NAME

"IO::Async::Handle" – event callbacks for a non–blocking file descriptor

SYNOPSIS

This class is likely not to be used directly, because subclasses of it exist to handle more specific cases. Here is an example of how it would be used to watch a listening socket for new connections. In real code, it is likely that the `Loop->listen` method would be used instead.

```
 use IO::Socket::INET;
 use IO::Async::Handle;

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

 my $socket = IO::Socket::INET->new( LocalPort => 1234, Listen => 1 );

 my $handle = IO::Async::Handle->new(
    handle => $socket,

    on_read_ready  => sub {
       my $new_client = $socket->accept;
       ...
    },
 );

 $loop->add( $handle );
```

For most other uses with sockets, pipes or other filehandles that carry a byte stream, the IO::Async::Stream class is likely to be more suitable. For non-stream sockets, see IO::Async::Socket.

DESCRIPTION

This subclass of IO::Async::Notifier allows non-blocking IO on filehandles. It provides event handlers for when the filehandle is read– or write-ready.

EVENTS

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

on_read_ready

Invoked when the read handle becomes ready for reading.

on_write_ready

Invoked when the write handle becomes ready for writing.

on_closed

Optional. Invoked when the handle becomes closed.

This handler is invoked before the filehandles are closed and the Handle removed from its containing Loop. The `loop` will still return the containing Loop object.

PARAMETERS

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

read_handle => IO
write_handle => IO

The reading and writing IO handles. Each must implement the `fileno` method. Primarily used for passing `STDIN` / `STDOUT`; see the SYNOPSIS section of IO::Async::Stream for an example.

handle => IO

The IO handle for both reading and writing; instead of passing each separately as above. Must implement `fileno` method in way that `IO::Handle` does.

read_fileno => INT
write_fileno => INT

File descriptor numbers for reading and writing. If these are given as an alternative to `read_handle` or `write_handle` then a new `IO::Handle` instance will be constructed around each.

on_read_ready => CODE
on_write_ready => CODE
on_closed => CODE

CODE references for event handlers.

want_readready => BOOL
want_writeready => BOOL

If present, enable or disable read– or write-ready notification as per the `want_readready` and `want_writeready` methods.

It is required that a matching `on_read_ready` or `on_write_ready` are available for any handle that is provided; either passed as a callback CODE reference or as an overridden the method. I.e. if only a `read_handle` is given, then `on_write_ready` can be absent. If `handle` is used as a shortcut, then both read and write-ready callbacks or methods are required.

If no IO handles are provided at construction time, the object is still created but will not yet be fully-functional as a Handle. IO handles can be assigned later using the `set_handle` or `set_handles` methods, or by `configure`. This may be useful when constructing an object to represent a network connection, before the `connect(2)` has actually been performed yet.

METHODS

The following methods documented with a trailing call to `->get` return Future instances.

set_handle

```
$handle->set_handles( %params )
```

Sets new reading or writing filehandles. Equivalent to calling the `configure` method with the same parameters.

set_handle

```
$handle->set_handle( $fh )
```

Shortcut for

```
$handle->configure( handle => $fh )
```

close

```
$handle->close
```

This method calls `close` on the underlying IO handles. This method will then remove the handle from its containing loop.

close_read
close_write

```
$handle->close_read

$handle->close_write
```

Closes the underlying read or write handle, and deconfigures it from the object. Neither of these methods will invoke the `on_closed` event, nor remove the object from the Loop if there is still one open handle in the object. Only when both handles are closed, will `on_closed` be fired, and the object removed.

new_close_future

```
$handle->new_close_future->get
```

Returns a new IO::Async::Future object which will become done when the handle is closed. Cancelling the `$future` will remove this notification ability but will not otherwise affect the `$handle`.

read_handle
write_handle

```
$handle = $handle->read_handle

$handle = $handle->write_handle
```

These accessors return the underlying IO handles.

read_fileno
write_fileno

```
$fileno = $handle->read_fileno

$fileno = $handle->write_fileno
```

These accessors return the file descriptor numbers of the underlying IO handles.

want_readready
want_writeready

```
$value = $handle->want_readready

$oldvalue = $handle->want_readready( $newvalue )

$value = $handle->want_writeready

$oldvalue = $handle->want_writeready( $newvalue )
```

These are the accessor for the `want_readready` and `want_writeready` properties, which define whether the object is interested in knowing about read– or write-readiness on the underlying file handle.

socket

```
$handle->socket( $ai )
```

Convenient shortcut to creating a socket handle, as given by an addrinfo structure, and setting it as the read and write handle for the object.

`$ai` may be either a `HASH` or `ARRAY` reference of the same form as given to IO::Async::OS's `extract_addrinfo` method.

This method returns nothing if it succeeds, or throws an exception if it fails.

bind

```
$handle = $handle->bind( %args )->get
```

Performs a `getaddrinfo` resolver operation with the `passive` flag set, and then attempts to bind a socket handle of any of the return values.

bind (1 argument)

```
$handle = $handle->bind( $ai )->get
```

When invoked with a single argument, this method is a convenient shortcut to creating a socket handle and `bind()`ing it to the address as given by an addrinfo structure, and setting it as the read and write handle for the object.

`$ai` may be either a `HASH` or `ARRAY` reference of the same form as given to IO::Async::OS's `extract_addrinfo` method.

The returned future returns the handle object itself for convenience.

connect

```
$handle = $handle->connect( %args )->get
```

A convenient wrapper for calling the `connect` method on the underlying IO::Async::Loop object.

SEE ALSO

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

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>