NAME

       "IO::Async::Test" – utility functions for use in test scripts

SYNOPSIS

```
use Test::More tests => 1;
use IO::Async::Test;

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

my $result;

$loop->do_something(
   some => args,

   on_done => sub {
      $result = the_outcome;
   }
);

wait_for { defined $result };

is( $result, what_we_expected, 'The event happened' );


...

my $buffer = "";
my $handle = IO::Handle-> ...

wait_for_stream { length $buffer >= 10 } $handle => $buffer;

is( substr( $buffer, 0, 10, "" ), "0123456789", 'Buffer was correct' );

my $result = wait_for_future( $stream->read_until( "\n" ) )->get;
```

DESCRIPTION

       This module provides utility functions that may be useful when writing test scripts for code which uses IO::Async (as well as being used in the IO::Async test scripts themselves).

       Test scripts are often synchronous by nature; they are a linear sequence of actions to perform, interspersed with assertions which check for given conditions. This goes against the very nature of IO::Async which, being an asynchronisation framework, does not provide a linear stepped way of working.

       In order to write a test, the `wait_for` function provides a way of synchronising the code, so that a given condition is known to hold, which would typically signify that some event has occurred, the outcome of which can now be tested using the usual testing primitives.

       Because the primary purpose of IO::Async is to provide IO operations on filehandles, a great many tests will likely be based around connected pipes or socket handles. The `wait_for_stream` function provides a convenient way to wait for some content to be written through such a connected stream.

FUNCTIONS

   testing_loop

```
testing_loop( $loop )
```

       Set the IO::Async::Loop object which the `wait_for` function will loop on.

wait_for

```
wait_for { COND } OPTS
```

Repeatedly call the `loop_once` method on the underlying loop (given to the `testing_loop` function), until the given condition function callback returns true.

To guard against stalled scripts, if the loop indicates a timeout for (a default of) 10 consequentive seconds, then an error is thrown.

Takes the following named options:

timeout => NUM
　　The time in seconds to wait before giving up the test as being stalled. Defaults to 10 seconds.

wait_for_stream

```
wait_for_stream { COND } $handle, $buffer
```

As `wait_for`, but will also watch the given IO handle for readability, and whenever it is readable will read bytes in from it into the given buffer. The buffer is NOT initialised when the function is entered, in case data remains from a previous call.

`$buffer` can also be a CODE reference, in which case it will be invoked being passed data read from the handle, whenever it is readable.

wait_for_future

```
$future = wait_for_future $future
```

Since version 0.68.

A handy wrapper around using `wait_for` to wait for a Future to become ready. The future instance itself is returned, allowing neater code.

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>