socat

CONTENTS

NAME

socat - Multipurpose relay (SOcket CAT)

SYNOPSIS

```
socat [options] <address> <address>
socat -V
socat -h[h[h]] | -?[?[?]]
filan
procan
```

DESCRIPTION

Socat is a command line based utility that establishes two bidirectional byte streams and transfers data between them. Because the streams can be constructed from a large set of different types of data sinks and sources (see address types), and because lots of address options may be applied to the streams, socat can be used for many different purposes.

Filan is a utility that prints information about its active file descriptors to stdout. It has been written for debugging **socat**, but might be useful for other purposes too. Use the -h option to find more infos.

Procan is a utility that prints information about process parameters to stdout. It has been written to better understand some UNIX process properties and for debugging **socat**, but might be useful for other purposes too.

The life cycle of a **socat** instance typically consists of four phases.

In the *init* phase, the command line options are parsed and logging is initialized.

During the *open* phase, **socat** opens the first address and afterwards the second address. These steps are usually blocking; thus, especially for complex address types like socks, connection requests or authentication dialogs must be completed before the next step is started.

In the *transfer* phase, **socat** watches both streams' read and write file descriptors via `select()` , and, when data is available on one side *and* can be written to the other side, socat reads it, performs newline character conversions if required, and writes the data to the write file descriptor of the other stream, then continues waiting for more data in both directions.

When one of the streams effectively reaches EOF, the *closing* phase begins. **Socat** transfers the EOF condition to the other stream, i.e. tries to shutdown only its write stream, giving it a chance to terminate gracefully. For a defined time **socat** continues to transfer data in the other direction, but then closes all remaining channels and terminates.

OPTIONS

Socat provides some command line options that modify the behaviour of the program. They have nothing to do with so called address options that are used as parts of address specifications.

`-V`
: Print version and available feature information to stdout, and exit.

`-h | -?`
: Print a help text to stdout describing command line options and available address types, and exit.

`-hh | -??`
: Like -h, plus a list of the short names of all available address options. Some options are platform dependend, so this output is helpful for checking the particular implementation.

`-hhh | -???`
: Like -hh, plus a list of all available address option names.

`-d`
: Without this option, only fatal and error messages are generated; applying this option also prints warning messages. See DIAGNOSTICS for more information.

`-d -d`
: Prints fatal, error, warning, and notice messages.

`-d -d -d`
: Prints fatal, error, warning, notice, and info messages.

`-d -d -d -d`
: Prints fatal, error, warning, notice, info, and debug messages.

`-D`
: Logs information about file descriptors before starting the transfer phase.

`-ly[<facility>]`
: Writes messages to syslog instead of stderr; severity as defined with -d option. With optional <facility>, the syslog type can be selected, default is "daemon". Third party libraries might not obey this option.

`-lf <logfile>`
: Writes messages to <logfile> [filename] instead of stderr. Some third party libraries, in particular libwrap, might not obey this option.

`-ls`
: Writes messages to stderr (this is the default). Some third party libraries might not obey this option, in particular libwrap appears to only log to syslog.

`-lp<progname>`
: Overrides the program name printed in error messages and used for constructing environment variable names.

`-lu`

Extends the timestamp of error messages to microsecond resolution. Does not work when logging to syslog.

`-lm[<facility>]`
Mixed log mode. During startup messages are printed to stderr; when **socat** starts the transfer phase loop or daemon mode (i.e. after opening all streams and before starting data transfer, or, with listening sockets with fork option, before the first accept call), it switches logging to syslog. With optional <facility>, the syslog type can be selected, default is "daemon".

`-lh`
Adds hostname to log messages. Uses the value from environment variable HOSTNAME or the value retrieved with `uname()` if HOSTNAME is not set.

`-v`
Writes the transferred data not only to their target streams, but also to stderr. The output format is text with some conversions for readability, and prefixed with "> " or "< " indicating flow directions.

`-x`
Writes the transferred data not only to their target streams, but also to stderr. The output format is hexadecimal, prefixed with "> " or "< " indicating flow directions. Can be combined with `-v` .

`-r <file>`
Dumps the raw (binary) data flowing from left to right address to the given file.

`-R <file>`
Dumps the raw (binary) data flowing from right to left address to the given file.

`-b<size>`
Sets the data transfer block <size> [size_t]. At most <size> bytes are transferred per step. Default is 8192 bytes.

`-s`
By default, **socat** terminates when an error occurred to prevent the process from running when some option could not be applied. With this option, **socat** is sloppy with errors and tries to continue. Even with this option, socat will exit on fatals, and will abort connection attempts when security checks failed.

`-t<timeout>`
When one channel has reached EOF, the write part of the other channel is shut down. Then, **socat** waits <timeout> [timeval] seconds before terminating. Default is 0.5 seconds. This timeout only applies to addresses where write and read part can be closed independently. When during the timeout interval the read part gives EOF, socat terminates without awaiting the timeout.

`-T<timeout>`
Total inactivity timeout: when socat is already in the transfer loop and nothing has happened for <timeout> [timeval] seconds (no data arrived, no interrupt occurred...) then it terminates. Useful with protocols like UDP that cannot transfer EOF.

`-u`
Uses unidirectional mode. The first address is only used for reading, and the second address is only used for writing (example).

`-U`
Uses unidirectional mode in reverse direction. The first address is only used for writing, and the second address is only used for reading.

`-g`
During address option parsing, don't check if the option is considered useful in the given address environment. Use it if you want to force, e.g., appliance of a socket option to a serial device.

`-L<lockfile>`
If lockfile exists, exits with error. If lockfile does not exist, creates it and continues, unlinks lockfile on exit.

`-W<lockfile>`

If lockfile exists, waits until it disappears. When lockfile does not exist, creates it and continues, unlinks lockfile on exit.

`-4`

Use IP version 4 in case that the addresses do not implicitly or explicitly specify a version; this is the default.

`-6`

Use IP version 6 in case that the addresses do not implicitly or explicitly specify a version.

ADDRESS SPECIFICATIONS

With the address command line arguments, the user gives **socat** instructions and the necessary information for establishing the byte streams.

An address specification usually consists of an address type keyword, zero or more required address parameters separated by ':' from the keyword and from each other, and zero or more address options separated by ','.

The keyword specifies the address type (e.g., TCP4, OPEN, EXEC). For some keywords there exist synonyms ('-' for STDIO, TCP for TCP4). Keywords are case insensitive. For a few special address types, the keyword may be omitted: Address specifications starting with a number are assumed to be FD (raw file descriptor) addresses; if a '/' is found before the first ':' or ',', GOPEN (generic file open) is assumed.

The required number and type of address parameters depend on the address type. E.g., TCP4 requires a server specification (name or address), and a port specification (number or service name).

Zero or more address options may be given with each address. They influence the address in some ways. Options consist of an option keyword or an option keyword and a value, separated by '='. Option keywords are case insensitive. For filtering the options that are useful with an address type, each option is member of one option group. For each address type there is a set of option groups allowed. Only options belonging to one of these address groups may be used (except with option -g).

Address specifications following the above schema are also called *single* address specifications. Two single addresses can be combined with "!!" to form a *dual* type address for one channel. Here, the first address is used by **socat** for reading data, and the second address for writing data. There is no way to specify an option only once for being applied to both single addresses.

Usually, addresses are opened in read/write mode. When an address is part of a dual address specification, or when option -u or -U is used, an address might be used only for reading or for writing. Considering this is important with some address types.

With socat version 1.5.0 and higher, the lexical analysis tries to handle quotes and parenthesis meaningfully and allows escaping of special characters. If one of the characters ({ [' is found, the corresponding closing character -) }] ' - is looked for; they may also be nested. Within these constructs, socats special characters and strings : , !! are not handled specially. All those characters and strings can be escaped with \ or within ""

ADDRESS TYPES

This section describes the available address types with their keywords, parameters, and semantics.

`CREATE:<filename>`

Opens <filename> with `creat()` and uses the file descriptor for writing. This address type requires write-only context, because a file opened with `creat` cannot be read from.
Flags like O_LARGEFILE cannot be applied. If you need them use OPEN with options create,create.
<filename> must be a valid existing or not existing path. If <filename> is a named pipe, `creat()` might block; if <filename> refers to a socket, this is an error.
Option groups: FD,REG,NAMED

Useful options: mode, user, group, unlink-early, unlink-late, append
See also: OPEN, GOPEN

`EXEC:<command-line>`
Forks a sub process that establishes communication with its parent process and invokes the specified program with `execvp()` . <command-line> is a simple command with arguments separated by single spaces. If the program name contains a '/', the part after the last '/' is taken as ARGV[0]. If the program name is a relative path, the `execvp()` semantics for finding the program via `$PATH` apply. After successful program start, **socat** writes data to stdin of the process and reads from its stdout using a UNIX domain socket generated by `socketpair()` per default. (example)
Option groups: FD,SOCKET,EXEC,FORK,TERMIOS
Useful options: path, fdin, fdout, chroot, su, su-d, nofork, pty, stderr, ctty, setsid, pipes, login, sigint, sigquit
See also: SYSTEM

`FD:<fdnum>`
Uses the file descriptor <fdnum>. It must already exist as valid UN*X file descriptor.
Option groups: FD (TERMIOS,REG,SOCKET)
See also: STDIO, STDIN, STDOUT, STDERR

`GOPEN:<filename>`
(Generic open) This address type tries to handle any file system entry except directories usefully. <filename> may be a relative or absolute path. If it already exists, its type is checked. In case of a UNIX domain socket, **socat** connects; if connecting fails, **socat** assumes a datagram socket and uses `sendto()` calls. If the entry is not a socket, **socat** opens it applying the `O_APPEND` flag. If it does not exist, it is opened with flag `O_CREAT` as a regular file (example).
Option groups: FD,REG,SOCKET,NAMED,OPEN
See also: OPEN, CREATE, UNIX-CONNECT

`IP-SENDTO:<host>:<protocol>`
Opens a raw IP socket. Depending on host specification or option pf, IP protocol version 4 or 6 is used. It uses <protocol> to send packets to <host> [IP address] and receives packets from host, ignores packets from other hosts. Protocol 255 uses the raw socket with the IP header being part of the data.
Option groups: FD,SOCKET,IP4,IP6
Useful options: pf, ttl
See also: IP4-SENDTO, IP6-SENDTO, IP-RECVFROM, IP-RECV, UDP-SENDTO, UNIX-SENDTO

`INTERFACE:<interface>`
Communicates with a network connected on an interface using raw packets including link level data. <interface> is the name of the network interface. Currently only available on Linux. Option groups: FD,SOCKET
Useful options: pf, type
See also: ip-recv

`IP4-SENDTO:<host>:<protocol>`
Like IP-SENDTO, but always uses IPv4.
Option groups: FD,SOCKET,IP4

`IP6-SENDTO:<host>:<protocol>`
Like IP-SENDTO, but always uses IPv6.
Option groups: FD,SOCKET,IP6

`IP-DATAGRAM:<address>:<protocol>`
Sends outgoing data to the specified address which may in particular be a broadcast or multicast address. Packets arriving on the local socket are checked if their source addresses match RANGE or TCPWRAP options. This address type can for example be used for implementing symmetric or asymmetric broadcast or multicast communications.
Option groups: FD, SOCKET, IP4, IP6, RANGE
Useful options: bind, range, tcpwrap, broadcast, ip-multicast-loop, ip-multicast-ttl, ip-multicast-if, ip-add-

membership, ip-add-source-membership, ttl, tos, pf
See also: IP4-DATAGRAM, IP6-DATAGRAM, IP-SENDTO, IP-RECVFROM, IP-RECV, UDP-DATAGRAM

`IP4-DATAGRAM:<host>:<protocol>`
Like IP-DATAGRAM, but always uses IPv4. (example)
Option groups: FD,SOCKET,IP4,RANGE

`IP6-DATAGRAM:<host>:<protocol>`
Like IP-DATAGRAM, but always uses IPv6. Please note that IPv6 does not know broadcasts.
Option groups: FD,SOCKET,IP6,RANGE

`IP-RECVFROM:<protocol>`
Opens a raw IP socket of <protocol>. Depending on option pf, IP protocol version 4 or 6 is used. It receives one packet from an unspecified peer and may send one or more answer packets to that peer. This mode is particularly useful with fork option where each arriving packet - from arbitrary peers - is handled by its own sub process. This allows a behaviour similar to typical UDP based servers like ntpd or named.
Please note that the reply packets might be fetched as incoming traffic when sender and receiver IP address are identical because there is no port number to distinguish the sockets.
This address works well with IP-SENDTO address peers (see above). Protocol 255 uses the raw socket with the IP header being part of the data.
See the note about RECVFROM addresses.
Option groups: FD,SOCKET,IP4,IP6,CHILD,RANGE
Useful options: pf, fork, range, ttl, broadcast
See also: IP4-RECVFROM, IP6-RECVFROM, IP-SENDTO, IP-RECV, UDP-RECVFROM, UNIX-RECVFROM

`IP4-RECVFROM:<protocol>`
Like IP-RECVFROM, but always uses IPv4.
Option groups: FD,SOCKET,IP4,CHILD,RANGE

`IP6-RECVFROM:<protocol>`
Like IP-RECVFROM, but always uses IPv6.
Option groups: FD,SOCKET,IP6,CHILD,RANGE

`IP-RECV:<protocol>`
Opens a raw IP socket of <protocol>. Depending on option pf, IP protocol version 4 or 6 is used. It receives packets from multiple unspecified peers and merges the data. No replies are possible. It can be, e.g., addressed by socat IP-SENDTO address peers. Protocol 255 uses the raw socket with the IP header being part of the data.
Option groups: FD,SOCKET,IP4,IP6,RANGE
Useful options: pf, range
See also: IP4-RECV, IP6-RECV, IP-SENDTO, IP-RECVFROM, UDP-RECV, UNIX-RECV

`IP4-RECV:<protocol>`
Like IP-RECV, but always uses IPv4.
Option groups: FD,SOCKET,IP4,RANGE

`IP6-RECV:<protocol>`
Like IP-RECV, but always uses IPv6.
Option groups: FD,SOCKET,IP6,RANGE

