cloexec` Whether to set close-on-exec.

`prefer` Address families to try, in order.

Returns: A connected Socket R6 object.

`Socket$new_listener()`**:** Create a listener using IPv4/IPv6 endpoint fallback.

Usage:

```
Socket$new_listener(
  address = NULL,
  port = NULL,
  backlog = 128L,
  reuse_address = TRUE,
  cloexec = TRUE,
  prefer = c("inet6", "inet")
)
```

Arguments:

`address` Local hostname, IP address, endpoint, or `NULL` for wildcard.

`port` Optional port when it is not embedded in `address`.

`backlog` Maximum pending connection queue length.

`reuse_address` Whether to set `SO_REUSEADDR`.

`cloexec` Whether to set close-on-exec.

`prefer` Address families to try, in order.

Returns: A listening Socket R6 object.

`Socket$is_open()`**:** Test whether the wrapped socket is open.

Usage:

```
Socket$is_open()
```

`Socket$fd()`**:** Return the wrapped socket file descriptor.

Usage:

```
Socket$fd()
```

`Socket$info()`**:** Return metadata for the wrapped socket.

Usage:

```
Socket$info()
```

`Socket$set_blocking()`: Set blocking mode.

Usage:

```
Socket$set_blocking(blocking = TRUE)
```

Arguments:

`blocking` Whether operations should block.

`Socket$bind()`: Bind the wrapped socket.

Usage:

```
Socket$bind(address = NULL, port = NULL)
```

Arguments:

`address` Local hostname, IP address, or Unix path.

`port` Local port, or `NULL` for Unix sockets.

`Socket$listen()`: Listen for stream connections.

Usage:

```
Socket$listen(backlog = 128L)
```

Arguments:

`backlog` Maximum pending connection queue length.

`Socket$accept()`: Accept a pending connection.

Usage:

```
Socket$accept(nonblocking = NULL)
```

Arguments:

`nonblocking` Whether the accepted socket should be nonblocking.

`Socket$connect()`: Connect the wrapped socket.

Usage:

```
Socket$connect(address, port = NULL)
```

Arguments:

`address` Remote hostname, IP address, or Unix path.

`port` Remote port, or `NULL` for Unix sockets.

`Socket$as_connection()`: Return the wrapped socket as an R connection.

Usage:

```
Socket$as_connection()
```

`Socket$write()`: Write bytes to the wrapped socket.

Usage:

```
Socket$write(object, flags = 0L)
```

Arguments:

`object` Raw bytes or a character scalar.
`flags` Native `send()` flags.

`Socket$read()`: Read bytes from the wrapped socket.

Usage:

```
Socket$read(n = 4096L, flags = 0L)
```

Arguments:

`n` Maximum number of bytes.
`flags` Native `recv()` flags.

`Socket$send()`: Send bytes on the wrapped socket.

Usage:

```
Socket$send(data, flags = 0L)
```

Arguments:

`data` Raw bytes or a character scalar.
`flags` Native `send()` flags.

`Socket$receive()`: Receive bytes from the wrapped socket.

Usage:

```
Socket$receive(n = 4096L, flags = 0L)
```

Arguments:

`n` Maximum number of bytes.
`flags` Native `recv()` flags.

`Socket$send_to()`: Send a datagram.

Usage:

```
Socket$send_to(data, address, port = NULL, flags = 0L)
```

Arguments:

`data` Raw bytes or a character scalar.
`address` Destination hostname or IP address.
`port` Destination port.
`flags` Native `sendto()` flags.

`Socket$receive_from()`: Receive a datagram.

Usage:

```
Socket$receive_from(n = 4096L, flags = 0L)
```

Arguments:

`n` Maximum datagram payload size.
`flags` Native `recvfrom()` flags.

`Socket$shutdown()`: Shut down part of the connection.

Usage:

```
Socket$shutdown(how = c("both", "read", "write"))
```

Arguments:

`how` One of `"read"`, `"write"`, or `"both"`.

`Socket$close()`: Close the wrapped socket.

Usage:

```
Socket$close()
```

`Socket$local_name()`: Return the local socket name.

Usage:

```
Socket$local_name()
```

`Socket$peer_name()`: Return the peer socket name.

Usage:

```
Socket$peer_name()
```

`Socket$poll()`: Poll the wrapped socket.

Usage:

```
Socket$poll(events = "read", timeout_ms = 60000L)
```

Arguments:

`events` Requested readiness events.

