



**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

***Rocky Enterprise Linux 9.2 Manual Pages on command 'APR::Socket.3pm'***

***\$ man APR::Socket.3pm***

libapache2-mod-perl2-2.0.12::doUseraContrlibapache2-mod-perl2-2.0.12::docs::api::APR::Socket(3pm)

NAME

APR::Socket - Perl API for APR sockets

Synopsis

```
use APR::Socket ();

### set the socket to the blocking mode if it isn't already
### and read in the loop and echo it back

use APR::Const -compile => qw(SO_NONBLOCK);

if ($sock->opt_get(APR::Const::SO_NONBLOCK)) {
    $sock->opt_set(APR::Const::SO_NONBLOCK => 0);
}

# read from/write to the socket (w/o handling possible failures)

my $wanted = 1024;

while ($sock->recv(my $buff, $wanted)) {
    $sock->send($buff);
}

### get/set IO timeout and try to read some data

use APR::Const -compile => qw(TIMEUP);

# timeout is in usecs!

my $timeout = $sock->timeout_get();

if ($timeout < 10_000_000) {
    $sock->timeout_set(20_000_000); # 20 secs
}

}
```

```

# now read, while handling timeouts

my $wanted = 1024;

my $buff;

my $rlen = eval { $sock->recv($buff, $wanted) };

if ($@ && ref $@ && $@ == APR::Const::TIMEUP) {

    # timeout, do something, e.g.

    warn "timed out, will try again later";

}

else {

    warn "asked for $wanted bytes, read $rlen bytes\n";

    # do something with the data

}

# non-blocking io poll

$sock->opt_set(APR::Const::SO_NONBLOCK => 1);

my $rc = $sock->poll($c->pool, 1_000_000, APR::Const::POLLIN);

if ($rc == APR::Const::SUCCESS) {

    # read the data

}

else {

    # handle the condition

}

# fetch the operating level socket

my $fd=$sock->fileno;

```

## Description

"APR::Socket" provides the Perl interface to APR sockets.

## API

"APR::Socket" provides the following methods:

"fileno"

Get the operating system socket, the file descriptor on UNIX.

```
$fd = $sock->fileno;
```

```
obj: $sock ( "APR::Socket object" )
```

The socket

```
ret: $fd ( integer )
```

The OS-level file descriptor.

since: 2.0.5 (not implemented on Windows)

"opt\_get"

Query socket options for the specified socket

```
$val = $sock->opt_get($opt);
```

obj: \$sock ( "APR::Socket object" )

the socket object to query

arg1: \$opt ( "APR::Const constant" )

the socket option we would like to configure. Here are the available socket options.

ret: \$val ( integer )

the currently set value for the socket option you've queried for

excpt: "APR::Error"

since: 2.0.00

Examples can be found in the socket options constants section. For example setting the IO to the blocking mode.

"opt\_set"

Setup socket options for the specified socket

```
$sock->opt_set($opt, $val);
```

obj: \$sock ( "APR::Socket object" object )

the socket object to set up.

arg1: \$opt ( "APR::Const" constant )

the socket option we would like to configure. Here are the available socket options.

arg2: \$val ( integer )

value for the option. Refer to the socket options section to learn about the expected values.

ret: no return value

excpt: "APR::Error"

since: 2.0.00

Examples can be found in the socket options constants section. For example setting the IO to the blocking mode.

"poll"

Poll the socket for events:

```
$src = $sock->poll($pool, $timeout, $events);
```

obj: \$sock ( "APR::Socket object" )

The socket to poll

arg1: \$pool ( "APR::Pool object" )

usually "\$c->pool".

arg2: \$timeout ( integer )

The amount of time to wait (in milliseconds) for the specified events to occur.

arg3: \$events ( "APR::Const :poll constants" )

The events for which to wait.

For example use "APR::Const::POLLIN" to wait for incoming data to be available,

"APR::Const::POLLOUT" to wait until it's possible to write data to the socket and

"APR::Const::POLLPRI" to wait for priority data to become available.

ret: \$rc ( "APR::Const constant" )

If "APR::Const::SUCCESS" is received than the polling was successful. If not -- the error code is returned, which can be converted to the error string with help of

"APR::Error::strerror".