`OPEN:<filename>`
Opens <filename> using the `open()` system call (example). This operation fails on UNIX domain sockets.
Note: This address type is rarely useful in bidirectional mode.
Option groups: FD,REG,NAMED,OPEN
Useful options: creat, excl, noatime, nofollow, append, rdonly, wronly, lock, readbytes, ignoreeof
See also: CREATE, GOPEN, UNIX-CONNECT

OPENSSL:<host>:<port>

Tries to establish a SSL connection to <port> [TCP service] on <host> [IP address] using TCP/IP version 4 or 6 depending on address specification, name resolution, or option pf.
NOTE: Up to version 1.7.2.4 the server certificate was only checked for validity against the system certificate store or cafile or capath, but not for match with the server's name or its IP address. Since version 1.7.3.0 socat checks the peer certificate for match with the <host> parameter or the value of the openssl-commonname option. Socat tries to match it against the certificates subject commonName, and the certificates extension subjectAltName DNS names. Wildcards in the certificate are supported.
Option groups: FD,SOCKET,IP4,IP6,TCP,OPENSSL,RETRY
Useful options: cipher, verify, commonname, cafile, capath, certificate, key, compress, bind, pf, connect-timeout, sourceport, retry
See also: OPENSSL-LISTEN, TCP

OPENSSL-LISTEN:<port>

Listens on tcp <port> [TCP service]. The IP version is 4 or the one specified with pf. When a connection is accepted, this address behaves as SSL server.
Note: You probably want to use the certificate option with this address.
NOTE: The client certificate is only checked for validity against cafile or capath, but not for match with the client's name or its IP address!
Option groups: FD,SOCKET,IP4,IP6,TCP,LISTEN,OPENSSL,CHILD,RANGE,RETRY
Useful options: pf, cipher, verify, commonname, cafile, capath, certificate, key, compress, fork, bind, range, tcpwrap, su, reuseaddr, retry
See also: OPENSSL, TCP-LISTEN

OPENSSL-DTLS-CLIENT:<host>:<port>

Tries to establish a DTLS connection to <port> [UDP service] on <host> [IP address] using UDP/IP version 4 or 6 depending on address specification, name resolution, or option pf.
Socat checks the peer certificates subjectAltName or commonName against the addresses option openssl-commonname or the host name. Wildcards in the certificate are supported.
Use **socat** option -b to make datagrams small enough to fit with overhead on the network. Use option -T to prevent indefinite hanging when peer went down quietly.
Option groups: FD,SOCKET,IP4,IP6,OPENSSL,RETRY
Useful options: cipher, verify, commonname, cafile, capath, certificate, key, compress, bind, pf, sourceport, retry
See also: OPENSSL-DTLS-SERVER, OPENSSL-CONNECT, UDP-CONNECT

OPENSSL-DTLS-SERVER:<port>

Listens on UDP <port> [UDP service]. The IP version is 4 or the one specified with pf. When a connection is accepted, this address behaves as DTLS server.
Note: You probably want to use the certificate option with this address.
NOTE: The client certificate is only checked for validity against cafile or capath, but not for match with the client's name or its IP address! Use **socat** option -b to make datagrams small enough to fit with overhead on the network. Use option -T to prevent indefinite hanging when peer went down quietly.
Option groups: FD,SOCKET,IP4,IP6,LISTEN,OPENSSL,CHILD,RANGE,RETRY
Useful options: pf, cipher, verify, commonname, cafile, capath, certificate, key, compress, fork, bind, range, tcpwrap, su, reuseaddr, retry
See also: OPENSSL-DTLS-CLIENT, OPENSSL-LISTEN, UDP-LISTEN

PIPE:<filename>

If <filename> already exists, it is opened. If it does not exist, a named pipe is created and opened. Beginning with socat version 1.4.3, the named pipe is removed when the address is closed (but see option unlink-close
Note: When a pipe is used for both reading and writing, it works as echo service.
Note: When a pipe is used for both reading and writing, and socat tries to write more bytes than the pipe can buffer (Linux 2.4: 2048 bytes), socat might block. Consider using socat option, e.g., `-b 2048`
Option groups: FD,NAMED,OPEN

Useful options: rdonly, nonblock, group, user, mode, unlink-early
See also: unnamed pipe

`PIPE`
Creates an unnamed pipe and uses it for reading and writing. It works as an echo, because everything written to it appeares immediately as read data.
Note: When socat tries to write more bytes than the pipe can queue (Linux 2.4: 2048 bytes), socat might block. Consider, e.g., using option `-b 2048`
Option groups: FD
See also: named pipe

`PROXY:<proxy>:<hostname>:<port>`
Connects to an HTTP proxy server on port 8080 using TCP/IP version 4 or 6 depending on address specification, name resolution, or option pf, and sends a CONNECT request for hostname:port. If the proxy grants access and succeeds to connect to the target, data transfer between socat and the target can start. Note that the traffic need not be HTTP but can be an arbitrary protocol.
Option groups: FD,SOCKET,IP4,IP6,TCP,HTTP,RETRY
Useful options: proxyport, ignorecr, proxyauth, resolve, crnl, bind, connect-timeout, mss, sourceport, retry
See also: SOCKS, TCP

`PTY`
Generates a pseudo terminal (pty) and uses its master side. Another process may open the pty's slave side using it like a serial line or terminal. (example). If both the ptmx and the openpty mechanisms are available, ptmx is used (POSIX).
Option groups: FD,NAMED,PTY,TERMIOS
Useful options: link, openpty, wait-slave, mode, user, group
See also: UNIX-LISTEN, PIPE, EXEC, SYSTEM

`READLINE`
Uses GNU readline and history on stdio to allow editing and reusing input lines (example). This requires the GNU readline and history libraries. Note that stdio should be a (pseudo) terminal device, otherwise readline does not seem to work.
Option groups: FD,READLINE,TERMIOS
Useful options: history, noecho
See also: STDIO

`SCTP-CONNECT:<host>:<port>`
Establishes an SCTP stream connection to the specified <host> [IP address] and <port> [TCP service] using IP version 4 or 6 depending on address specification, name resolution, or option pf.
Option groups: FD,SOCKET,IP4,IP6,SCTP,CHILD,RETRY
Useful options: bind, pf, connect-timeout, tos, mtudiscover, sctp-maxseg, sctp-nodelay, nonblock, sourceport, retry, readbytes
See also: SCTP4-CONNECT, SCTP6-CONNECT, SCTP-LISTEN, TCP-CONNECT

`SCTP4-CONNECT:<host>:<port>`
Like SCTP-CONNECT, but only supports IPv4 protocol.
Option groups: FD,SOCKET,IP4,SCTP,CHILD,RETRY

`SCTP6-CONNECT:<host>:<port>`
Like SCTP-CONNECT, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6,SCTP,CHILD,RETRY

`SCTP-LISTEN:<port>`
Listens on <port> [TCP service] and accepts an SCTP connection. The IP version is 4 or the one specified with address option pf, socat option (-4, -6), or environment variable SOCAT_DEFAULT_LISTEN_IP. Note that opening this address usually blocks until a client connects.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP4,IP6,SCTP,RETRY
Useful options: crnl, fork, bind, range, tcpwrap, pf, max-children, backlog, accept-timeout, sctp-maxseg, sctp-

nodelay, su, reuseaddr, retry, cool-write
See also: SCTP4-LISTEN, SCTP6-LISTEN, TCP-LISTEN, SCTP-CONNECT

`SCTP4-LISTEN:<port>`
Like SCTP-LISTEN, but only supports IPv4 protocol.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP4,SCTP,RETRY

`SCTP6-LISTEN:<port>`
Like SCTP-LISTEN, but only supports IPv6 protocol.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP6,SCTP,RETRY

`SOCKET-CONNECT:<domain>:<protocol>:<remote-address>`
Creates a stream socket using the first and second given socket parameters and `SOCK_STREAM` (see man socket(2)) and connects to the remote-address. The two socket parameters have to be specified by int numbers. Consult your OS documentation and include files to find the appropriate values. The remote-address must be the data representation of a sockaddr structure without sa_family and (BSD) sa_len components.
Please note that you can - beyond the options of the specified groups - also use options of higher level protocols when you apply socat option -g.
Option groups: FD,SOCKET,CHILD,RETRY
Useful options: bind, setsockopt,
See also: TCP, UDP-CONNECT, UNIX-CONNECT, SOCKET-LISTEN, SOCKET-SENDTO

`SOCKET-DATAGRAM:<domain>:<type>:<protocol>:<remote-address>`
Creates a datagram socket using the first three given socket parameters (see man socket(2)) and sends outgoing data to the remote-address. The three socket parameters have to be specified by int numbers. Consult your OS documentation and include files to find the appropriate values. The remote-address must be the data representation of a sockaddr structure without sa_family and (BSD) sa_len components.
Please note that you can - beyond the options of the specified groups - also use options of higher level protocols when you apply socat option -g.
Option groups: FD,SOCKET,RANGE
Useful options: bind, range, setsockopt,
See also: UDP-DATAGRAM, IP-DATAGRAM, SOCKET-SENDTO, SOCKET-RECV, SOCKET-RECVFROM

`SOCKET-LISTEN:<domain>:<protocol>:<local-address>`
Creates a stream socket using the first and second given socket parameters and `SOCK_STREAM` (see man socket(2)) and waits for incoming connections on local-address. The two socket parameters have to be specified by int numbers. Consult your OS documentation and include files to find the appropriate values. The local-address must be the data representation of a sockaddr structure without sa_family and (BSD) sa_len components.
Please note that you can - beyond the options of the specified groups - also use options of higher level protocols when you apply socat option -g.
Option groups: FD,SOCKET,LISTEN,RANGE,CHILD,RETRY
Useful options: setsockopt, setsockopt-listen,
See also: TCP, UDP-CONNECT, UNIX-CONNECT, SOCKET-LISTEN, SOCKET-SENDTO, SOCKET-SENDTO

`SOCKET-RECV:<domain>:<type>:<protocol>:<local-address>`
Creates a socket using the three given socket parameters (see man socket(2)) and binds it to <local-address>. Receives arriving data. The three parameters have to be specified by int numbers. Consult your OS documentation and include files to find the appropriate values. The local-address must be the data representation of a sockaddr structure without sa_family and (BSD) sa_len components.
Option groups: FD,SOCKET,RANGE
Useful options: range, setsockopt, setsockopt-listen
See also: UDP-RECV, IP-RECV, UNIX-RECV, SOCKET-DATAGRAM, SOCKET-SENDTO, SOCKET-RECVFROM

`SOCKET-RECVFROM:<domain>:<type>:<protocol>:<local-address>`

Creates a socket using the three given socket parameters (see man socket(2)) and binds it to <local-address>. Receives arriving data and sends replies back to the sender. The first three parameters have to be specified as int numbers. Consult your OS documentation and include files to find the appropriate values. The local-address must be the data representation of a sockaddr structure without sa_family and (BSD) sa_len components.
See the note about RECVFROM addresses.
Option groups: FD,SOCKET,CHILD,RANGE
Useful options: fork, range, setsockopt, setsockopt-listen
See also: UDP-RECVFROM, IP-RECVFROM, UNIX-RECVFROM, SOCKET-DATAGRAM, SOCKET-SENDTO, SOCKET-RECV

`SOCKET-SENDTO:<domain>:<type>:<protocol>:<remote-address>`

Creates a socket using the three given socket parameters (see man socket(2)). Sends outgoing data to the given address and receives replies. The three parameters have to be specified as int numbers. Consult your OS documentation and include files to find the appropriate values. The remote-address must be the data representation of a sockaddr structure without sa_family and (BSD) sa_len components.
Option groups: FD,SOCKET
Useful options: bind, setsockopt, setsockopt-listen
See also: UDP-SENDTO, IP-SENDTO, UNIX-SENDTO, SOCKET-DATAGRAM, SOCKET-RECV SOCKET-RECVFROM

`SOCKS4:<socks-server>:<host>:<port>`

Connects via <socks-server> [IP address] to <host> [IPv4 address] on <port> [TCP service], using socks version 4 protocol over IP version 4 or 6 depending on address specification, name resolution, or option pf (example).
Option groups: FD,SOCKET,IP4,IP6,TCP,SOCKS4,RETRY
Useful options: socksuser, socksport, sourceport, pf, retry
See also: SOCKS4A, PROXY, TCP

`SOCKS4A:<socks-server>:<host>:<port>`

like SOCKS4, but uses socks protocol version 4a, thus leaving host name resolution to the socks server.
Option groups: FD,SOCKET,IP4,IP6,TCP,SOCKS4,RETRY

`STDERR`

Uses file descriptor 2.
Option groups: FD (TERMIOS,REG,SOCKET)
See also: FD

`STDIN`

Uses file descriptor 0.
Option groups: FD (TERMIOS,REG,SOCKET)
Useful options: readbytes
See also: FD

`STDIO`

Uses file descriptor 0 for reading, and 1 for writing.
Option groups: FD (TERMIOS,REG,SOCKET)
Useful options: readbytes
See also: FD

`STDOUT`

Uses file descriptor 1.
Option groups: FD (TERMIOS,REG,SOCKET)
See also: FD

`SYSTEM:<shell-command>`

Forks a sub process that establishes communication with its parent process and invokes the specified program with `system()` . Please note that <shell-command> [string] must not contain ',' or "!!", and that shell meta

characters may have to be protected. After successful program start, **socat** writes data to stdin of the process and reads from its stdout.
Option groups: FD,SOCKET,EXEC,FORK,TERMIOS
Useful options: path, fdin, fdout, chroot, su, su-d, nofork, pty, stderr, ctty, setsid, pipes, sigint, sigquit
See also: EXEC

`TCP:<host>:<port>`
Connects to <port> [TCP service] on <host> [IP address] using TCP/IP version 4 or 6 depending on address specification, name resolution, or option pf.
Option groups: FD,SOCKET,IP4,IP6,TCP,RETRY
Useful options: crnl, bind, pf, connect-timeout, tos, mtudiscover, mss, nodelay, nonblock, sourceport, retry, readbytes
See also: TCP4, TCP6, TCP-LISTEN, UDP, SCTP-CONNECT, UNIX-CONNECT

`TCP4:<host>:<port>`
Like TCP, but only supports IPv4 protocol (example).
Option groups: FD,SOCKET,IP4,TCP,RETRY

`TCP6:<host>:<port>`
Like TCP, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6,TCP,RETRY