`timeout_ms` Timeout in milliseconds.

`Socket$get_option()`: Get a socket option.

Usage:

```
Socket$get_option(
  level = "socket",
  option,
  type = c("int", "logical", "timeval", "linger", "raw"),
  size = 256L
)
```

Arguments:

`level` Option level.

`option` Option name or numeric constant.

`type` Return type.

`size` Maximum raw option size.

`Socket$set_options()`: Set multiple socket options.

Usage:

```
Socket$set_options(options)
```

Arguments:

`options` Option specifications.

`Socket$set_option()`: Set a socket option.

Usage:

```
Socket$set_option(
  level = "socket",
  option,
  value,
  type = c("int", "logical", "timeval", "linger", "raw")
)
```

Arguments:

`level` Option level.

`option` Option name or numeric constant.

`value` Option value.

`type` Value type.

`Socket$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Socket$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`socket_connections` *Adapt sockets and R connections for byte I/O.*

Description

Adapt a socket handle to an R connection. The returned connection can be passed to base R functions such as `readBin()`, `writeBin()`, `readLines()`, and `writeLines()`. Closing the connection does not close the underlying socket handle; call `socket_close()` explicitly when the handle is no longer needed.

Uses R's connection dispatch, so this works with `socket_connection()` and other readable R connections.

Uses R's connection dispatch, so this works with `socket_connection()` and other writable R connections.

Usage

```
socket_connection(socket, close_socket = FALSE)

socket_connection_read(connection, n = 4096L)

socket_connection_write(connection, data)
```

Arguments

socket	A socketR socket handle.
close_socket	Whether closing the adapter should also close the underlying socket. The default FALSE preserves adapter-only close behavior.
connection	An open R connection.
n	Maximum number of bytes to read.
data	A raw vector.

Value

An R connection object.

A raw vector, possibly shorter than n at end-of-file.

Number of bytes written.

Examples

```
s <- socket_create()
con <- socket_connection(s)
close(con)
socket_close(s)
```

---

socket_connect_auto *Connect using the first available IPv6 or IPv4 endpoint.*

Description

Resolves address for both requested address families, tries endpoints in prefer order, and creates a socket matching the endpoint that succeeds. This is the address-aware alternative to an "auto" socket domain. It supports IPv4 and IPv6 Internet sockets only; Unix-domain filesystem paths are not resolved by this helper. Use type = "dgram" for UDP or the default type = "stream" for TCP.

Usage

```
socket_connect_auto(
  address,
  port = NULL,
  type = c("stream", "dgram"),
  protocol = 0L,
  nonblocking = FALSE,
  cloexec = TRUE,
  prefer = c("inet6", "inet")
)
```

Arguments

address	A hostname or numeric IP address.
port	A port between 0 and 65535, or NULL when embedded in a bracketed endpoint such as "[::1]:12345".
type	Socket type: "stream" or "dgram".
protocol	Numeric protocol, usually zero.
nonblocking	Whether the socket should be nonblocking.
cloexec	Whether to set close-on-exec.
prefer	Address families to try, in order.

Value

A connected socketR handle.

Examples

```
if (interactive()) {
  client <- socket_connect_auto("localhost", 80L)
  socket_close(client)
}
```

---

socket_datagrams *Datagram socket I/O.*

Description

Send bytes to a datagram destination.

Usage

```
socket_send_to(socket, data, address, port = NULL, flags = 0L)

socket_receive_from(socket, n = 4096L, flags = 0L)
```

Arguments

socket	A datagram socket handle.
data	A raw vector or character scalar.
address	Destination hostname, IP address, or Unix-domain path.
port	Destination port; NULL for Unix sockets.
flags	Native recvfrom() flags.
n	Maximum datagram payload size.

Value

Number of bytes sent, or NA_integer_ if I/O would block.

A list with data and address, or NULL if I/O would block.

Examples

```
if (interactive()) {
  receiver <- socket_create(type = "dgram")
  sender <- socket_create(type = "dgram")
  socket_bind(receiver, "127.0.0.1", 0L)
  socket_send_to(sender, "hello", "127.0.0.1",
                 socket_local_name(receiver)$port)
  socket_receive_from(receiver)
  socket_close(sender); socket_close(receiver)
}
```

---

socket_fd *Return the native file descriptor, or NA for a closed socket.*

Description

Return the local socket name.