since: 2.0.00

For example poll a non-blocking socket up to 1 second when reading data from the client:

```
use APR::Socket ();
use APR::Connection ();
use APR::Error ();
use APR::Const -compile => qw(SO_NONBLOCK POLLIN SUCCESS TIMEUP);
$sock->opt_set(APR::Const::SO_NONBLOCK => 1);
my $rc = $sock->poll($c->pool, 1_000_000, APR::Const::POLLIN);
if ($rc == APR::Const::SUCCESS) {
    # Data is waiting on the socket to be read.
    # $sock->recv(my $buf, BUFF_LEN)
}
elsif ($rc == APR::Const::TIMEUP) {
    # One second elapsed and still there is no data waiting to be
    # read. for example could try again.
}
else {
    die "poll error: " . APR::Error::strerror($rc);
```

```
}
```

```
"recv"
```

Read incoming data from the socket

```
$len = $sock->recv($buffer, $wanted);
```

obj: \$sock ( "APR::SockAddr object" object )

The socket to read from

arg1: \$buffer ( SCALAR )

The buffer to fill. All previous data will be lost.

arg2: \$wanted ( int )

How many bytes to attempt to read.

ret: \$len ( number )

How many bytes were actually read.

\$buffer gets populated with the string that is read. It will contain an empty string

if there was nothing to read.

excpt: "APR::Error"

If you get the '(11) Resource temporarily unavailable' error (exception

"APR::Const::EAGAIN") (or another equivalent, which might be different on non-POSIX systems), then you didn't ensure that the socket is in a blocking IO mode before using

it. Note that you should use "APR::Status::is\_EAGAIN" to perform this check (since

different error codes may be returned for the same event on different OSes). For

example if the socket is set to the non-blocking mode and there is no data right away,

you may get this exception thrown. So here is how to check for it and retry a few

times after short delays:

```
use APR::Status ();
```

```
$sock->opt_set(APR::Const::SO_NONBLOCK, 1);
```

```
# ....
```

```
my $tries = 0;
```

```
my $buffer;
```

```
RETRY: my $rlen = eval { $socket->recv($buffer, SIZE) };
```

```
if ($@)
```

```
    die $@ unless ref $@ && APR::Status::is_EAGAIN($@);
```

```
    if ($tries++ < 3) {
```

```
        # sleep 250msec
```

```

        select undef, undef, undef, 0.25;
        goto RETRY;
    }
    else {
        # do something else
    }
}

warn "read $rlen bytes\n"

```

If timeout was set via "timeout\_set/C\_timeout\_set\_", you may need to catch the "APR::Const::TIMEUP" exception. For example:

```

use APR::Const -compile => qw(TIMEUP);
$sock->timeout_set(1_000_000); # 1 sec
my $buffer;
eval { $sock->recv($buffer, $wanted) };
if ($@ && $@ == APR::Const::TIMEUP) {
    # timeout, do something, e.g.
}

```

If not handled -- you may get the error '70007: The timeout specified has expired'.

Another error condition that may occur is the '(104) Connection reset by peer' error, which is up to your application logic to decide whether it's an error or not. This error usually happens when the client aborts the connection.

```

use APR::Const -compile => qw(ECONNABORTED);
my $buffer;
eval { $sock->recv($buffer, $wanted) };
if ($@ == APR::Const::ECONNABORTED) {
    # ignore it or deal with it
}

```

since: 2.0.00

Here is the quick prototype example, which doesn't handle any errors (mod\_perl will do that for you):

```

use APR::Socket ();
# set the socket to the blocking mode if it isn't already
use APR::Const -compile => qw(SO_NONBLOCK);

```

```

if ($sock->opt_get(APR::Const::SO_NONBLOCK)) {
    $sock->opt_set(APR::Const::SO_NONBLOCK => 0);
}
# read from/write to the socket (w/o handling possible failures)
my $wanted = 1024;
while ($sock->recv(my $buffer, $wanted)) {
    $sock->send($buffer);
}

```

If you want to handle errors by yourself, the loop may look like:

```

use APR::Const -compile => qw(ECONNABORTED);
# ...
while (1) {
    my $buf;
    my $len = eval { $sock->recv($buf, $wanted) };
    if ($@) {
        # handle the error, e.g. to ignore aborted connections but
        # rethrow any other errors:
        if ($@ == APR::Const::ECONNABORTED) {
            # ignore
            last;
        }
        else {
            die $@; # rethrow
        }
    }
    if ($len) {
        $sock->send($buffer);
    }
    else {
        last;
    }
}

```

Write data to the socket

```
$wlen = $sock->send($buf, $opt_len);
```

obj: \$sock ( "APR::Socket object" )

The socket to write to

arg1: \$buf ( scalar )

The data to send

opt arg2: \$opt\_len ( int )

There is no need to pass this argument, unless you want to send less data than contained in \$buf.

ret: \$wlen ( integer )

How many bytes were sent

since: 2.0.00

For examples see the "recv" item.

