NAME

"IO::Async::Socket" – event callbacks and send buffering for a socket filehandle

SYNOPSIS

```
use IO::Async::Socket;

use IO::Async::Loop;
my $loop = IO::Async::Loop->new;

my $socket = IO::Async::Socket->new(
   on_recv => sub {
      my ( $self, $dgram, $addr ) = @_;

      print "Received reply: $dgram\n",
      $loop->stop;
   },
   on_recv_error => sub {
      my ( $self, $errno ) = @_;
      die "Cannot recv - $errno\n";
   },
);
$loop->add( $socket );

$socket->connect(
   host     => "some.host.here",
   service  => "echo",
   socktype => 'dgram',
)->get;

$socket->send( "A TEST DATAGRAM" );

$loop->run;
```

DESCRIPTION

This subclass of IO::Async::Handle contains a socket filehandle. It provides a queue of outgoing data. It invokes the `on_recv` handler when new data is received from the filehandle. Data may be sent to the filehandle by calling the `send` method.

It is primarily intended for `SOCK_DGRAM` or `SOCK_RAW` sockets (such as UDP or packet-capture); for `SOCK_STREAM` sockets (such as TCP) an instance of IO::Async::Stream is more appropriate.

EVENTS

The following events are invoked, either using subclass methods or CODE references in parameters:

on_recv `$data`**,** `$addr`

Invoke on receipt of a packet, datagram, or stream segment.

The `on_recv` handler is invoked once for each packet, datagram, or stream segment that is received. It is passed the data itself, and the sender's address.

on_recv_error `$errno`

Optional. Invoked when the `recv` method on the receiving handle fails.

on_send_error `$errno`

Optional. Invoked when the `send` method on the sending handle fails.

The `on_recv_error` and `on_send_error` handlers are passed the value of `$!` at the time the error occurred. (The `$!` variable itself, by its nature, may have changed from the original error by the time this handler runs so it should always use the value passed in).

If an error occurs when the corresponding error callback is not supplied, and there is not a subclass method for it, then the `close` method is called instead.

on_outgoing_empty
Optional. Invoked when the sending data buffer becomes empty.

PARAMETERS

The following named parameters may be passed to `new` or `configure`:

read_handle => IO
The IO handle to receive from. Must implement `fileno` and `recv` methods.

write_handle => IO
The IO handle to send to. Must implement `fileno` and `send` methods.

handle => IO
Shortcut to specifying the same IO handle for both of the above.

on_recv => CODE
on_recv_error => CODE
on_outgoing_empty => CODE
on_send_error => CODE
autoflush => BOOL
Optional. If true, the `send` method will atempt to send data to the operating system immediately, without waiting for the loop to indicate the filehandle is write-ready.

recv_len => INT
Optional. Sets the buffer size for `recv` calls. Defaults to 64 KiB.

recv_all => BOOL
Optional. If true, repeatedly call `recv` when the receiving handle first becomes read-ready. By default this is turned off, meaning at most one fixed-size buffer is received. If there is still more data in the kernel's buffer, the handle will stil be readable, and will be received from again.

This behaviour allows multiple streams and sockets to be multiplexed simultaneously, meaning that a large bulk transfer on one cannot starve other filehandles of processing time. Turning this option on may improve bulk data transfer rate, at the risk of delaying or stalling processing on other filehandles.

send_all => INT
Optional. Analogous to the `recv_all` option, but for sending. When `autoflush` is enabled, this option only affects deferred sending if the initial attempt failed.

The condition requiring an `on_recv` handler is checked at the time the object is added to a Loop; it is allowed to create a `IO::Async::Socket` object with a read handle but without a `on_recv` handler, provided that one is later given using `configure` before the stream is added to its containing Loop, either directly or by being a child of another Notifier already in a Loop, or added to one.

METHODS

send

```
$socket->send( $data, $flags, $addr )
```

This method adds a segment of data to be sent, or sends it immediately, according to the `autoflush` parameter. `$flags` and `$addr` are optional.

If the `autoflush` option is set, this method will try immediately to send the data to the underlying filehandle, optionally using the given flags and destination address. If this completes successfully then it will have been sent by the time this method returns. If it fails to send, then the data is queued as if `autoflush` were not set, and will be flushed as normal.

EXAMPLES

Send-first on a UDP Socket
`UDP` is carried by the `SOCK_DGRAM` socket type, for which the string `'dgram'` is a convenient shortcut:

```
$socket->connect(
   host     => $hostname,
   service  => $service,
   socktype => 'dgram',
   ...
)
```

Receive-first on a UDP Socket

A typical server pattern with `UDP` involves binding a well-known port number instead of connecting to one, and waiting on incoming packets.

```
$socket->bind(
   service  => 12345,
   socktype => 'dgram',
)->get;
```

SEE ALSO

- IO::Handle – Supply object methods for I/O handles

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>