`TCP-LISTEN:<port>`
Listens on <port> [TCP service] and accepts a TCP/IP connection. The IP version is 4 or the one specified with address option pf, socat option (-4, -6), or environment variable SOCAT_DEFAULT_LISTEN_IP. Note that opening this address usually blocks until a client connects.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP4,IP6,TCP,RETRY
Useful options: crnl, fork, bind, range, tcpwrap, pf, max-children, backlog, accept-timeout, mss, su, reuseaddr, retry, cool-write
See also: TCP4-LISTEN, TCP6-LISTEN, UDP-LISTEN, SCTP-LISTEN, UNIX-LISTEN, OPENSSL-LISTEN, TCP-CONNECT

`TCP4-LISTEN:<port>`
Like TCP-LISTEN, but only supports IPv4 protocol (example).
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP4,TCP,RETRY

`TCP6-LISTEN:<port>`
Like TCP-LISTEN, but only supports IPv6 protocol.
Additional useful option: ipv6only
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP6,TCP,RETRY

`TUN[:<if-addr>/<bits>]`
Creates a Linux TUN/TAP device and optionally assignes it the address and netmask given by the parameters. The resulting network interface is almost ready for use by other processes; socat serves its "wire side". This address requires read and write access to the tunnel cloning device, usually `/dev/net/tun` , as well as permission to set some `ioctl()s`. **Option iff-up is required to immediately activate the interface!**
Note: If you intend to transfer packets between two Socat "wire sides" you need a protocol that keeps packet boundaries, e.g.UDP; TCP might work with option nodelay.
Option groups: FD,NAMED,OPEN,TUN
Useful options: iff-up, tun-device, tun-name, tun-type, iff-no-pi
See also: ip-recv

`UDP:<host>:<port>`
Connects to <port> [UDP service] on <host> [IP address] using UDP/IP version 4 or 6 depending on address specification, name resolution, or option pf.
Please note that, due to UDP protocol properties, no real connection is established; data has to be sent for `connecting' to the server, and no end-of-file condition can be transported.
Option groups: FD,SOCKET,IP4,IP6

Useful options: ttl, tos, bind, sourceport, pf
See also: UDP4, UDP6, UDP-LISTEN, TCP, IP

`UDP4:<host>:<port>`
Like UDP, but only supports IPv4 protocol.
Option groups: FD,SOCKET,IP4

`UDP6:<host>:<port>`
Like UDP, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6

`UDP-DATAGRAM:<address>:<port>`
Sends outgoing data to the specified address which may in particular be a broadcast or multicast address. Packets arriving on the local socket are checked for the correct remote port only when option sourceport is used (this is a change with **Socat** version 1.7.4.0) and if their source addresses match RANGE or TCPWRAP options. This address type can for example be used for implementing symmetric or asymmetric broadcast or multicast communications.
Option groups: FD,SOCKET,IP4,IP6,RANGE
Useful options: bind, range, tcpwrap, broadcast, ip-multicast-loop, ip-multicast-ttl, ip-multicast-if, ip-add-membership, ip-add-source-membership, ttl, tos, sourceport, pf
See also: UDP4-DATAGRAM, UDP6-DATAGRAM, UDP-SENDTO, UDP-RECVFROM, UDP-RECV, UDP-CONNECT, UDP-LISTEN, IP-DATAGRAM

`UDP4-DATAGRAM:<address>:<port>`
Like UDP-DATAGRAM, but only supports IPv4 protocol (example1, example2).
Option groups: FD,SOCKET,IP4, RANGE

`UDP6-DATAGRAM:<address>:<port>`
Like UDP-DATAGRAM, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6,RANGE

`UDP-LISTEN:<port>`
Waits for a UDP/IP packet arriving on <port> [UDP service] and `connects' back to sender. The accepted IP version is 4 or the one specified with option pf. Please note that, due to UDP protocol properties, no real connection is established; data has to arrive from the peer first, and no end-of-file condition can be transported. Note that opening this address usually blocks until a client connects.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP4,IP6
Useful options: fork, bind, range, pf
See also: UDP, UDP4-LISTEN, UDP6-LISTEN, TCP-LISTEN

`UDP4-LISTEN:<port>`
Like UDP-LISTEN, but only support IPv4 protocol.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP4

`UDP6-LISTEN:<port>`
Like UDP-LISTEN, but only support IPv6 protocol.
Option groups: FD,SOCKET,LISTEN,CHILD,RANGE,IP6

`UDP-SENDTO:<host>:<port>`
Communicates with the specified peer socket, defined by <port> [UDP service] on <host> [IP address], using UDP/IP version 4 or 6 depending on address specification, name resolution, or option pf. It sends packets to and receives packets from that peer socket only. This address effectively implements a datagram client. It works well with socat UDP-RECVFROM and UDP-RECV address peers.
Option groups: FD,SOCKET,IP4,IP6
Useful options: ttl, tos, bind, sourceport, pf
See also: UDP4-SENDTO, UDP6-SENDTO, UDP-RECVFROM, UDP-RECV, UDP-CONNECT, UDP-LISTEN, IP-SENDTO

`UDP4-SENDTO:<host>:<port>`

Like UDP-SENDTO, but only supports IPv4 protocol.
Option groups: FD,SOCKET,IP4

`UDP6-SENDTO:<host>:<port>`

Like UDP-SENDTO, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6

`UDP-RECVFROM:<port>`

Creates a UDP socket on <port> [UDP service] using UDP/IP version 4 or 6 depending on option pf. It receives one packet from an unspecified peer and may send one or more answer packets to that peer. This mode is particularly useful with fork option where each arriving packet - from arbitrary peers - is handled by its own sub process. This allows a behaviour similar to typical UDP based servers like ntpd or named. This address works well with socat UDP-SENDTO address peers.
Note: When the second address fails before entering the transfer loop the packet is dropped. Use option retry or forever on the second address to avoid data loss.
Option groups: FD,SOCKET,IP4,IP6,CHILD,RANGE
Useful options: fork, ttl, tos, bind, sourceport, pf
See also: UDP4-RECVFROM, UDP6-RECVFROM, UDP-SENDTO, UDP-RECV, UDP-CONNECT, UDP-LISTEN, IP-RECVFROM, UNIX-RECVFROM

`UDP4-RECVFROM:<port>`

Like UDP-RECVFROM, but only supports IPv4 protocol.
Option groups: FD,SOCKET,IP4,CHILD,RANGE

`UDP6-RECVFROM:<port>`

Like UDP-RECVFROM, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6,CHILD,RANGE

`UDP-RECV:<port>`

Creates a UDP socket on <port> [UDP service] using UDP/IP version 4 or 6 depending on option pf. It receives packets from multiple unspecified peers and merges the data. No replies are possible. It works well with, e.g., socat UDP-SENDTO address peers; it behaves similar to a syslog server.
Note: if you need the fork option, use UDP-RECVFROM in unidirectional mode (with option -u) instead.
Option groups: FD,SOCKET,IP4,IP6,RANGE
Useful options: pf, bind, sourceport, ttl, tos
See also: UDP4-RECV, UDP6-RECV, UDP-SENDTO, UDP-RECVFROM, UDP-CONNECT, UDP-LISTEN, IP-RECV, UNIX-RECV

`UDP4-RECV:<port>`

Like UDP-RECV, but only supports IPv4 protocol.
Option groups: FD,SOCKET,IP4,RANGE

`UDP6-RECV:<port>`

Like UDP-RECV, but only supports IPv6 protocol.
Option groups: FD,SOCKET,IP6,RANGE

`UNIX-CONNECT:<filename>`

Connects to <filename> assuming it is a UNIX domain socket. If <filename> does not exist, this is an error; if <filename> is not a UNIX domain socket, this is an error; if <filename> is a UNIX domain socket, but no process is listening, this is an error.
Option groups: FD,SOCKET,NAMED,RETRY,UNIX
) Useful options: bind
See also: UNIX-LISTEN, UNIX-SENDTO, TCP

`UNIX-LISTEN:<filename>`

Listens on <filename> using a UNIX domain stream socket and accepts a connection. If <filename> exists and is not a socket, this is an error. If <filename> exists and is a UNIX domain socket, binding to the address fails

(use option unlink-early!). Note that opening this address usually blocks until a client connects. Beginning with socat version 1.4.3, the file system entry is removed when this address is closed (but see option unlink-close) (example).
Option groups: FD,SOCKET,NAMED,LISTEN,CHILD,RETRY,UNIX
Useful options: fork, umask, mode, user, group, unlink-early
See also: UNIX-CONNECT, UNIX-RECVFROM, UNIX-RECV, TCP-LISTEN

`UNIX-SENDTO:<filename>`
Communicates with the specified peer socket, defined by [<filename>] assuming it is a UNIX domain datagram socket. It sends packets to and receives packets from that peer socket only. Please note that it might be necessary to bind the local socket to an address (e.g. `/tmp/sock1`, which must not exist before). This address type works well with socat UNIX-RECVFROM and UNIX-RECV address peers.
Option groups: FD,SOCKET,NAMED,UNIX
Useful options: bind
See also: UNIX-RECVFROM, UNIX-RECV, UNIX-CONNECT, UDP-SENDTO, IP-SENDTO

`UNIX-RECVFROM:<filename>`
Creates a UNIX domain datagram socket [<filename>]. Receives one packet and may send one or more answer packets to that peer. This mode is particularly useful with fork option where each arriving packet - from arbitrary peers - is handled by its own sub process. This address works well with socat UNIX-SENDTO address peers.
Option groups: FD,SOCKET,NAMED,CHILD,UNIX
See the note about RECVFROM addresses.
Useful options: fork
See also: UNIX-SENDTO, UNIX-RECV, UNIX-LISTEN, UDP-RECVFROM, IP-RECVFROM

`UNIX-RECV:<filename>`
Creates a UNIX domain datagram socket [<filename>]. Receives packets from multiple unspecified peers and merges the data. No replies are possible. It can be, e.g., addressed by socat UNIX-SENDTO address peers. It behaves similar to a syslog server.
Option groups: FD,SOCKET,NAMED,UNIX
See also: UNIX-SENDTO, UNIX-RECVFROM, UNIX-LISTEN, UDP-RECV, IP-RECV

`UNIX-CLIENT:<filename>`
Communicates with the specified peer socket, defined by [<filename>] assuming it is a UNIX domain socket. It first tries to connect and, if that fails, assumes it is a datagram socket, thus supporting both types.
Option groups: FD,SOCKET,NAMED,UNIX
Useful options: bind
See also: UNIX-CONNECT, UNIX-SENDTO, GOPEN

`VSOCK-CONNECT:<cid>:<port>`
Establishes a VSOCK stream connection to the specified <cid> [VSOCK cid] and <port> [VSOCK port].
Option groups: FD,SOCKET,CHILD,RETRY
Useful options: bind, pf, connect-timeout, retry, readbytes
See also: VSOCK-LISTEN,

`VSOCK-LISTEN:<port>`
Listens on <port> [VSOCK port] and accepts a VSOCK connection. Note that opening this address usually blocks until a client connects.
Option groups: FD,SOCKET,LISTEN,CHILD,RETRY
Useful options: fork, bind, pf, max-children, backlog, su, reuseaddr, retry, cool-write
See also: VSOCK-CONNECT

`ABSTRACT-CONNECT:<string>`

`ABSTRACT-LISTEN:<string>`

`ABSTRACT-SENDTO:<string>`

`ABSTRACT-RECVFROM:<string>`

`ABSTRACT-RECV:<string>`

`ABSTRACT-CLIENT:<string>`

The ABSTRACT addresses are almost identical to the related UNIX addresses except that they do not address file system based sockets but an alternate UNIX domain address space. To achieve this the socket address strings are prefixed with "\0" internally. This feature is available (only?) on Linux. Option groups are the same as with the related UNIX addresses, except that the ABSTRACT addresses are not member of the NAMED group.

ADDRESS OPTIONS

Address options can be applied to address specifications to influence the process of opening the addresses and the properties of the resulting data channels.

For technical reasons not every option can be applied to every address type; e.g., applying a socket option to a regular file will fail. To catch most useless combinations as early as in the open phase, the concept of *option groups* was introduced. Each option belongs to one or more option groups. Options can be used only with address types that support at least one of their option groups (but see option -g).

Address options have data types that their values must conform to. Every address option consists of just a keyword or a keyword followed by "=value", where value must conform to the options type. Some address options manipulate parameters of system calls; e.g., option sync sets the `O_SYNC` flag with the `open()` call. Other options cause a system or library call; e.g., with option `ttl=value' the `setsockopt(fd, SOL_IP, IP_TTL, value, sizeof(int))` call is applied. Other options set internal **socat** variables that are used during data transfer; e.g., `crnl' causes explicit character conversions. A few options have more complex implementations; e.g., su-d (substuser-delayed) inquires some user and group infos, stores them, and applies them later after a possible `chroot()` call.

If multiple options are given to an address, their sequence in the address specification has (almost) no effect on the sequence of their execution/application. Instead, **socat** has built in an *option phase* model that tries to bring the options in a useful order. Some options exist in different forms (e.g., unlink, unlink-early, unlink-late) to control the time of their execution.

If the same option is specified more than once within one address specification, with equal or different values, the effect depends on the kind of option. Options resulting in function calls like `setsockopt()` cause multiple invocations. With options that set parameters for a required call like `open()` or set internal flags, the value of the last option occurrence is effective.

The existence or semantics of many options are system dependent. **Socat** usually does NOT try to emulate missing libc or kernel features, it just provides an interface to the underlying system. So, if an operating system lacks a feature, the related option is simply not available on this platform.

The following paragraphs introduce just the more common address options. For a more comprehensive reference and to find information about canonical option names, alias names, option phases, and platforms see file **xio.help**.

FD option group

This option group contains options that are applied to a UN*X style file descriptor, no matter how it was generated. Because all current **socat** address types are file descriptor based, these options may be applied to any address.
Note: Some of these options are also member of another option group, that provides another, non-fd based mechanism. For these options, it depends on the actual address type and its option groups which mechanism is used. The second, non-fd based mechanism is prioritized.

`cloexec=<bool>`
Sets the `FD_CLOEXEC` flag with the `fcntl()` system call to value <bool>. If set, the file descriptor is closed on `exec()` family function calls. **Socat** internally handles this flag for the fds it controls, so in most cases there will be no need to apply this option.

