NAME

"IO::Async::FileStream" – read the tail of a file

SYNOPSIS

```
use IO::Async::FileStream;

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

open my $logh, "<", "var/logs/daemon.log" or
   die "Cannot open logfile - $!";

my $filestream = IO::Async::FileStream->new(
   read_handle => $logh,

   on_initial => sub {
      my ( $self ) = @_;
      $self->seek_to_last( "\n" );
   },

   on_read => sub {
      my ( $self, $buffref ) = @_;

      while( $$buffref =~ s/^(.*\n)// ) {
         print "Received a line $1";
      }

      return 0;
   },
);

$loop->add( $filestream );

$loop->run;
```

DESCRIPTION

This subclass of IO::Async::Stream allows reading the end of a regular file which is being appended to by some other process. It invokes the `on_read` event when more data has been added to the file.

This class provides an API identical to IO::Async::Stream when given a `read_handle`; it should be treated similarly. In particular, it can be given an `on_read` handler, or subclassed to provide an `on_read` method, or even used as the `transport` for an IO::Async::Protocol::Stream object.

It will not support writing.

To watch a file, directory, or other filesystem entity for updates of other properties, such as `mtime`, see also IO::Async::File.

EVENTS

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

Because this is a subclass of IO::Async::Stream in read-only mode, all the events supported by `Stream` relating to the read handle are supported here. This is not a full list; see also the documentation relating to IO::Async::Stream.

`$ret` **= on_read \$buffer,** `$eof`

Invoked when more data is available in the internal receiving buffer.

Note that `$eof` only indicates that all the data currently available in the file has now been read; in contrast to a regular IO::Async::Stream, this object will not stop watching after this condition. Instead, it will

continue watching the file for updates.

on_truncated
Invoked when the file size shrinks. If this happens, it is presumed that the file content has been replaced. Reading will then commence from the start of the file.

on_initial `$size`
Invoked the first time the file is looked at. It is passed the initial size of the file. The code implementing this method can use the `seek` or `seek_to_last` methods to set the initial read position in the file to skip over some initial content.

This method may be useful to skip initial content in the file, if the object should only respond to new content added after it was created.

PARAMETERS

The following named parameters may be passed to `new` or `configure`, in addition to the parameters relating to reading supported by IO::Async::Stream.

filename => STRING
Optional. If supplied, watches the named file rather than the filehandle given in `read_handle`. The file will be opened by the constructor, and then watched for renames. If the file is renamed, the new filename is opened and tracked similarly after closing the previous file.

interval => NUM
Optional. The interval in seconds to poll the filehandle using `stat(2)` looking for size changes. A default of 2 seconds will be applied if not defined.

METHODS

seek

```
$filestream->seek( $offset, $whence )
```

Callable only during the `on_initial` event. Moves the read position in the filehandle to the given offset. `$whence` is interpreted as for `sysseek`, should be either `SEEK_SET`, `SEEK_CUR` or `SEEK_END`. Will be set to `SEEK_SET` if not provided.

Normally this would be used to seek to the end of the file, for example

```
on_initial => sub {
   my ( $self, $filesize ) = @_;
   $self->seek( $filesize );
}
```

seek_to_last

```
$success = $filestream->seek_to_last( $str_pattern, %opts )
```

Callable only during the `on_initial` event. Attempts to move the read position in the filehandle to just after the last occurrence of a given match. `$str_pattern` may be a literal string or regexp pattern.

Returns a true value if the seek was successful, or false if not. Takes the following named arguments:

blocksize => INT
Optional. Read the file in blocks of this size. Will take a default of 8KiB if not defined.

horizon => INT
Optional. Give up looking for a match after this number of bytes. Will take a default value of 4 times the blocksize if not defined.

To force it to always search through the entire file contents, set this explicitly to `0`.

Because regular file reading happens synchronously, this entire method operates entirely synchronously. If the file is very large, it may take a while to read back through the entire contents. While this is happening no other events can be invoked in the process.

When looking for a string or regexp match, this method appends the previously-read buffer to each block read from the file, in case a match becomes split across two reads. If `blocksize` is reduced to a very

small value, take care to ensure it isn't so small that a match may not be noticed.

This is most likely useful for seeking after the last complete line in a line-based log file, to commence reading from the end, while still managing to capture any partial content that isn't yet a complete line.

```
on_initial => sub {
   my $self = shift;
   $self->seek_to_last( "\n" );
}
```

TODO

- Move the actual file update watching code into IO::Async::Loop, possibly as a new watch/unwatch method pair `watch_file`.
- Consider if a construction-time parameter of `seek_to_end` or `seek_to_last` might be neater than a small code block in `on_initial`, if that turns out to be the only or most common form of use.

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>