"timeout\_get"

Get socket timeout settings

```
$usecs = $sock->timeout_get();
```

obj: \$sock ( "APR::Socket object" )

The socket to set up.

ret: \$usecs ( number )

Currently set timeout in microseconds (and also the blocking IO behavior). See ("APR::timeout\_set") for possible values and their meaning.

excpt: "APR::Error"

since: 2.0.00

"timeout\_set"

Setup socket timeout.

```
$sock->timeout_set($usecs);
```

obj: \$sock ( "APR::Socket object" )

The socket to set up.

arg1: \$usecs ( number )

Value for the timeout in microseconds and also the blocking IO behavior.

The possible values are:

t > 0

"send()" and "recv()" throw ("APR::Const::TIMEUP" exception) if specified time

elapses with no data sent or received.

Notice that the positive value is in micro seconds. So if you want to set the timeout for 5 seconds, the value should be: 5\_000\_000.

This mode sets the socket into a non-blocking IO mode.

`t == 0`

"send()" and "recv()" calls never block.

`t < 0`

"send()" and "recv()" calls block.

Usually just -1 is used for this case, but any negative value will do.

This mode sets the socket into a blocking IO mode.

ret: no return value

excpt: "APR::Error"

since: 2.0.00

#### Unsupported API

"APR::Socket" also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod\_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

#### "bind"

META: Autogenerated - needs to be reviewed/completed

Bind the socket to its associated port

```
$ret = $sock->bind($sa);
```

obj: \$sock ( "APR::Socket object" )

The socket to bind

arg1: \$sa ( "APR::SockAddr object" )

The socket address to bind to

ret: \$ret ( integer )

since: subject to change

This may be where we will find out if there is any other process using the selected port.

#### "close"

META: Autogenerated - needs to be reviewed/completed

Close a socket.

```
$ret = $sock->close();
```

```
obj: $sock ( "APR::Socket object" )
```

The socket to close

```
ret: $ret ( integer )
```

since: subject to change

"connect"

META: Autogenerated - needs to be reviewed/completed

Issue a connection request to a socket either on the same machine or a different one.

```
$ret = $sock->connect($sa);
```

```
obj: $sock ( "APR::Socket object" )
```

The socket we wish to use for our side of the connection

```
arg1: $sa ( "APR::SockAddr object" )
```

The address of the machine we wish to connect to. If NULL, APR assumes that the sockaddr\_in in the apr\_socket is completely filled out.

```
ret: $ret ( integer )
```

since: subject to change

"listen"

META: Autogenerated - needs to be reviewed/completed

Listen to a bound socket for connections.

```
$ret = $sock->listen($backlog);
```

```
obj: $sock ( "APR::Socket object" )
```

The socket to listen on

```
arg1: $backlog ( integer )
```

The number of outstanding connections allowed in the sockets listen queue. If this value is less than zero, the listen queue size is set to zero.

```
ret: $ret ( integer )
```

since: subject to change

"recvfrom"

META: Autogenerated - needs to be reviewed/completed

```
$ret = $from->recvfrom($sock, $flags, $buf, $len);
```

```
obj: $from ( "APR::SockAddr object" )
```

The apr\_sockaddr\_t to fill in the recipient info

```
arg1: $sock ( "APR::SockAddr object" )
```

The socket to use

arg2: \$flags ( integer )

The flags to use

arg3: \$buf ( integer )

The buffer to use

arg4: \$len ( string )

The length of the available buffer

ret: \$ret ( integer )

since: subject to change

"sendto"

META: Autogenerated - needs to be reviewed/completed

```
$ret = $sock->sendto($where, $flags, $buf, $len);
```

obj: \$sock ( "APR::Socket object" )

The socket to send from

arg1: \$where ( "APR::Socket object" )

The apr\_sockaddr\_t describing where to send the data

arg2: \$flags ( integer )

The flags to use

arg3: \$buf ( scalar )

The data to send

arg4: \$len ( string )

The length of the data to send

ret: \$ret ( integer )

since: subject to change

## See Also

mod\_perl 2.0 documentation.

## Copyright

mod\_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.

## Authors

The mod\_perl development team and numerous contributors.

perl v5.34.0

libapache2-mod-perl2-2.0.12::docs::api::APR::Socket(3pm)