`setlk`
Tries to set a discretionary write lock to the whole file using the `fcntl(fd, F_SETLK, ...)` system call. If the file is already locked, this call results in an error. On Linux, when the file permissions for group are "S" (g-x,g+s), and the file system is locally mounted with the "mand" option, the lock is mandatory, i.e. prevents other processes from opening the file.

`setlkw`
Tries to set a discretionary waiting write lock to the whole file using the `fcntl(fd, F_SETLKW, ...)` system call. If the file is already locked, this call blocks. See option setlk for information about making this lock mandatory.

`setlk-rd`
Tries to set a discretionary read lock to the whole file using the `fcntl(fd, F_SETLK, ...)` system call. If the file is already write locked, this call results in an error. See option setlk for information about making this lock mandatory.

`setlkw-rd`
Tries to set a discretionary waiting read lock to the whole file using the `fcntl(fd, F_SETLKW, ...)` system call. If the file is already write locked, this call blocks. See option setlk for information about making this lock mandatory.

`flock-ex`
Tries to set a blocking exclusive advisory lock to the file using the `flock(fd, LOCK_EX)` system call. **Socat** hangs in this call if the file is locked by another process.

`flock-ex-nb`
Tries to set a nonblocking exclusive advisory lock to the file using the `flock(fd, LOCK_EX|LOCK_NB)` system call. If the file is already locked, this option results in an error.

`flock-sh`
Tries to set a blocking shared advisory lock to the file using the `flock(fd, LOCK_SH)` system call. **Socat** hangs in this call if the file is locked by another process.

`flock-sh-nb`
Tries to set a nonblocking shared advisory lock to the file using the `flock(fd, LOCK_SH|LOCK_NB)` system call. If the file is already locked, this option results in an error.

`lock`
Sets a blocking lock on the file. Uses the setlk or flock mechanism depending on availability on the particular platform. If both are available, the POSIX variant (setlkw) is used.

`user=<user>`
Sets the <user> (owner) of the stream. If the address is member of the NAMED option group, **socat** uses the `chown()` system call after opening the file or binding to the UNIX domain socket (race condition!). Without filesystem entry, **socat** sets the user of the stream using the `fchown()` system call. These calls might require root privilege.

`user-late=<user>`
Sets the owner of the fd to <user> with the `fchown()` system call after opening or connecting the channel. This is useful only on file system entries.

`group=<group>`
Sets the <group> of the stream. If the address is member of the NAMED option group, **socat** uses the `chown()` system call after opening the file or binding to the UNIX domain socket (race condition!). Without filesystem

entry, **socat** sets the group of the stream with the `fchown()` system call. These calls might require group membership or root privilege.

`group-late=<group>`
Sets the group of the fd to <group> with the `fchown()` system call after opening or connecting the channel. This is useful only on file system entries.

`mode=<mode>`
Sets the <mode> [mode_t] (permissions) of the stream. If the address is member of the NAMED option group and uses the `open()` or `creat()` call, the mode is applied with these. If the address is member of the NAMED option group without using these system calls, **socat** uses the `chmod()` system call after opening the filesystem entry or binding to the UNIX domain socket (race condition!). Otherwise, **socat** sets the mode of the stream using `fchmod()` . These calls might require ownership or root privilege.

`perm-late=<mode>`
Sets the permissions of the fd to value <mode> [mode_t] using the `fchmod()` system call after opening or connecting the channel. This is useful only on file system entries.

`append=<bool>`
Always writes data to the actual end of file. If the address is member of the OPEN option group, **socat** uses the `O_APPEND` flag with the `open()` system call (example). Otherwise, **socat** applies the `fcntl(fd, F_SETFL, O_APPEND)` call.

`nonblock=<bool>`
Tries to open or use file in nonblocking mode. Its only effects are that the `connect()` call of TCP addresses does not block, and that opening a named pipe for reading does not block. If the address is member of the OPEN option group, **socat** uses the `O_NONBLOCK` flag with the `open()` system call. Otherwise, **socat** applies the `fcntl(fd, F_SETFL, O_NONBLOCK)` call.

`binary`
Opens the file in binary mode to avoid implicit line terminator conversions (Cygwin).

`text`
Opens the file in text mode to force implicit line terminator conversions (Cygwin).

`noinherit`
Does not keep this file open in a spawned process (Cygwin).

`cool-write`
Takes it easy when write fails with EPIPE or ECONNRESET and logs the message with *notice* level instead of *error*. This prevents the log file from being filled with useless error messages when socat is used as a high volume server or proxy where clients often abort the connection.
This option is experimental.

`end-close`
Changes the (address dependent) method of ending a connection to just close the file descriptors. This is useful when the connection is to be reused by or shared with other processes (example).
Normally, socket connections will be ended with `shutdown(2)` which terminates the socket even if it is shared by multiple processes. `close(2)` "unlinks" the socket from the process but keeps it active as long as there are still links from other processes.
Similarly, when an address of type EXEC or SYSTEM is ended, socat usually will explicitly kill the sub process. With this option, it will just close the file descriptors.

`shut-none`
Changes the (address dependent) method of shutting down the write part of a connection to not do anything.

`shut-down`
Changes the (address dependent) method of shutting down the write part of a connection to shutdown(fd, SHUT_WR). Is only useful with sockets.

`shut-close`
: Changes the (address dependent) method of shutting down the write part of a connection to close(fd).

`shut-null`
: When one address indicates EOF, **socat** will send a zero sized packet to the write channel of the other address to transfer the EOF condition. This is useful with UDP and other datagram protocols. Has been tested against netcat and socat with option null-eof.

`null-eof`
: Normally **socat** will ignore empty (zero size payload) packets arriving on datagram sockets, so it survives port scans. With this option **socat** interprets empty datagram packets as EOF indicator (see shut-null).

`ioctl-void=<request>`
: Calls `ioctl()` with the request value as second argument and NULL as third argument. This option allows utilizing ioctls that are not explicitly implemented in socat.

`ioctl-int=<request>:<value>`
: Calls `ioctl()` with the request value as second argument and the integer value as third argument.

`ioctl-intp=<request>:<value>`
: Calls `ioctl()` with the request value as second argument and a pointer to the integer value as third argument.

`ioctl-bin=<request>:<value>`
: Calls `ioctl()` with the request value as second argument and a pointer to the given data value as third argument. This data must be specified in <dalan> form.

`ioctl-string=<request>:<value>`
: Calls `ioctl()` with the request value as second argument and a pointer to the given string as third argument. <dalan> form.

NAMED option group

These options work on file system entries.
Please note that, with UNIX domain client addresses, this means the bind entry, not the target/peer entry.
See also options user, group, and mode.

`user-early=<user>`
: Changes the <user> (owner) of the file system entry before accessing it, using the `chown()` system call. This call might require root privilege.

`group-early=<group>`
: Changes the <group> of the file system entry before accessing it, using the `chown()` system call. This call might require group membership or root privilege.

`perm-early=<mode>`
: Changes the <mode> [mode_t] of the file system entry before accessing it, using the `chmod()` system call. This call might require ownership or root privilege.

`umask=<mode>`
: Sets the umask of the process to <mode> [mode_t] before accessing the file system entry (useful with UNIX domain sockets!). This call might affect all further operations of the **socat** process!

`unlink-early`
: Unlinks (removes) the file before opening it and even before applying user-early etc.

`unlink`
: Unlinks (removes) the file before accessing it, but after user-early etc.

`unlink-late`
: Unlinks (removes) the file after opening it to make it inaccessible for other processes after a short race condition.

`unlink-close`
: Removes the addresses file system entry when closing the address. For named pipes, UNIX domain sockets, and the symbolic links of pty addresses, the default is 1; for created files, opened files, and generic opened files the default is 0.

OPEN option group

The OPEN group options allow setting flags with the `open()` system call. E.g., option `creat' sets the `O_CREAT` flag. See also options append and nonblock.

`creat=<bool>`
: Creates the file if it does not exist (example).

`dsync=<bool>`
: Blocks `write()` calls until metainfo is physically written to media.

`excl=<bool>`
: With option creat, if file exists this is an error.

`largefile=<bool>`
: On 32 bit systems, allows a file larger than 2^31 bytes.

`noatime`
: Sets the O_NOATIME options, so reads do not change the access timestamp.

`noctty=<bool>`
: Does not make this file the controlling terminal.

`nofollow=<bool>`
: Does not follow symbolic links.

`nshare=<bool>`
: Does not allow sharing this file with other processes.

`rshare=<bool>`
: Does not allow other processes to open this file for writing.

`rsync=<bool>`
: Blocks `write()` until metainfo is physically written to media.

`sync=<bool>`
: Blocks `write()` until data is physically written to media.

`rdonly=<bool>`
: Opens the file for reading only.

`wronly=<bool>`
: Opens the file for writing only.

`trunc`
: Truncates the file to size 0 during opening it.

REG and BLK option group

These options are usually applied to a UN*X file descriptor, but their semantics make sense only on a file supporting random access.

`seek=<offset>`
: Applies the `lseek(fd, <offset>, SEEK_SET)` (or `lseek64`) system call, thus positioning the file pointer absolutely to <offset> [off_t or off64_t]. Please note that a missing value defaults to 1, not 0.

`seek-cur=<offset>`
: Applies the `lseek(fd, <offset>, SEEK_CUR)` (or `lseek64`) system call, thus positioning the file pointer <offset> [off_t or off64_t] bytes relatively to its current position (which is usually 0). Please note that a missing value defaults to 1, not 0.

`seek-end=<offset>`
: Applies the `lseek(fd, <offset>, SEEK_END)` (or `lseek64`) system call, thus positioning the file pointer <offset> [off_t or off64_t] bytes relatively to the files current end. Please note that a missing value defaults to 1, not 0.

`ftruncate=<offset>`
: Applies the `ftruncate(fd, <offset>)` (or `ftruncate64` if available) system call, thus truncating the file at the position <offset> [off_t or off64_t]. Please note that a missing value defaults to 1, not 0.

`secrm=<bool>`

`unrm=<bool>`

`compr=<bool>`

`fs-sync=<bool>`

`immutable=<bool>`

`fs-append=<bool>`

`nodump=<bool>`

`fs-noatime=<bool>`

`journal-data=<bool>`

`notail=<bool>`

`dirsync=<bool>`
: These options change non standard file attributes on operating systems and file systems that support these features, like Linux with ext2fs and successors, xfs, or reiserfs. See man 1 chattr for information on these options. Please note that there might be a race condition between creating the file and applying these options.

PROCESS option group

Options of this group change the process properties instead of just affecting one data channel. For EXEC and SYSTEM addresses and for LISTEN and CONNECT type addresses with option FORK, these options apply to the child processes instead of the main socat process.

`chroot=<directory>`
: Performs a `chroot()` operation to <directory> after processing the address (example). This call might require root privilege.

`chroot-early=<directory>`
: Performs a `chroot()` operation to <directory> before opening the address. This call might require root privilege.

`setgid=<group>`
Changes the primary <group> of the process after processing the address. This call might require root privilege. Please note that this option does not drop other group related privileges.

`setgid-early=<group>`
Like setgit but is performed before opening the address.

`setuid=<user>`
Changes the <user> (owner) of the process after processing the address. This call might require root privilege. Please note that this option does not drop group related privileges. Check if option su better fits your needs.

`setuid-early=<user>`
Like setuid but is performed before opening the address.

`su=<user>`
Changes the <user> (owner) and groups of the process after processing the address (example). This call might require root privilege.

`su-d=<user>`
Short name for `substuser-delayed`. Changes the <user> (owner) and groups of the process after processing the address (example). The user and his groups are retrieved *before* a possible `chroot()` . This call might require root privilege.

`setpgid=<pid_t>`
Makes the process a member of the specified process group <pid_t>. If no value is given, or if the value is 0 or 1, the process becomes leader of a new process group.

`setsid`
Makes the process the leader of a new session (example).

READLINE option group

These options apply to the readline address type.

`history=<filename>`
Reads and writes history from/to <filename> (example).

`noprompt`
Since version 1.4.0, socat per default tries to determine a prompt - that is then passed to the readline call - by remembering the last incomplete line of the output. With this option, socat does not pass a prompt to readline, so it begins line editing in the first column of the terminal.

`noecho=<pattern>`
Specifies a regular pattern for a prompt that prevents the following input line from being displayed on the screen and from being added to the history. The prompt is defined as the text that was output to the readline address after the lastest newline character and before an input character was typed. The pattern is a regular expression, e.g. "^[Pp]assword:.*$" or "([Uu]ser:|[Pp]assword:)". See regex(7) for details. (example)

`prompt=<string>`
Passes the string as prompt to the readline function. readline prints this prompt when stepping through the history. If this string matches a constant prompt issued by an interactive program on the other socat address, consistent look and feel can be achieved.

APPLICATION option group

This group contains options that work at data level. Note that these options only apply to the "raw" data transferred by socat, but not to protocol data used by addresses like PROXY.

`cr`
: Converts the default line termination character NL ('\n', 0x0a) to/from CR ('\r', 0x0d) when writing/reading on this channel.

`crnl`
: Converts the default line termination character NL ('\n', 0x0a) to/from CRNL ("\r\n", 0x0d0a) when writing/reading on this channel (example). Note: socat simply strips all CR characters.

`ignoreeof`
: When EOF occurs on this channel, **socat** ignores it and tries to read more data (like "tail -f") (example).

`readbytes=<bytes>`
: **socat** reads only so many bytes from this address (the address provides only so many bytes for transfer and pretends to be at EOF afterwards). Must be greater than 0.

`lockfile=<filename>`
: If lockfile exists, exits with error. If lockfile does not exist, creates it and continues, unlinks lockfile on exit.

`waitlock=<filename>`
: If lockfile exists, waits until it disappears. When lockfile does not exist, creates it and continues, unlinks lockfile on exit.

`escape=<int>`
: Specifies the numeric code of a character that triggers EOF on the input stream. It is useful with a terminal in raw mode (example).

SOCKET option group

These options are intended for all kinds of sockets, e.g. IP or UNIX domain. Most are applied with a `setsockopt()` call.

`bind=<sockname>`
: Binds the socket to the given socket address using the `bind()` system call. The form of <sockname> is socket domain dependent: IP4 and IP6 allow the form [hostname|hostaddress][:(service|port)] (example), UNIX domain sockets require <filename>, VSOCK allow the form [cid][:(port)].

