

# PerlMongers - POE Tutorial

M. Cashner

April 20, 2001

## 1 Introduction

POE is an acronym of "Persistent Object Enviroment". It was originally designed as the core of a persistent object server. It would appear that Eris had other plans.

At its heart, POE is a framework for event driven state machines. This heart has two chambers: an event dispatcher (POE::Kernel) and state machines that are driven by dispatched events (POE::Session).

The rest of POE exists to help the state machines perform high-level functions (I/O, Sockets, etc).

POE simulates threading using cooperative timeslicing. Your POE program can do multiple things at the same time, *without* threading. There's a lot of magic in the POE core to do this and its, um, wierd.

To try and explain this in smaller words, POE programs sit and wait for things to happen. Maybe a client connects to your server. POE tells your Session that something is trying to connect. The Session acts appropriately. The rest of the time, your program sits and waits.

Maybe this will make more sense if we see some POE in action.

## 2 Writing POE Programs

*See Section 2.1 for the full code of this sample program.*

The simplest of all programs start with the following:

```
#!/usr/bin/perl
use strict;
use warnings;
use POE;
```

The `use POE;` line automatically load POE::Kernel and POE::Session, since nothing will work without those.

Now we need to define some sessions:

```
for(0..8) {  
    POE::Session->create(  
        inline_states => {  
            _start => \&state_start,  
            _stop => \&state_stop,  
        },  
    );  
}
```

This for loop will create 9 sessions. The *\_start* state gets called when the session is starting up. This is where other components get set up. If you want to listen on a socket, you create that listener here.

When there is nothing left for a session to do (ie it hasn't been told to listen for other states or events), the session shuts down which results in a call to *\_stop*.

Other states can be defined if you want to make them available to the session.

Next, we define the states themselves:

```
my $count = 0;  
  
sub state_start {  
    $count++;  
    print "State Start $count\n";  
}  
  
sub state_stop {  
    print "State Stop $count\n";  
    $count--;  
}
```

These are pretty simple and self explanatory. As each session starts, we increment a counter and print something. As each session stops, we decrement the counter and print something.

Now its time to start the whole thing going:

```
$poe_kernel->run();
```

`$poe_kernel` is exported by POE into your namespace. `run()` starts the kernel and the magic begins. If you're running this program on a command line, you'll now see:

```
katerina:~/src/lpm% ./test.pl
State Start 1
State Start 2
State Start 3
State Start 4
State Start 5
State Start 6
State Start 7
State Start 8
State Start 9
State Stop 9
State Stop 8
State Stop 7
State Stop 6
State Stop 5
State Stop 4
State Stop 3
State Stop 2
State Stop 1
```

Now wasn't that fun? The full code follows:

## 2.1 Full Sample Code

```
#!/usr/bin/perl

use strict;
use warnings;
use POE;

my $count;
for(0..8) {
    POE::Session->create(
        inline_states => {
            _start => \&state_start,
            _stop => \&state_stop,
        },,
    );
}

$poe_kernel->run();

exit;

#####
```

```

sub state_start {
    $count++;
    print "State Start $count\n";
}

sub state_stop {
    print "State Stop $count\n";
    $count--;
}

```

### 3 Intermediate Example

```

#!/usr/bin/perl

use strict;
use warnings;

use Socket;
use POE qw(Wheel::SocketFactory
    Wheel::ReadWrite
    Driver::SysRW
    Filter::Stream
);

#####
# MAIN
#####

local $| = 1;
our $debug      = 1;      # be very very noisy
our $serverport = 11211;  # 'poe' in base10 :P

fork and exit unless $debug;

POE::Session->create(
    inline_states => {
        _start => \&parent_start,
        _stop  => \&parent_stop,

        socket_birth => \&socket_birth,
        socket_death => \&socket_death,
    }
);

# $poe_kernel is exported from POE
$poe_kernel->run();

```

```

exit;

#####

sub parent_start {
    my $heap = $_[HEAP];

    print "= L = Listener birth\n" if $debug;

    $heap->{listener} = POE::Wheel::SocketFactory->new(
        BindAddress => '127.0.0.1',
        BindPort    => $serverport,
        Reuse       => 'yes',
        SuccessState => 'socket_birth',
        FailureState => 'socket_death',
    );
}

sub parent_stop {
    my $heap = $_[HEAP];
    delete $heap->{listener};
    delete $heap->{session};
    print "= L = Listener death\n" if $debug;
}

#####
# SOCKET #
#####

sub socket_birth {
    my ( $socket, $address, $port ) = @_ [ ARG0, ARG1, ARG2 ];
    $address = inet_ntoa($address);

    print "= S = Socket birth\n" if $debug;

    POE::Session->create(
        inline_states => {
            _start      => \&socket_success,
            _stop       => \&socket_death,

            socket_input => \&socket_input,
            socket_death => \&socket_death,
        },
        args => [ $socket, $address, $port ],
    );
}

```

```

}

sub socket_death {
    my $heap = $_[HEAP];
    if($heap->{socket_wheel}) {
        print "= S = Socket death\n" if $debug;
        delete $heap->{socket_wheel};
    }
}

sub socket_success {
    my ( $heap, $kernel, $connected_socket, $address, $port ) =
        @_[ HEAP, KERNEL, ARG0, ARG1, ARG2 ];

    print "= I = CONNECTION from $address : $port \n" if $debug;

    $heap->{socket_wheel} = POE::Wheel::ReadWrite->new(
        Handle      => $connected_socket,
        Driver      => POE::Driver::SysRW->new(),
        Filter      => POE::Filter::Stream->new(),

        InputState => 'socket_input',
        ErrorState => 'socket_death',
    );

    $heap->{socket_wheel}->put(
        "1 Welcome. Say something. I'll say it back.\n\n");
}

sub socket_input {
    my ( $heap, $buf ) = @_[ HEAP, ARG0 ];
    $buf =~ s/[\r\n]//gs;
    $heap->{socket_wheel}->put("$buf\n");
}

```