Usage

```
socket_fd(socket)

socket_info(socket)

socket_local_name(socket)

socket_peer_name(socket)
```

Arguments

socket	A connected socket handle.

Value

An integer descriptor or NA_integer_.

A list containing socket metadata.

Address metadata as a list.

Address metadata as a list.

Examples

```
s <- socket_create()
socket_local_name(s)
socket_close(s)
```

`socket_get_option` *Get a socket option.*

Description

Get a socket option.

Usage

```
socket_get_option(
  socket,
  level = "socket",
  option,
  type = c("int", "logical", "timeval", "linger", "raw"),
  size = 256L
)
```

Arguments

`socket`	A socketR socket handle.
`level`	Option level such as `"socket"` or `"tcp"`.
`option`	Option name or native numeric constant.
`type`	Return type: `"int"`, `"logical"`, `"timeval"`, `"linger"`, or `"raw"`.
`size`	Maximum raw option size.

Value

The option value.

socket_io *Stream socket I/O.*

Description

Write bytes to a connected socket. This is a connection-style alias for socket_send(). It returns the number of bytes written, which may be less than the input length for a nonblocking socket or a large payload.

This is a connection-style alias for socket_receive(). Blocking sockets wait until at least one byte is available or the peer closes the connection.

Usage

```
socket_write(socket, object, flags = 0L)

socket_send(socket, data, flags = 0L)

socket_read(socket, n = 4096L, flags = 0L)

socket_receive(socket, n = 4096L, flags = 0L)
```

Arguments

socket	A connected socket handle.
object	A raw vector or character scalar.
flags	Native recv() flags.
data	A raw vector or character scalar.
n	Maximum number of bytes to receive.

Value

Number of bytes written, or NA_integer_ if I/O would block.

Number of bytes sent, or NA_integer_ if nonblocking I/O would block.

A raw vector, or NULL if a nonblocking read would block.

A raw vector, or NULL if nonblocking I/O would block.

Examples

```
if (interactive()) {
  server <- socket_create()
  client <- socket_create()
  socket_bind(server, "127.0.0.1", 0L)
  socket_listen(server)
  socket_connect(client, "127.0.0.1", socket_local_name(server)$port)
  peer <- socket_accept(server)
  socket_write(client, "hello")
```

```
  socket_read(peer, 5L)
  socket_close(peer); socket_close(client); socket_close(server)
}
```

`socket_lifecycle` *Manage a socket handle lifecycle.*

Description

Manage a socket handle lifecycle.

Test whether a socket handle is open.

Shut down part of a full-duplex socket.

Usage

```
socket_close(socket)

socket_is_open(socket)

socket_shutdown(socket, how = c("both", "read", "write"))
```

Arguments

`socket`	A socketR socket handle.
`how`	One of `"read"`, `"write"`, or `"both"`.

Value

Invisibly, `TRUE`.

A logical scalar.

Invisibly, the socket handle.

Examples

```
s <- socket_create()
socket_close(s)
```

`socket_listen_auto` *Listen on the first available IPv6 or IPv4 address family.*

Description

Creates a stream listener, tries bind candidates in `prefer` order, and returns the first listener that binds and listens successfully. When `address` is `NULL`, the family wildcard (`"::"` or `"0.0.0.0"`) is used. This helper supports IPv4 and IPv6 stream sockets only. Unix-domain sockets require an explicit filesystem path with `socket_create()` and `socket_bind()`.

Usage

```
socket_listen_auto(
  address = NULL,
  port = NULL,
  backlog = 128L,
  reuse_address = TRUE,
  cloexec = TRUE,
  prefer = c("inet6", "inet")
)
```

Arguments

`address`	A local hostname or IP address, or `NULL` for a wildcard.
`port`	A port between 0 and 65535, or `NULL` when embedded in a bracketed endpoint such as `"[::1]:12345"`.
`backlog`	Maximum pending connection queue length.
`reuse_address`	Whether to set `SO_REUSEADDR` before binding.
`cloexec`	Whether to set close-on-exec.
`prefer`	Address families to try, in order.

Value

A listening socketR handle.

Examples

```
listener <- socket_listen_auto(port = 0L, prefer = "inet")
socket_local_name(listener)
socket_close(listener)
```

socket_options *Common socket option helpers.*

Description

These helpers read or set frequently used socket options. With the value argument omitted, the current option value is returned; otherwise the option is updated and the socket handle is returned invisibly. The socket_linger() helper uses on and seconds to configure SO_LINGER.