`connect-timeout=<seconds>`
: Abort the connection attempt after <seconds> [timeval] with error status.

`so-bindtodevice=<interface>`
: Binds the socket to the given <interface>. This option might require root privilege.

`broadcast`
: For datagram sockets, allows sending to broadcast addresses and receiving packets addressed to broadcast addresses.

`debug`
: Enables socket debugging.

`dontroute`
: Only communicates with directly connected peers, does not use routers.

`keepalive`
: Enables sending keepalives on the socket.

`linger=<seconds>`

Blocks `shutdown()` or `close()` until data transfers have finished or the given timeout [int] expired.

`oobinline`
Places out-of-band data in the input data stream.

`priority=<priority>`
Sets the protocol defined <priority> [<int>] for outgoing packets.

`rcvbuf=<bytes>`
Sets the size of the receive buffer after the `socket()` call to <bytes> [int]. With TCP sockets, this value corresponds to the socket's maximal window size.

`rcvbuf-late=<bytes>`
Sets the size of the receive buffer when the socket is already connected to <bytes> [int]. With TCP sockets, this value corresponds to the socket's maximal window size.

`rcvlowat=<bytes>`
Specifies the minimum number of received bytes [int] until the socket layer will pass the buffered data to **socat**.

`reuseaddr`
Allows other sockets to bind to an address even if parts of it (e.g. the local port) are already in use by **socat** (example).

`sndbuf=<bytes>`
Sets the size of the send buffer after the `socket()` call to <bytes> [int].

`sndbuf-late=<bytes>`
Sets the size of the send buffer when the socket is connected to <bytes> [int].

`sndlowat=<bytes>`
Specifies the minimum number of bytes in the send buffer until the socket layer will send the data to <bytes> [int].

`pf=<string>`
Forces the use of the specified IP version or protocol. <string> can be something like "ip4" or "ip6". The resulting value is used as first argument to the `socket()` or `socketpair()` calls. This option affects address resolution and the required syntax of bind and range options.

`type=<type>`
Sets the type of the socket, specified as second argument to the `socket()` or `socketpair()` calls, to <type> [int]. Address resolution is not affected by this option. Under Linux, 1 means stream oriented socket, 2 means datagram socket, and 3 means raw socket.

`protocol`
Sets the protocol of the socket, specified as third argument to the `socket()` or `socketpair()` calls, to <protocol> [int]. Address resolution is not affected by this option. 6 means TCP, 17 means UDP.

`reuseport`
Set the `SO_REUSEPORT` socket option.

`so-timestamp`
Sets the SO_TIMESTAMP socket option. This enables receiving and logging of timestamp ancillary messages.

`setsockopt=<level>:<optname>:<optval>`
Invokes `setsockopt()` for the socket with the given parameters. `level` [int] is used as second argument to `setsockopt()` and specifies the layer, e.g. SOL_TCP for TCP (6 on Linux), or SOL_SOCKET for the socket layer (1 on Linux). `optname` [int] is the third argument to `setsockopt()` and tells which socket option is to be set. For the actual numbers you might have to look up the appropriate include files of your system. For the 4th and 5th `setsockopt()` parameters, `value` [dalan] specifies an arbitrary sequence of bytes that are passed to the function per pointer, with the automatically derived length parameter.

`setsockopt-int=<level>:<optname>:<optval>`
: Like `setsockopt`, but <optval> is a pointer to int [int]

`setsockopt-listen=<level>:<optname>:<optval>`
: Like `setsockopt`, but for listen type addresses it is applied to the listening socket instead of the connected socket.

`setsockopt-string=<level>:<optname>:<optval>`
: Like `setsockopt`, but <optval> is a string. This string is passed to the function with trailing null character, and the length parameter is automatically derived from the data.

UNIX option group

These options apply to UNIX domain based addresses.

`unix-tightsocklen=[0|1]`
: On socket operations, pass a socket address length that does not include the whole `struct sockaddr_un` record but (besides other components) only the relevant part of the filename or abstract string. Default is 1.

IP4 and IP6 option groups

These options can be used with IPv4 and IPv6 based sockets.

`tos=<tos>`
: Sets the TOS (type of service) field of outgoing packets to <tos> [byte] (see RFC 791).

`ttl=<ttl>`
: Sets the TTL (time to live) field of outgoing packets to <ttl> [byte].

`ip-options=<data>`
: Sets IP options like source routing. Must be given in binary form, recommended format is a leading "x" followed by an even number of hex digits. This option may be used multiple times, data are appended. E.g., to connect to host 10.0.0.1 via some gateway using a loose source route, use the gateway as address parameter and set a loose source route using the option `ip-options=x8307040a000001` .
IP options are defined in RFC 791.

`mtudiscover=<0|1|2>`
: Takes 0, 1, 2 to never, want, or always use path MTU discover on this socket.

`ip-pktinfo`
: Sets the IP_PKTINFO socket option. This enables receiving and logging of ancillary messages containing destination address and interface (Linux) (example).

`ip-recverr`
: Sets the IP_RECVERR socket option. This enables receiving and logging of ancillary messages containing detailed error information.

`ip-recvopts`
: Sets the IP_RECVOPTS socket option. This enables receiving and logging of IP options ancillary messages (Linux, *BSD).

`ip-recvtos`
: Sets the IP_RECVTOS socket option. This enables receiving and logging of TOS (type of service) ancillary messages (Linux).

`ip-recvttl`
: Sets the IP_RECVTTL socket option. This enables receiving and logging of TTL (time to live) ancillary messages (Linux, *BSD).

`ip-recvdstaddr`
: Sets the IP_RECVDSTADDR socket option. This enables receiving and logging of ancillary messages containing destination address (*BSD) (example).

`ip-recvif`
: Sets the IP_RECVIF socket option. This enables receiving and logging of interface ancillary messages (*BSD) (example).

`ip-add-membership=<multicast-address:interface-address>`

`ip-add-membership=<multicast-address:interface-name>`

`ip-add-membership=<multicast-address:interface-index>`

`ip-add-membership=<multicast-address:interface-address:interface-name>`

`ip-add-membership=<multicast-address:interface-address:interface-index>`
: Makes the socket member of the specified multicast group. This is currently only implemented for IPv4. The option takes the IP address of the multicast group and info about the desired network interface. The most common syntax is the first one, while the others are only available on systems that provide `struct mreqn` (Linux).
The indices of active network interfaces can be shown using the utility **procan**.

`ip-add-source-membership=<multicast-address:interface-address:source-address>`
: Makes the socket member of the specified multicast group for the specified source, i.e. only multicast traffic from this address is to be delivered. This is currently only implemented for IPv4.

`ip-multicast-if=<hostname>`
: Specifies hostname or address of the network interface to be used for multicast traffic.

`ip-multicast-loop=<bool>`
: Specifies if outgoing multicast traffic should loop back to the interface.

`ip-multicast-ttl=<byte>`
: Sets the TTL used for outgoing multicast traffic. Default is 1.

`ip-transparent`
: Sets the IP_TRANSPARENT socket option. This option might require root privilege.

`res-debug`

`res-aaonly`

`res-usevc`

`res-primary`

`res-igntc`

`res-recurse`

`res-defnames`

`res-stayopen`

`res-dnsrch`
: These options set the corresponding resolver (name resolution) option flags. Append "=0" to clear a default option. See man resolver(5) for more information on these options. Note: these options are valid only for the address they are applied to.

IP6 option group

These options can only be used on IPv6 based sockets. See IP options for options that can be applied to both IPv4 and IPv6 sockets.

`ipv6only=<bool>`
: Sets the IPV6_V6ONLY socket option. If 0, the TCP stack will also accept connections using IPv4 protocol on the same port. The default is system dependent.

`ipv6-recvdstopts`
: Sets the IPV6_RECVDSTOPTS socket option. This enables receiving and logging of ancillary messages containing the destination options.

`ipv6-recvhoplimit`
: Sets the IPV6_RECVHOPLIMIT socket option. This enables receiving and logging of ancillary messages containing the hoplimit.

`ipv6-recvhopopts`
: Sets the IPV6_RECVHOPOPTS socket option. This enables receiving and logging of ancillary messages containing the hop options.

`ipv6-recvpktinfo`
: Sets the IPV6_RECVPKTINFO socket option. This enables receiving and logging of ancillary messages containing destination address and interface.

`ipv6-unicast-hops=link(TYPE_INT)(<int>)`
: Sets the IPV6_UNICAST_HOPS socket option. This sets the hop count limit (TTL) for outgoing unicast packets.

`ipv6-recvrthdr`
: Sets the IPV6_RECVRTHDR socket option. This enables receiving and logging of ancillary messages containing routing information.

`ipv6-tclass`
: Sets the IPV6_TCLASS socket option. This sets the transfer class of outgoing packets.

`ipv6-recvtclass`
: Sets the IPV6_RECVTCLASS socket option. This enables receiving and logging of ancillary messages containing the transfer class.

TCP option group

These options may be applied to TCP sockets. They work by invoking `setsockopt()` with the appropriate parameters.

`cork`
: Doesn't send packets smaller than MSS (maximal segment size).

`defer-accept`
: While listening, accepts connections only when data from the peer arrived.

`keepcnt=<count>`
: Sets the number of keepalives before shutting down the socket to <count> [int].

`keepidle=<seconds>`
: Sets the idle time before sending the first keepalive to <seconds> [int].

`keepintvl=<seconds>`

Sets the interval between two keepalives to <seconds> [int].

linger2=<seconds>
Sets the time to keep the socket in FIN-WAIT-2 state to <seconds> [int].

mss=<bytes>
Sets the MSS (maximum segment size) after the `socket()` call to <bytes> [int]. This value is then proposed to the peer with the SYN or SYN/ACK packet (example).

mss-late=<bytes>
Sets the MSS of the socket after connection has been established to <bytes> [int].

nodelay
Turns off the Nagle algorithm for measuring the RTT (round trip time).

rfc1323
Enables RFC1323 TCP options: TCP window scale, round-trip time measurement (RTTM), and protect against wrapped sequence numbers (PAWS) (AIX).

stdurg
Enables RFC1122 compliant urgent pointer handling (AIX).

syncnt=<count>
Sets the maximal number of SYN retransmits during connect to <count> [int].

md5sig
Enables generation of MD5 digests on the packets (FreeBSD).

noopt
Disables use of TCP options (FreeBSD, MacOSX).

nopush
sets the TCP_NOPUSH socket option (FreeBSD, MacOSX).

sack-disable
Disables use the selective acknowledge feature (OpenBSD).

signature-enable
Enables generation of MD5 digests on the packets (OpenBSD).

abort-threshold=<milliseconds>
Sets the time to wait for an answer of the peer on an established connection (HP-UX).

conn-abort-threshold=<milliseconds>
Sets the time to wait for an answer of the server during the initial connect (HP-UX).

keepinit
Sets the time to wait for an answer of the server during connect() before giving up. Value in half seconds, default is 150 (75s) (Tru64).

paws
Enables the "protect against wrapped sequence numbers" feature (Tru64).

sackena
Enables selective acknowledge (Tru64).

tsoptena
Enables the time stamp option that allows RTT recalculation on existing connections (Tru64).

UDP option group

This option may be applied to UDP datagram sockets.

`udp-ignore-peerport>`
: Address UDP-DATAGRAM expects incoming responses to come from the port specified in its second parameter. With this option, it accepts packets coming from any port.

SCTP option group

These options may be applied to SCTP stream sockets.

`sctp-nodelay`
: Sets the SCTP_NODELAY socket option that disables the Nagle algorithm.

`sctp-maxseg=<bytes>`
: Sets the SCTP_MAXSEG socket option to <bytes> [int]. This value is then proposed to the peer with the SYN or SYN/ACK packet.

UDP, TCP, and SCTP option group

Here we find options that are related to the network port mechanism and thus can be used with UDP, TCP, and SCTP client and server addresses.

`sourceport=<port>`
: For outgoing (client) TCP and UDP connections, it sets the source <port> using an extra `bind()` call. With TCP or UDP listen addresses, socat immediately shuts down the connection if the client does not use this sourceport. UDP-RECV, UDP-RECVFROM, UDP-SENDTO, and UDP-DATAGRAM addresses ignore the packet when it does not match. (example).

`lowport`
: Outgoing (client) TCP and UDP connections with this option use an unused random source port between 640 and 1023 incl. On UNIX class operating systems, this requires root privilege, and thus indicates that the client process is authorized by local root. TCP and UDP listen addresses with this option immediately shut down the connection if the client does not use a sourceport <= 1023. This mechanism can provide limited authorization under some circumstances.

SOCKS option group

When using SOCKS type addresses, some socks specific options can be set.

`socksport=<tcp service>`
: Overrides the default "socks" service or port 1080 for the socks server port with <TCP service>.

`socksuser=<user>`
: Sends the <user> [string] in the username field to the socks server. Default is the actual user name ($LOGNAME or $USER) (example).

HTTP option group

Options that can be provided with HTTP type addresses. The only HTTP address currently implemented is proxy-connect.

`proxyport=<TCP service>`
: Overrides the default HTTP proxy port 8080 with <TCP service>.

`ignorecr`
: The HTTP protocol requires the use of CR+NL as line terminator. When a proxy server violates this standard, socat might not understand its answer. This option directs socat to interprete NL as line terminator and to ignore CR in the answer. Nevertheless, socat sends CR+NL to the proxy.

`proxy-authorization=<username>:<password>`
: Provide "basic" authentication to the proxy server. The argument to the option is used with a "Proxy-Authorization: Basic" header in base64 encoded form.
Note: username and password are visible for every user on the local machine in the process list; username and password are transferred to the proxy server unencrypted (base64 encoded) and might be sniffed.

`proxy-authorization-file=<filename>`
: Like option proxy-authorization, but the credentials are read from the file and therefore not visible in the process list.

`resolve`
: Per default, socat sends to the proxy a CONNECT request containing the target hostname. With this option, socat resolves the hostname locally and sends the IP address. Please note that, according to RFC 2396, only name resolution to IPv4 addresses is implemented.

RANGE option group

These options check if a connecting client should be granted access. They can be applied to listening and receiving network sockets. tcp-wrappers options fall into this group.

