1 APR::Socket - Perl API for APR sockets
1.1 Synopsis

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 $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) {
    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;
1.2 Description

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

1.3 API

APR::Socket provides the following methods:

1.3.1 fileno

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

```perl
$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)

1.3.2 opt_get

Query socket options for the specified socket

```perl
$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
- **except**: 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.

1.3.3 opt_set

Setup socket options for the specified socket

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

- obj: $sock (APR::Socket 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.

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

1.3.4 poll

Poll the socket for events:

```php
$rc = $sock->poll($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 \texttt{APR::Const::POLLIN} to wait for incoming data to be available, \texttt{APR::Const::POLLOUT} to wait until it's possible to write data to the socket and \texttt{APR::Const::POLLPRI} to wait for priority data to become available.

- **ref:** $rc (\texttt{APR::Const} constant)
  
  If \texttt{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 \texttt{APR::Error::strerror}.

- **since:** 2.0.00

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

```perl
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);
}
```

### 1.3.5 recv

Read incoming data from the socket

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

- **obj:** $sock (\texttt{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 \(\text{(number)}\)

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.

- **except:** 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 $@; # retrow
        }
    }
    if ($len) {
        $sock->send($buffer);
    } else {
        last;
    }
}
1.3.6 send

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.

1.3.7 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.
- **except: APR::Error**
- **since: 2.0.00**
1.3.8 timeout_set

Setup socket timeout.

```perl
$sock->timeout_set($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
- **except:** **APR::Error**
- **since:** 2.0.00

### 1.4 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.
1.4.1 bind

META: Autogenerated - needs to be reviewed/completed

Bind the socket to its associated port

```php
$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.

1.4.2 close

META: Autogenerated - needs to be reviewed/completed

Close a socket.

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

- **obj** $sock (APR::Socket object)
  The socket to close
- **ret** $ret (integer)
  since: subject to change

1.4.3 connect

META: Autogenerated - needs to be reviewed/completed

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

```php
$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

1.4.4 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

1.4.5 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**

### 1.4.6 `sendto`

META: Autogenerated - needs to be reviewed/completed

```perl
$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**

### 1.5 See Also

mod_perl 2.0 documentation.

### 1.6 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.
1.7 Authors

The mod_perl development team and numerous contributors.
# Table of Contents:

1  APR::Socket - Perl API for APR sockets ........................................ 1
   1.1 Synopsis .................................................................................. 2
   1.2 Description ............................................................................... 3
   1.3 API ......................................................................................... 3
      1.3.1 fileno ............................................................................... 3
      1.3.2 opt_get ............................................................................. 3
      1.3.3 opt_set ............................................................................. 4
      1.3.4 poll ................................................................................... 4
      1.3.5 recv .................................................................................. 5
      1.3.6 send .................................................................................. 8
      1.3.7 timeout_get ...................................................................... 8
      1.3.8 timeout_set ...................................................................... 9
   1.4 Unsupported API ....................................................................... 9
      1.4.1 bind ............................................................................... 10
      1.4.2 close .............................................................................. 10
      1.4.3 connect ........................................................................... 10
      1.4.4 listen .............................................................................. 11
      1.4.5 recvfrom ......................................................................... 11
      1.4.6 sendto ............................................................................. 12
   1.5 See Also .................................................................................. 12
   1.6 Copyright ............................................................................... 12
   1.7 Authors ................................................................................... 13