Usage

```
socket_reuse_address(socket, value)

socket_keep_alive(socket, value)

socket_broadcast(socket, value)

socket_no_delay(socket, value)

socket_quick_ack(socket, value)

socket_receive_buffer_size(socket, value)

socket_send_buffer_size(socket, value)

socket_receive_timeout(socket, value)

socket_send_timeout(socket, value)

socket_linger(socket, on, seconds = 0L)
```

Arguments

socket	A socketR socket handle.
value	Optional option value to set. Its type depends on the helper: logical, integer, or numeric seconds.
on	Optional logical linger enable flag.
seconds	Linger duration in seconds.

Value

The option value when reading; invisibly, the socket when setting.

Examples

```
s <- socket_create()
socket_reuse_address(s, TRUE)
socket_reuse_address(s)
socket_close(s)
```

`socket_readiness` *Control socket blocking and readiness.*

Description

Control socket blocking and readiness.

Poll sockets for readiness.

Usage

```
socket_set_blocking(socket, blocking = TRUE)

socket_poll(sockets, events = "read", timeout_ms = 60000L)
```

Arguments

`socket`	A socketR socket handle.
`blocking`	Whether operations should block.
`sockets`	A socket handle or list of handles.
`events`	Requested `"read"`, `"write"`, or combined readiness.
`timeout_ms`	Timeout in milliseconds; use `-1` to wait indefinitely.

Value

Invisibly, the socket handle.

A data frame with readiness flags and native revents values.

Examples

```
s <- socket_create()
socket_set_blocking(s, FALSE)
socket_close(s)


s <- socket_create()
socket_poll(s, "read", timeout_ms = 0L)
socket_close(s)
```

socket_resolve *Resolve a hostname or address into socket endpoints.*

Description

Resolve a hostname or address into socket endpoints.

Usage

```
socket_resolve(address, domain = c("inet", "inet6"), port = 0L)
```

Arguments

address	A hostname or numeric IP address.
domain	Address family, such as "inet" or "inet6".
port	Numeric service port between 0 and 65535.

Value

A list of resolved endpoint metadata.

socket_setup *Create and configure POSIX sockets.*

Description

Create a POSIX socket.

Usage

```
socket_create(
  domain = c("inet", "inet6", "unix"),
  type = c("stream", "dgram"),
  protocol = 0L,
  nonblocking = FALSE,
  cloexec = TRUE
)

socket_bind(socket, address = NULL, port = NULL)

socket_listen(socket, backlog = 128L)

socket_accept(socket, nonblocking = NULL)

socket_connect(socket, address, port = NULL)
```

Arguments

domain	Address family: "inet", "inet6", or "unix".
type	Socket type: "stream" or "dgram".
protocol	Numeric protocol, usually zero.
nonblocking	Whether the accepted socket should be nonblocking; NULL inherits the listener mode.
cloexec	Whether to set close-on-exec.
socket	A socketR socket handle.
address	A hostname, IP address, or Unix-domain path.
port	A port between 0 and 65535; NULL for Unix sockets.
backlog	Maximum pending connection queue length.

Value

A socketR external-pointer handle.

Invisibly, the socket handle.

Invisibly, the socket handle.

A socket handle, or NULL when no connection is pending.

TRUE when connected, or FALSE when connection is in progress.

Examples

```
s <- socket_create()
socket_close(s)
```

---

socket_set_option *Set a socket option.*

Description

Set a socket option.

Usage

```
socket_set_option(
  socket,
  level = "socket",
  option,
  value,
  type = c("int", "logical", "timeval", "linger", "raw")
)
```

Arguments

socket	A socketR socket handle.
level	Option level such as "socket" or "tcp".
option	Option name or native numeric constant.
value	Value matching type.
type	Value type: "int", "logical", "timeval", "linger", or "raw".

Value

Invisibly, the socket handle.

---

socket_set_options *Set multiple socket options in order.*

Description

Each element of options must be a list containing option and value, with optional level and type entries. Options are applied in list order. If a native option fails, the error includes the zero-based count of options already applied in its applied field.

Usage

```
socket_set_options(socket, options)
```

Arguments

socket	A socketR socket handle.
options	A non-empty list of option specifications.

Value

The socket handle, invisibly.

Index