`range=<address-range>`
: After accepting a connection, tests if the peer is within *range*. For IPv4 addresses, address-range takes the form address/bits, e.g. 10.0.0.0/8, or address:mask, e.g. 10.0.0.0:255.0.0.0 (example); for IPv6, it is [ip6-address]/bits, e.g. [::1]/128. If the client address does not match, **socat** refuses the connection attempt, issues a warning, and keeps listening/receiving.

`tcpwrap[=<name>]`
: Uses Wietse Venema's libwrap (tcpd) library to determine if the client is allowed to connect. The configuration files are /etc/hosts.allow and /etc/hosts.deny per default, see "man 5 hosts_access" for more information. The optional <name> (type string) is passed to the wrapper functions as daemon process name (example). If omitted, the basename of socats invocation (argv[0]) is passed. If both tcpwrap and range options are applied to an address, both conditions must be fulfilled to allow the connection.

`allow-table=<filename>`
: Takes the specified file instead of /etc/hosts.allow.

`deny-table=<filename>`
: Takes the specified file instead of /etc/hosts.deny.

`tcpwrap-etc=<directoryname>`
: Looks for hosts.allow and hosts.deny in the specified directory. Is overridden by options hosts-allow and hosts-deny.

LISTEN option group

Options specific to listening sockets.

`backlog=<count>`

Sets the backlog value passed with the `listen()` system call to <count> [int]. Default is 5.

`accept-timeout=<seconds>`
End waiting for a connection after <seconds> [timeval] with error status.

`max-children=<count>`
Limits the number of concurrent child processes [int]. Default is no limit.

CHILD option group

Options for addresses with multiple connections via child processes.

`fork`
After establishing a connection, handles its channel in a child process and keeps the parent process attempting to produce more connections, either by listening or by connecting in a loop (example).
OPENSSL-CONNECT and OPENSSL-LISTEN differ in when they actually fork off the child: OPENSSL-LISTEN forks *before* the SSL handshake, while OPENSSL-CONNECT forks *afterwards*. retry and forever options are not inherited by the child process.
On some operating systems (e.g. FreeBSD) this option does not work for UDP-LISTEN addresses.

EXEC option group

Options for addresses that invoke a program.

`path=<string>`
Overrides the PATH environment variable for searching the program with <string>. This `$PATH` value is effective in the child process too.

`login`
Prefixes `argv[0]` for the `execvp()` call with '-', thus making a shell behave as login shell.

FORK option group

EXEC or SYSTEM addresses invoke a program using a child process and transfer data between **socat** and the program. The interprocess communication mechanism can be influenced with the following options. Per default, a `socketpair()` is created and assigned to stdin and stdout of the child process, while stderr is inherited from the **socat** process, and the child process uses file descriptors 0 and 1 for communicating with the main socat process.

`nofork`
Does not fork a subprocess for executing the program, instead calls execvp() or system() directly from the actual socat instance. This avoids the overhead of another process between the program and its peer, but introduces a lot of restrictions:

- this option can only be applied to the second **socat** address.
- it cannot be applied to a part of a dual address.
- the first socat address cannot be OPENSSL or READLINE
- socat options -b, -t, -D, -l, -v, -x become useless
- for both addresses, options ignoreeof, cr, and crnl become useless
- for the second address (the one with option nofork), options append, cloexec, flock, user, group, mode, nonblock, perm-late, setlk, and setpgid cannot be applied. Some of these could be used on the first address though.

`pipes`

Creates a pair of unnamed pipes for interprocess communication instead of a socket pair.

`openpty`
: Establishes communication with the sub process using a pseudo terminal created with `openpty()` instead of the default (socketpair or ptmx).

`ptmx`
: Establishes communication with the sub process using a pseudo terminal created by opening **/dev/ptmx** or **/dev/ptc** instead of the default (socketpair).

`pty`
: Establishes communication with the sub process using a pseudo terminal instead of a socket pair. Creates the pty with an available mechanism. If openpty and ptmx are both available, it uses ptmx because this is POSIX compliant (example).

`ctty`
: Makes the pty the controlling tty of the sub process (example).

`stderr`
: Directs stderr of the sub process to its output channel by making stderr a `dup()` of stdout (example).

`fdin=<fdnum>`
: Assigns the sub processes input channel to its file descriptor <fdnum> instead of stdin (0). The program started from the subprocess has to use this fd for reading data from **socat** (example).

`fdout=<fdnum>`
: Assigns the sub processes output channel to its file descriptor <fdnum> instead of stdout (1). The program started from the subprocess has to use this fd for writing data to **socat** (example).

`sighup`, `sigint`, `sigquit`
: Has **socat** pass signals of this type to the sub process. If no address has this option, socat terminates on these signals.

TERMIOS option group

For addresses that work on a tty (e.g., stdio, file:/dev/tty, exec:...,pty), the terminal parameters defined in the UN*X termios mechanism are made available as address option parameters. Please note that changes of the parameters of your interactive terminal remain effective after **socat**'s termination, so you might have to enter "reset" or "stty sane" in your shell afterwards. For EXEC and SYSTEM addresses with option PTY, these options apply to the pty by the child processes.

`b0`
: Disconnects the terminal.

`b19200`
: Sets the serial line speed to 19200 baud. Some other rates are possible; use something like `socat -hh |grep ' b[1-9]'` to find all speeds supported by your implementation.
Note: On some operating systems, these options may not be available. Use ispeed or ospeed instead.

`echo=<bool>`
: Enables or disables local echo.

`icanon=<bool>`
: Sets or clears canonical mode, enabling line buffering and some special characters.

`raw`
: Sets raw mode, thus passing input and output almost unprocessed. This option is obsolete, use option rawer or cfmakeraw instead.

`rawer`
: Makes terminal rawer than raw option. This option implicitly turns off echo. (example).

`cfmakeraw`
: Sets raw mode by invoking `cfmakeraw()` or by simulating this call. This option implicitly turns off echo.

`ignbrk=<bool>`
: Ignores or interpretes the BREAK character (e.g., ^C)

`brkint=<bool>`

`bs0`

`bs1`

`bsdly=<0|1>`

`clocal=<bool>`

`cr0`
`cr1`
`cr2`
`cr3`
: Sets the carriage return delay to 0, 1, 2, or 3, respectively. 0 means no delay, the other values are terminal dependent.

`crdly=<0|1|2|3>`

`cread=<bool>`

`crtscts=<bool>`

`cs5`
`cs6`
`cs7`
`cs8`
: Sets the character size to 5, 6, 7, or 8 bits, respectively.

`csize=<0|1|2|3>`

`cstopb=<bool>`
: Sets two stop bits, rather than one.

`dsusp=<byte>`
: Sets the value for the VDSUSP character that suspends the current foreground process and reactivates the shell (all except Linux).

`echoctl=<bool>`
: Echos control characters in hat notation (e.g. ^A)

`echoe=<bool>`

`echok=<bool>`

`echoke=<bool>`

`echonl=<bool>`

`echoprt=<bool>`

`eof=<byte>`

`eol=<byte>`

`eol2=<byte>`

`erase=<byte>`

`discard=<byte>`

`ff0`

`ff1`

`ffdly=<bool>`

`flusho=<bool>`

`hupcl=<bool>`

`icrnl=<bool>`

`iexten=<bool>`

`igncr=<bool>`

`ignpar=<bool>`

`imaxbel=<bool>`

`inlcr=<bool>`

`inpck=<bool>`

`intr=<byte>`

`isig=<bool>`

`ispeed=<unsigned-int>`
Set the baud rate for incoming data on this line.
See also: ospeed, b19200

`istrip=<bool>`

`iuclc=<bool>`

`ixany=<bool>`

`ixoff=<bool>`

`ixon=<bool>`

`kill=<byte>`

`lnext=<byte>`

`min=<byte>`

`nl0`
Sets the newline delay to 0.

`nl1`

`nldly=<bool>`

`noflsh=<bool>`

`ocrnl=<bool>`

`ofdel=<bool>`

`ofill=<bool>`

`olcuc=<bool>`

`onlcr=<bool>`

`onlret=<bool>`

`onocr=<bool>`

`opost=<bool>`
: Enables or disables output processing; e.g., converts NL to CR-NL.

`ospeed=<unsigned-int>`
: Set the baud rate for outgoing data on this line.
See also: ispeed, b19200

`parenb=<bool>`
: Enable parity generation on output and parity checking for input.

`parmrk=<bool>`

`parodd=<bool>`

`pendin=<bool>`

`quit=<byte>`

`reprint=<byte>`

`sane`
: Brings the terminal to something like a useful default state.

`start=<byte>`

`stop=<byte>`

`susp=<byte>`

`swtc=<byte>`

`tab0`

`tab1`

`tab2`

`tab3`

`tabdly=<unsigned-int>`

`time=<byte>`

`tostop=<bool>`

`vt0`

`vt1`

`vtdly=<bool>`

`werase=<byte>`

`xcase=<bool>`

`xtabs`

`i-pop-all`
: With UNIX System V STREAMS, removes all drivers from the stack.

`i-push=<string>`
: With UNIX System V STREAMS, pushes the driver (module) with the given name (string) onto the stack. For example, to make sure that a character device on Solaris supports termios etc, use the following options: `i-pop-all,i-push=ptem,i-push=ldterm,i-push=ttcompat`

PTY option group

These options are intended for use with the pty address type.

`link=<filename>`
: Generates a symbolic link that points to the actual pseudo terminal (pty). This might help to solve the problem that ptys are generated with more or less unpredictable names, making it difficult to directly access the socat generated pty automatically. With this option, the user can specify a "fix" point in the file hierarchy that helps him to access the actual pty (example). Beginning with **socat** version 1.4.3, the symbolic link is removed when the address is closed (but see option unlink-close).

`wait-slave`
: Blocks the open phase until a process opens the slave side of the pty. Usually, socat continues after generating the pty with opening the next address or with entering the transfer loop. With the wait-slave option, socat waits until some process opens the slave side of the pty before continuing. This option only works if the operating system provides the `poll()` system call. And it depends on an undocumented behaviour of pty's, so it does not work on all operating systems. It has successfully been tested on Linux, FreeBSD, NetBSD, and on Tru64 with openpty.

`pty-interval=<seconds>`
: When the wait-slave option is set, socat periodically checks the HUP condition using `poll()` to find if the pty's slave side has been opened. The default polling interval is 1s. Use the pty-interval option [timeval] to change this value.

OPENSSL option group

These options apply to the openssl and openssl-listen address types.

`cipher=<cipherlist>`
: Selects the list of ciphers that may be used for the connection. See the man page of `ciphers` , section **CIPHER LIST FORMAT**, for detailed information about syntax, values, and default of <cipherlist>.
Several cipher strings may be given, separated by ':'. Some simple cipher strings:

3DES
: Uses a cipher suite with triple DES.

MD5
: Uses a cipher suite with MD5.

aNULL
: Uses a cipher suite without authentication.

NULL
: Does not use encryption.

HIGH
Uses a cipher suite with "high" encryption.

Note that the peer must support the selected property, or the negotiation will fail.

`method=<ssl-method>`
This option is based on deprecated functions and is only available when **socat** was build with option `--with-openssl-method`. Sets the protocol version to be used. Valid strings (not case sensitive) are:

`SSL2`
Select SSL protocol version 2.

`SSL3`
Select SSL protocol version 3.

`SSL23`
Select the best available SSL or TLS protocol.

`TLS1`
Select TLS protocol version 1.

`TLS1.1`
Select TLS protocol version 1.1.

`TLS1.2`
Select TLS protocol version 1.2. When this option is not provided OpenSSL negotiates the mothod with its peer.

`verify=<bool>`
Controls check of the peer's certificate. Default is 1 (true). Disabling verify might open your socket for everyone, making the encryption useless!

`cert=<filename>`
Specifies the file with the certificate and private key for authentication. The certificate must be in OpenSSL format (*.pem). With openssl-listen, use of this option is strongly recommended. Except with cipher aNULL, "no shared ciphers" error will occur when no certificate is given.

`key=<filename>`
Specifies the file with the private key. The private key may be in this file or in the file given with the cert option. The party that has to proof that it is the owner of a certificate needs the private key.

`dhparams=<filename>`
Specifies the file with the Diffie Hellman parameters. These parameters may also be in the file given with the cert option in which case the dhparams option is not needed.

`cafile=<filename>`
Specifies the file with the trusted (root) authority certificates. The file must be in PEM format and should contain one or more certificates. The party that checks the authentication of its peer trusts only certificates that are in this file.

`capath=<dirname>`
Specifies the directory with the trusted (root) certificates. The directory must contain certificates in PEM format and their hashes (see OpenSSL documentation)

`egd=<filename>`
On some systems, openssl requires an explicit source of random data. Specify the socket name where an entropy gathering daemon like egd provides random data, e.g. /dev/egd-pool.

`pseudo`

On systems where openssl cannot find an entropy source and where no entropy gathering daemon can be utilized, this option activates a mechanism for providing pseudo entropy. This is achieved by taking the current time in microseconds for feeding the libc pseudo random number generator with an initial value. openssl is then feeded with output from random() calls.
NOTE:This mechanism is not sufficient for generation of secure keys!

`compress`
Enable or disable the use of compression for a connection. Setting this to "none" disables compression, setting it to "auto" lets OpenSSL choose the best available algorithm supported by both parties. The default is to not touch any compression-related settings. NOTE: Requires OpenSSL 0.9.8 or higher and disabling compression with OpenSSL 0.9.8 affects all new connections in the process.

`commonname=<string>`
Specify the commonname that the peer certificate must match. With OPENSSL-CONNECT address this overrides the given hostname or IP target address; with OPENSSL-LISTEN this turns on check of peer certificates commonname. This option has only meaning when option verify is not disabled and the chosen cipher provides a peer certificate.

`no-sni=<bool>`
Do not use the client side Server Name Indication (SNI) feature that selects the desired server certificate.
Note: SNI is automatically used since **socat** version 1.7.4.0 and uses commonname or the given host name.

`snihost=<string>`
Set the client side Server Name Indication (SNI) host name different from the addressed server name or common name. This might be useful when the server certificate has multiple host names or wildcard names because the SNI host name is passed in cleartext to the server and might be eavesdropped; with this option a mock name of the desired certificate may be transferred.

