NAME

AE – simpler/faster/newer/cooler AnyEvent API

SYNOPSIS

```
use AnyEvent; # not AE

# file handle or descriptor readable
my $w = AE::io $fh, 0, sub { ...  };

# one-shot or repeating timers
my $w = AE::timer $seconds,         0, sub { ... }; # once
my $w = AE::timer $seconds, $interval, sub { ... }; # repeated

print AE::now;  # prints current event loop time
print AE::time; # think Time::HiRes::time or simply CORE::time.

# POSIX signal
my $w = AE::signal TERM => sub { ... };

# child process exit
my $w = AE::child $pid, sub {
   my ($pid, $status) = @_;
   ...
};

# called when event loop idle (if applicable)
my $w = AE::idle sub { ... };

my $cv = AE::cv; # stores whether a condition was flagged
$cv->send; # wake up current and all future recv's
$cv->recv; # enters "main loop" till $condvar gets ->send
# use a condvar in callback mode:
$cv->cb (sub { $_[0]->recv });
```

DESCRIPTION

This module documents the new simpler AnyEvent API.

The rationale for the new API is that experience with EV shows that this API actually ‘‘works’’, despite its lack of extensibility, leading to a shorter, easier and faster API.

The main differences from AnyEvent is that function calls are used instead of method calls, and that no named arguments are used.

This makes calls to watcher creation functions really short, which can make a program more readable despite the lack of named parameters. Function calls also allow more static type checking than method calls, so many mistakes are caught at compile-time with this API.

Also, some backends (Perl and EV) are so fast that the method call overhead is very noticeable (with EV it increases the execution time five– to six-fold, with Perl the method call overhead is about a factor of two).

Note that the `AE` API is an alternative to, not the future version of, the AnyEvent API. Both APIs can be used interchangeably and there are no plans to ‘‘switch’’, so if in doubt, feel free to use the AnyEvent API in new code.

As the AE API is complementary, not everything in the AnyEvent API is available, and you still need to use AnyEvent for the finer stuff. Also, you should not `use AE` directly, `use AnyEvent` will provide the AE namespace.

At the moment, these functions will become slower then their method-call counterparts when using AnyEvent::Strict or AnyEvent::Debug::wrap.

FUNCTIONS

This section briefly describes the alternative watcher constructors and other functions available inside the `AE` namespace. Semantics are not described here; please refer to the description of the function or method with the same name in the AnyEvent manpage for the details.

`$w` = AE::io `$fh_or_fd`, `$watch_write`, `$cb`

Creates an I/O watcher that listens for read events (`$watch_write` false) or write events (`$watch_write` is true) on the file handle or file descriptor `$fh_or_fd`.

The callback `$cb` is invoked as soon and as long as I/O of the type specified by `$watch_write`) can be done on the file handle/descriptor.

Example: wait until STDIN becomes readable.

```
$stdin_ready = AE::io *STDIN, 0, sub { scalar <STDIN> };
```

Example: wait until STDOUT becomes writable and print something.

```
$stdout_ready = AE::io *STDOUT, 1, sub { print STDOUT "woaw\n" };
```

`$w` = AE::timer `$after`, `$interval`, `$cb`

Creates a timer watcher that invokes the callback `$cb` after at least `$after` second have passed (`$after` can be negative or `0`).

If `$interval` is `0`, then the callback will only be invoked once, otherwise it must be a positive number of seconds that specifies the interval between successive invocations of the callback.

Example: print ‘‘too late’’ after at least one second has passed.

```
$timer_once = AE::timer 1, 0, sub { print "too late\n" };
```

Example: print ‘‘blubb’’ once a second, starting as soon as possible.

```
$timer_repeated = AE::timer 0, 1, sub { print "blubb\n" };
```

`$w` = AE::signal `$signame`, `$cb`

Invoke the callback `$cb` each time one or more occurrences of the named signal `$signame` are detected.

`$w` = AE::child `$pid`, `$cb`

Invokes the callback `$cb` when the child with the given `$pid` exits (or all children, when `$pid` is zero).

The callback will get the actual pid and exit status as arguments.

`$w` = AE::idle `$cb`

Invoke the callback `$cb` each time the event loop is ‘‘idle’’ (has no events outstanding), but do not prevent the event loop from polling for more events.

`$cv` = AE::cv

`$cv` = AE::cv { BLOCK }

Create a new condition variable. The first form is identical to `AnyEvent->condvar`, the second form additionally sets the callback (as if the `cb` method is called on the condition variable).

AE::now

Returns the current event loop time (may be cached by the event loop).

AE::now_update

Ensures that the current event loop time is up to date.

AE::time

Return the current time (not cached, always consults a hardware clock).

AE::postpone { BLOCK }

Exactly the same as `AnyEvent::postpone`.

AE::log $level, $msg[, @args]
Exactly the same as AnyEvent::log (or AnyEvent::Log::log).

AUTHOR

Marc Lehmann <schmorp@schmorp.de>
http://anyevent.schmorp.de