`fips`
Enables FIPS mode if compiled in. For info about the FIPS encryption implementation standard see http://oss-institute.org/fips-faq.html. This mode might require that the involved certificates are generated with a FIPS enabled version of openssl. Setting or clearing this option on one socat address affects all OpenSSL addresses of this process.

RETRY option group

Options that control retry of some system calls, especially connection attempts.

`retry=<num>`
Number of retries before the connection or listen attempt is aborted. Default is 0, which means just one attempt.

`interval=<timespec>`
Time between consecutive attempts (seconds, [timespec]). Default is 1 second.

`forever`
Performs an unlimited number of retry attempts.

TUN option group

Options that control Linux TUN/TAP interface device addresses.

`tun-device=<device-file>`
Instructs socat to take another path for the TUN clone device. Default is `/dev/net/tun`.

`tun-name=<if-name>`

Gives the resulting network interface a specific name instead of the system generated (tun0, tun1, etc.)

`tun-type=[tun|tap]`
Sets the type of the TUN device; use this option to generate a TAP device. See the Linux docu for the difference between these types. When you try to establish a tunnel between two TUN devices, their types should be the same.

`iff-no-pi`
Sets the IFF_NO_PI flag which controls if the device includes additional packet information in the tunnel. When you try to establish a tunnel between two TUN devices, these flags should have the same values.

`iff-up`
Sets the TUN network interface status UP. Strongly recommended.

`iff-broadcast`
Sets the BROADCAST flag of the TUN network interface.

`iff-debug`
Sets the DEBUG flag of the TUN network interface.

`iff-loopback`
Sets the LOOPBACK flag of the TUN network interface.

`iff-pointopoint`
Sets the POINTOPOINT flag of the TUN device.

`iff-notrailers`
Sets the NOTRAILERS flag of the TUN device.

`iff-running`
Sets the RUNNING flag of the TUN device.

`iff-noarp`
Sets the NOARP flag of the TUN device.

`iff-promisc`
Sets the PROMISC flag of the TUN device.

`iff-allmulti`
Sets the ALLMULTI flag of the TUN device.

`iff-master`
Sets the MASTER flag of the TUN device.

`iff-slave`
Sets the SLAVE flag of the TUN device.

`iff-multicast`
Sets the MULTICAST flag of the TUN device.

`iff-portsel`
Sets the PORTSEL flag of the TUN device.

`iff-automedia`
Sets the AUTOMEDIA flag of the TUN device.

`iff-dynamic`
Sets the DYNAMIC flag of the TUN device.

DATA VALUES

This section explains the different data types that address parameters and address options can take.

address-range
Is currently only implemented for IPv4 and IPv6. See address-option `range'

bool
"0" or "1"; if value is omitted, "1" is taken.

byte
An unsigned int number, read with `strtoul()` , lower or equal to `UCHAR_MAX` .

command-line
A string specifying a program name and its arguments, separated by single spaces.

data
This is a more general data specification. The given text string contains information about the target data type and value. Generally a leading character specifies the type of the following data item. In its specific context a default data type may exist.
Currently only the following specifications are implemented:

i
A signed integer number, stored in host byte order.
Example: **i-1000** (Integer number -1000)

I
An unsigned integer number, stored in host byte order.

l
A signed long integer number, stored in host byte order.

L
An unsigned long integer number, stored in host byte order.

s
A signed short integer number, stored in host byte order.

S
An unsigned short integer number, stored in host byte order.

b
A signed byte (signed char).

B
An unsigned byte (unsigned char).

x
Following is an even number of hex digits, stored as sequence of bytes.
Example: **x7f000001** (IP address 127.0.0.1)

"
Following is a string that is used with the common conversions \n \r \t \f \b \a \e \0; the string must be closed with '"'. Please note that the quotes and backslashes need to be escaped from shell and **socat** conversion.
Example: **"Hello world!\n"**

'

A single char, with the usual conversions. Please note that the quotes and backslashes need to be escaped from shell and **socat** conversion.
Example: **'a'**

Data items may be separated with white space without need to repeat the type specifier again.

directory
A string with usual UN*X directory name semantics.

facility
The name of a syslog facility in lower case characters.

fdnum
An unsigned int type, read with `strtoul()` , specifying a UN*X file descriptor.

filename
A string with usual UN*X filename semantics.

group
If the first character is a decimal digit, the value is read with `strtoul()` as unsigned integer specifying a group id. Otherwise, it must be an existing group name.

int
A number following the rules of the `strtol()` function with base "0", i.e. decimal number, octal number with leading "0", or hexadecimal number with leading "0x". The value must fit into a C int.

interface
A string specifying the device name of a network interface as shown by ifconfig or procan, e.g. "eth0".

IP address
An IPv4 address in numbers-and-dots notation, an IPv6 address in hex notation enclosed in brackets, or a hostname that resolves to an IPv4 or an IPv6 address.
Examples: 127.0.0.1, [::1], www.dest-unreach.org, dns1

IPv4 address
An IPv4 address in numbers-and-dots notation or a hostname that resolves to an IPv4 address.
Examples: 127.0.0.1, www.dest-unreach.org, dns2

IPv6 address
An IPv6 address in hexnumbers-and-colons notation enclosed in brackets, or a hostname that resolves to an IPv6 address.
Examples: [::1], [1234:5678:9abc:def0:1234:5678:9abc:def0], ip6name.domain.org

long
A number read with `strtol()` . The value must fit into a C long.

long long
A number read with `strtoll()` . The value must fit into a C long long.

off_t
An implementation dependend signed number, usually 32 bits, read with strtol or strtoll.

off64_t
An implementation dependend signed number, usually 64 bits, read with strtol or strtoll.

mode_t
An unsigned integer, read with `strtoul()` , specifying mode (permission) bits.

pid_t
: A number, read with `strtol()` , specifying a process id.

port
: A uint16_t (16 bit unsigned number) specifying a TCP or UDP port, read with `strtoul()` .

protocol
: An unsigned 8 bit number, read with `strtoul()` .

size_t
: An unsigned number with size_t limitations, read with `strtoul` .

sockname
: A socket address. See address-option `bind'

string
: A sequence of characters, not containing '\0' and, depending on the position within the command line, ':', ',', or "!!". Note that you might have to escape shell meta characters in the command line.

TCP service
: A service name, not starting with a digit, that is resolved by `getservbyname()` , or an unsigned int 16 bit number read with `strtoul()` .

timeval
: A double float specifying seconds; the number is mapped into a struct timeval, consisting of seconds and microseconds.

timespec
: A double float specifying seconds; the number is mapped into a struct timespec, consisting of seconds and nanoseconds.

UDP service
: A service name, not starting with a digit, that is resolved by `getservbyname()` , or an unsigned int 16 bit number read with `strtoul()` .

unsigned int
: A number read with `strtoul()` . The value must fit into a C unsigned int.

user
: If the first character is a decimal digit, the value is read with `strtoul()` as unsigned integer specifying a user id. Otherwise, it must be an existing user name.

VSOCK cid
: A uint32_t (32 bit unsigned number) specifying a VSOCK Context Identifier (CID), read with `strtoul()` . There are several special addresses: VMADDR_CID_ANY (-1U) means any address for binding; VMADDR_CID_HOST (2) is the well-known address of the host.

VSOCK port
: A uint32_t (32 bit unsigned number) specifying a VSOCK port, read with `strtoul()` .

EXAMPLES

```
socat - TCP4:www.domain.org:80
```

transfers data between STDIO (-) and a TCP4 connection to port 80 of host www.domain.org. This example results in an interactive connection similar to telnet or netcat. The stdin terminal parameters are not changed, so you may close the relay with ^D or abort it with ^C.

```
socat -d -d READLINE,history=$HOME/.http_history \
TCP4:www.domain.org:www,crnl
```

this is similar to the previous example, but you can edit the current line in a bash like manner (READLINE) and use the history file .http_history; **socat** prints messages about progress (-d -d). The port is specified by service name (www), and correct network line termination characters (crnl) instead of NL are used.

```
socat TCP4-LISTEN:www TCP4:www.domain.org:www
```

installs a simple TCP port forwarder. With TCP4-LISTEN it listens on local port "www" until a connection comes in, accepts it, then connects to the remote host (TCP4) and starts data transfer. It will not accept a second connection.

```
socat -d -d -lmlocal2 \
TCP4-LISTEN:80,bind=myaddr1,su=nobody,fork,range=10.0.0.0/8,reuseaddr \
TCP4:www.domain.org:80,bind=myaddr2
```

TCP port forwarder, each side bound to another local IP address (bind). This example handles an almost arbitrary number of parallel or consecutive connections by fork'ing a new process after each `accept()` . It provides a little security by su'ing to user nobody after forking; it only permits connections from the private 10 network (range); due to reuseaddr, it allows immediate restart after master process's termination, even if some child sockets are not completely shut down. With -lmlocal2, socat logs to stderr until successfully reaching the accept loop. Further logging is directed to syslog with facility local2.

```
socat TCP4-LISTEN:5555,fork,tcpwrap=script \
EXEC:/bin/myscript,chroot=/home/sandbox,su-d=sandbox,pty,stderr
```

a simple server that accepts connections (TCP4-LISTEN) and fork's a new child process for each connection; every child acts as single relay. The client must match the rules for daemon process name "script" in /etc/hosts.allow and /etc/hosts.deny, otherwise it is refused access (see "man 5 hosts_access"). For EXEC'uting the program, the child process chroot's to **/home/sandbox**, su's to user sandbox, and then starts the program **/home/sandbox/bin/myscript**. **Socat** and myscript communicate via a pseudo tty (pty); myscript's stderr is redirected to stdout, so its error messages are transferred via **socat** to the connected client.

```
socat EXEC:"mail.sh target@domain.com",fdin=3,fdout=4 \
TCP4:mail.relay.org:25,crnl,bind=alias1.server.org,mss=512
```

mail.sh is a shell script, distributed with **socat**, that implements a simple SMTP client. It is programmed to "speak" SMTP on its FDs 3 (in) and 4 (out). The fdin and fdout options tell **socat** to use these FDs for communication with the program. Because mail.sh inherits stdin and stdout while **socat** does not use them, the script can read a mail body from stdin. **Socat** makes alias1 your local source address (bind), cares for correct network line termination (crnl) and sends at most 512 data bytes per packet (mss).

```
socat -,escape=0x0f /dev/ttyS0,rawer,crnl
```

opens an interactive connection via the serial line, e.g. for talking with a modem. rawer sets the console's and ttyS0's terminal parameters to practicable values, crnl converts to correct newline characters. escape allows terminating the socat process with character control-O. Consider using READLINE instead of the first address.

```
socat UNIX-LISTEN:/tmp/.X11-unix/X1,fork \
SOCKS4:host.victim.org:127.0.0.1:6000,socksuser=nobody,sourceport=20
```

with UNIX-LISTEN, **socat** opens a listening UNIX domain socket **/tmp/.X11-unix/X1**. This path corresponds to local XWindow display :1 on your machine, so XWindow client connections to DISPLAY=:1 are accepted. **Socat** then speaks with the SOCKS4 server host.victim.org that might permit sourceport 20 based connections due to an FTP related weakness in its static IP filters. **Socat** pretends to be invoked by socksuser nobody, and requests to be connected to loopback port 6000 (only weak sockd configurations will allow this). So we get a connection to the victims XWindow server and, if it does not require MIT cookies or Kerberos authentication, we can start work. Please note that there can only be one connection at a time, because TCP can establish only one session with a given set of addresses and ports.

```
socat -u /tmp/readdata,seek-end=0,ignoreeof -
```

this is an example for unidirectional data transfer (-u). **Socat** transfers data from file /tmp/readdata (implicit address GOPEN), starting at its current end (seek-end=0 lets **socat** start reading at current end of file; use seek=0 or no seek option to first read the existing data) in a "tail -f" like mode (ignoreeof). The "file" might also be a listening UNIX domain socket (do not use a seek option then).

```
(sleep 5; echo PASSWORD; sleep 5; echo ls; sleep 1) |
socat - EXEC:'ssh -l user server',pty,setsid,ctty
```

EXEC'utes an ssh session to server. Uses a pty for communication between **socat** and ssh, makes it ssh's controlling tty (ctty), and makes this pty the owner of a new process group (setsid), so ssh accepts the password from **socat**.

```
socat -u TCP4-LISTEN:3334,reuseaddr,fork \
OPEN:/tmp/in.log,creat,append
```

implements a simple network based message collector. For each client connecting to port 3334, a new child process is generated (option fork). All data sent by the clients are append'ed to the file /tmp/in.log. If the file does not exist, socat creat's it. Option reuseaddr allows immediate restart of the server process.

```
socat READLINE,noecho='[Pp]assword:' EXEC:'ftp ftp.server.com',pty,setsid,ctty
```

wraps a command line history (READLINE) around the EXEC'uted ftp client utility. This allows editing and reuse of FTP commands for relatively comfortable browsing through the ftp directory hierarchy. The password is echoed! pty is required to have ftp issue a prompt. Nevertheless, there may occur some confusion with the password and FTP prompts.

```
socat PTY,link=$HOME/dev/vmodem0,rawer,wait-slave \
EXEC:'"ssh modemserver.us.org socat - /dev/ttyS0,nonblock,rawer"'
```

generates a pseudo terminal device (PTY) on the client that can be reached under the symbolic link **$HOME/dev/vmodem0**. An application that expects a serial line or modem can be configured to use **$HOME/dev/vmodem0**; its traffic will be directed to a modemserver via ssh where another socat instance links it to **/dev/ttyS0**.

```
socat TCP4-LISTEN:2022,reuseaddr,fork \
PROXY:proxy:www.domain.org:22,proxyport=3128,proxyauth=user:pass
```

starts a forwarder that accepts connections on port 2022, and directs them through the proxy daemon listening on port 3128 (proxyport) on host proxy, using the CONNECT method, where they are authenticated as "user" with "pass" (proxyauth). The proxy should establish connections to host www.domain.org on port 22 then.

```
socat - SSL:server:4443,cafile=server.crt,cert=client.pem
```

is an OpenSSL client that tries to establish a secure connection to an SSL server. Option cafile specifies a file that contains trust certificates: we trust the server only when it presents one of these certificates and proofs that it owns the related private key. Otherwise the connection is terminated. With cert a file containing the client certificate and the associated private key is specified. This is required in case the server wishes a client authentication; many Internet servers do not.
The first address ('-') can be replaced by almost any other socat address.

```
socat OPENSSL-LISTEN:4443,reuseaddr,pf=ip4,fork,cert=server.pem,cafile=client.crt PIPE
```

is an OpenSSL server that accepts TCP connections, presents the certificate from the file server.pem and forces the client to present a certificate that is verified against cafile.crt.
The second address ('PIPE') can be replaced by almost any other socat address.

For instructions on generating and distributing OpenSSL keys and certificates see the additional socat docu `socat-openssl.txt`.

`echo |socat -u - file:/tmp/bigfile,create,largefile,seek=100000000000`

creates a 100GB sparse file; this requires a file system type that supports this (ext2, ext3, reiserfs, jfs; not minix, vfat). The operation of writing 1 byte might take long (reiserfs: some minutes; ext2: "no" time), and the resulting file can consume some disk space with just its inodes (reiserfs: 2MB; ext2: 16KB).

`socat tcp-l:7777,reuseaddr,fork system:'filan -i 0 -s >&2',nofork`

listens for incoming TCP connections on port 7777. For each accepted connection, invokes a shell. This shell has its stdin and stdout directly connected to the TCP socket (nofork). The shell starts filan and lets it print the socket addresses to stderr (your terminal window).

`echo -e "\0\14\0\0\c" |socat -u - file:/usr/bin/squid.exe,seek=0x00074420`

functions as primitive binary editor: it writes the 4 bytes 000 014 000 000 to the executable /usr/bin/squid at offset 0x00074420 (this is a real world patch to make the squid executable from Cygwin run under Windows, actual per May 2004).

`socat - tcp:www.blackhat.org:31337,readbytes=1000`

connects to an unknown service and prevents being flooded.

`socat -U TCP:target:9999,end-close TCP-L:8888,reuseaddr,fork`

merges data arriving from different TCP streams on port 8888 to just one stream to target:9999. The end-close option prevents the child processes forked off by the second address from terminating the shared connection to 9999 (close(2) just unlinks the inode which stays active as long as the parent process lives; shutdown(2) would actively terminate the connection).

`socat - UDP4-DATAGRAM:192.168.1.0:123,sp=123,broadcast,range=192.168.1.0/24`

sends a broadcast to the network 192.168.1.0/24 and receives the replies of the timeservers there. Ignores NTP packets from hosts outside this network.

`socat - SOCKET-DATAGRAM:2:2:17:x007bxc0a80100x0000000000000000,bind=x007bx00000000x0000000000000000,setsockopt-int=1:6:1,range=x0000xc0a80100x0000000000000000:x0000xffffff00x0000000000000000`

is semantically equivalent to the previous example, but all parameters are specified in generic form. the value 6 of setsockopt-int is the Linux value for `SO_BROADCAST`.

`socat - IP4-DATAGRAM:255.255.255.255:44,broadcast,range=10.0.0.0/8`

sends a broadcast to the local network(s) using protocol 44. Accepts replies from the private address range only.

`socat - UDP4-DATAGRAM:224.255.0.1:6666,bind=:6666,ip-add-membership=224.255.0.1:eth0`

transfers data from stdin to the specified multicast address using UDP. Both local and remote ports are 6666. Tells the interface eth0 to also accept multicast packets of the given group. Multiple hosts on the local network can run this command, so all data sent by any of the hosts will be received by all the other ones. Note that there are many possible reasons for failure, including IP-filters, routing issues, wrong interface selection by the operating system, bridges, or a badly configured switch.

`socat UDP:host2:4443 TUN:192.168.255.1/24,up`

establishes one side of a virtual (but not private!) network with host2 where a similar process might run, with UDP-L and tun address 192.168.255.2. They can reach each other using the addresses 192.168.255.1 and

192.168.255.2. Note that streaming eg.via TCP or SSL does not guarantee to retain packet boundaries and might thus cause packet loss.

```
socat - VSOCK-CONNECT:2:1234
```

establishes a VSOCK connection with the host (host is always reachable with the well-know CID=2) on 1234 port.

```
socat - VSOCK-LISTEN:1234
```

listens for a VSOCK connection on 1234 port.

```
socat - VSOCK-CONNECT:31:4321,bind:5555
```

establishes a VSOCK connection with the guest that have CID=31 on 1234 port, binding the local socket to the 5555 port.

```
socat VSOCK-LISTEN:3333,reuseaddr,fork VSOCK-CONNECT:42,3333
```

starts a forwarder that accepts VSOCK connections on port 3333, and directs them to the guest with CID=42 on the same port.

```
socat VSOCK-LISTEN:22,reuseaddr,fork TCP:localhost:22
```

forwards VSOCK connections from 22 port to the local SSH server. Running this in a VM allows you to connect via SSH from the host using VSOCK, as in the example below.

```
socat TCP4-LISTEN:22222,reuseaddr,fork VSOCK-CONNECT:33:22
```

forwards TCP connections from 22222 port to the guest with CID=33 listening on VSOCK port 22. Running this in the host, allows you to connect via SSH running "ssh -p 22222 user@localhost", if the guest runs the example above.

```
socat PTY,link=/var/run/ppp,rawer INTERFACE:hdlc0
```

circumvents the problem that pppd requires a serial device and thus might not be able to work on a synchronous line that is represented by a network device. socat creates a PTY to make pppd happy, binds to the network interface `hdlc0`, and can transfer data between both devices. Use pppd on device `/var/run/ppp` then.

```
socat -T 1 -d -d TCP-L:10081,reuseaddr,fork,crlf SYSTEM:"echo -e \"\\\"HTTP/1.0 200
OK\\\nDocumentType: text/plain\\\n\\\ndate: \$\
(date\)\\\nserver:\$SOCAT_SOCKADDR:\$SOCAT_SOCKPORT\\\nclient:
\$SOCAT_PEERADDR:\$SOCAT_PEERPORT\\\n\\\"\"; cat; echo -e \"\\\"\\\n\\\"\""
```

creates a simple HTTP echo server: each HTTP client that connects gets a valid HTTP reply that contains information about the client address and port as it is seen by the server host, the host address (which might vary on multihomed servers), and the original client request.

```
socat -d -d UDP4-RECVFROM:9999,so-broadcast,so-timestamp,ip-pktinfo,ip-recverr,ip-recvopts,ip-
recvtos,ip-recvttl!!- SYSTEM:'export; sleep 1' |grep SOCAT
```

waits for an incoming UDP packet on port 9999 and prints the environment variables provided by socat. On BSD based systems you have to replace `ip-pktinfo` with `ip-recvdstaddr`,`ip-recvif`. Especially interesting is SOCAT_IP_DSTADDR: it contains the target address of the packet which may be a unicast, multicast, or broadcast address.

```
echo -e "M-SEARCH * HTTP/1.1\nHOST: 239.255.255.250:1900\nMAN: \"ssdp:discover\"\nMX: 4\nST:
\"ssdp:all\"\n" |./socat - UDP-DATAGRAM:239.255.255.250:1900,crlf
```

sends an SSDP (Simple Service Discovery Protocol) query to the local network and collects and outputs the answers received.

DIAGNOSTICS

Socat uses a logging mechanism that allows filtering messages by severity. The severities provided are more or less compatible to the appropriate syslog priority. With one or up to four occurrences of the -d command line option, the lowest priority of messages that are issued can be selected. Each message contains a single uppercase character specifying the messages severity (one of F, E, W, N, I, or D)

FATAL:
Conditions that require unconditional and immediate program termination.

ERROR:
Conditions that prevent proper program processing. Usually the program is terminated (see option -s).

WARNING:
Something did not function correctly or is in a state where correct further processing cannot be guaranteed, but might be possible.

NOTICE:
Interesting actions of the program, e.g. for supervising **socat** in some kind of server mode.

INFO:
Description of what the program does, and maybe why it happens. Allows monitoring the lifecycles of file descriptors.

DEBUG:
Description of how the program works, all system or library calls and their results.

Log messages can be written to stderr, to a file, or to syslog.

On exit, **socat** gives status 0 if it terminated due to EOF or inactivity timeout, with a positive value on error, and with a negative value on fatal error.

FILES

/usr/bin/socat
/usr/bin/filan
/usr/bin/procan

ENVIRONMENT VARIABLES

Input variables carry information from the environment to socat, output variables are set by socat for use in executed scripts and programs.

In the output variables beginning with "SOCAT" this prefix is actually replaced by the upper case name of the executable or the value of option -lp.

SOCAT_DEFAULT_LISTEN_IP (input)
(Values 4 or 6) Sets the IP version to be used for listen, recv, and recvfrom addresses if no pf (protocol-family) option is given. Is overridden by socat options -4 or -6.

SOCAT_PREFERRED_RESOLVE_IP (input)
(Values 0, 4, or 6) Sets the IP version to be used when resolving target host names when version is not specified by address type, option pf (protocol-family), or address format. If name resolution does not return a matching entry, the first result (with differing IP version) is taken. With value 0, socat always selects the first record and its IP version.

SOCAT_FORK_WAIT (input)
Specifies the time (seconds) to sleep the parent and child processes after successful fork(). Useful for debugging.

SOCAT_VERSION (output)
Socat sets this variable to its version string, e.g. `"1.7.0.0"` for released versions or e.g. `"1.6.0.1+envvar"` for temporary versions; can be used in scripts invoked by socat.

SOCAT_PID (output)
Socat sets this variable to its process id. In case of fork address option, SOCAT_PID gets the child processes id. Forking for exec and system does not change SOCAT_PID.

SOCAT_PPID (output)
Socat sets this variable to its process id. In case of fork, SOCAT_PPID keeps the pid of the master process.

SOCAT_PEERADDR (output)
With passive socket addresses (all LISTEN and RECVFROM addresses), this variable is set to a string describing the peers socket address. Port information is not included.

SOCAT_PEERPORT (output)
With appropriate passive socket addresses (TCP, UDP, and SCTP - LISTEN and RECVFROM), this variable is set to a string containing the number of the peer port.

SOCAT_SOCKADDR (output)
With all LISTEN addresses, this variable is set to a string describing the local socket address. Port information is not included example

SOCAT_SOCKPORT (output)
With TCP-LISTEN, UDP-LISTEN, and SCTP-LISTEN addresses, this variable is set to the local port.

SOCAT_TIMESTAMP (output)
With all RECVFROM addresses where address option so-timestamp is applied, socat sets this variable to the resulting timestamp.

SOCAT_IP_OPTIONS (output)
With all IPv4 based RECVFROM addresses where address option ip-recvopts is applied, socat fills this variable with the IP options of the received packet.

SOCAT_IP_DSTADDR (output)
With all IPv4 based RECVFROM addresses where address option ip-recvdstaddr (BSD) or ip-pktinfo (other platforms) is applied, socat sets this variable to the destination address of the received packet. This is particularly useful to identify broadcast and multicast addressed packets.

SOCAT_IP_IF (output)
With all IPv4 based RECVFROM addresses where address option ip-recvif (BSD) or ip-pktinfo (other platforms) is applied, socat sets this variable to the name of the interface where the packet was received.

SOCAT_IP_LOCADDR (output)
With all IPv4 based RECVFROM addresses where address option ip-pktinfo is applied, socat sets this variable to the address of the interface where the packet was received.

SOCAT_IP_TOS (output)
With all IPv4 based RECVFROM addresses where address option ip-recvtos is applied, socat sets this variable to the TOS (type of service) of the received packet.

SOCAT_IP_TTL (output)
With all IPv4 based RECVFROM addresses where address option ip-recvttl is applied, socat sets this variable to the TTL (time to live) of the received packet.

SOCAT_IPV6_HOPLIMIT (output)
With all IPv6 based RECVFROM addresses where address option ipv6-recvhoplimit is applied, socat sets this variable to the hoplimit value of the received packet.

SOCAT_IPV6_DSTADDR (output)
With all IPv6 based RECVFROM addresses where address option ipv6-recvpktinfo is applied, socat sets this variable to the destination address of the received packet.

SOCAT_IPV6_TCLASS (output)
With all IPv6 based RECVFROM addresses where address option ipv6-recvtclass is applied, **socat** sets this variable to the transfer class of the received packet.

SOCAT_OPENSSL_X509_ISSUER (output)
Issuer field from peer certificate

SOCAT_OPENSSL_X509_SUBJECT (output)
Subject field from peer certificate

SOCAT_OPENSSL_X509_COMMONNAME (output)
commonName entries from peer certificates subject. Multiple values are separated by " // ".

SOCAT_OPENSSL_X509_* (output)
all other entries from peer certificates subject

SOCAT_OPENSSL_X509V3_DNS (output)
DNS entries from peer certificates extensions - subjectAltName field. Multiple values are separated by " // ".

HOSTNAME (input)
Is used to determine the hostname for logging (see -lh).

LOGNAME (input)
Is used as name for the socks client user name if no socksuser is given.
With options su and su-d, LOGNAME is set to the given user name.

USER (input)
Is used as name for the socks client user name if no socksuser is given and LOGNAME is empty.
With options su and su-d, USER is set to the given user name.

SHELL (output)
With options su and su-d, SHELL is set to the login shell of the given user.

PATH (output)
Can be set with option path for exec and system addresses.

HOME (output)
With options su and su-d, HOME is set to the home directory of the given user.

CREDITS

The work of the following groups and organizations was invaluable for this project:

The *FSF* (GNU, http://www.fsf.org/ project with their free and portable development software and lots of other useful tools and libraries.

The *Linux developers community* (http://www.linux.org/) for providing a free, open source operating system.

The *Open Group* (http://www.unix-systems.org/) for making their standard specifications available on the Internet for free.

VERSION

This man page describes version 1.7.4 of **socat**.

BUGS

Addresses cannot be nested, so a single socat process cannot, e.g., drive ssl over socks.

Address option ftruncate without value uses default 1 instead of 0.

Verbose modes (-x and/or -v) display line termination characters inconsistently when address options cr or crnl are used: They show the data *after* conversion in either direction.

The data transfer blocksize setting (-b) is ignored with address readline.

Send bug reports to <socat@dest-unreach.org>

SEE ALSO

nc(1), rinetd(8), openssl(1), stunnel(8), rlwrap(1), setsid(1)

Socat home page http://www.dest-unreach.org/socat/

AUTHOR

Gerhard Rieger <rieger@dest-unreach.org> and contributors