All about logging



Why log?

The poor man's debugger

THE BEST DEBUGGER EVAR

print "WTF is going on?\n";

Who loves logs?

Developers

What is going on?

Operations

What did they do this time?



What does ops want?

Multiple outputs

Syslog

STDERR

File

Runtime controls

Simple, consistent configuration

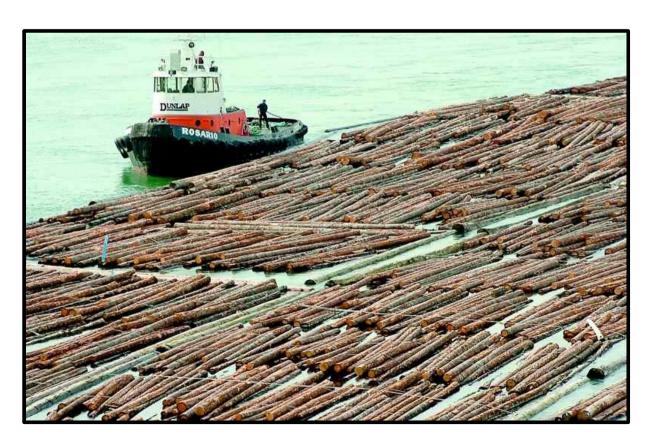
Rainbows

Existing Modules

Millions of them on CPAN

Two popular options:

Log4Perl Log::Dispatch



What do devs want?

Easy to write

Easy to add to existing code

Easy to configure when debugging problem code.

Data::Dumper

Stack traces

A pony

Why make a new logging module?

Everything else sucks

Lager sucks, too but it sucks less

and

It sucks in new and exciting ways...



What's "Lager"?

It started out as lager

Then it was called Next::OpenSIPS::Log

Now it's Log::Lager.

Thank you Marchex.

Parsing

Sucks: Random formats require specialized parsers.

Sucks less: Log entries are formatted as JSON

Single line JSON



Configuration Granularity

Sucks: Application wide settings

Sucks less: Focus on trouble spots

Lexical (per block) configuration with a "kill switch" for ops.

Configure by package name

Configure by subroutine name

Performance

Sucks: Computing things only to throw them away

Sucks less: Lazy evaluation via callbacks

Callbacks are executed only when a message is emitted.

```
TRACE sub { expensive_calculation($foo) };
```

Ease of use

Sucks: Create, configure and pass logging objects

Sucks less: Namespace pollution

Imports logging routines

FATAL ERROR

WARN INFO

DEBUG TRACE

GUTS

Log levels

Sucks: Sorted log levels

TRACE > INFO

Means you can't see TRACE without INFO

Cluttered logs

Sucks less: Orthogonal log levels

Each level is configured independent of the others

Pristine logs

Complex Data Structures

Sucks: require manual use of Data::Dumper

Sucks less: Built in data dumping

JSON

Abridged data structures that won't choke JSON Optional pretty printing

Logging With Lager



Log Function Behaviors

Pass in a list of stuff:

Serialize as elements in a JSON list.

Recursive dump of data structures through Data::Abridge.

Pass in a single code reference:

Lazy evaluation of code – runs only when needed.

Can emulate any other input type.

Pass in a single Log::Lager::Message subtype:

Use your own object to control any initialization, formatting as desired.

Example

```
sub get bug list {
 my $sp = shift;
 my $url = join '', $BASE, map /##SP##/ ? $sp : $ , @BUGLIST URL;
 my $html = get url($url);
 my $t = HTML::TreeBuilder->new from content($html);
 return if $t->look down( class => qr/\bzero results\b/);
 my $buglist = $t->look down( class => gr/\bbz buglist\b/);
 my @buglist = $buglist->look down(class => gr/\bbz bugitem\b/);
 my @bugs;
  for my $bug item ( @buglist ) {
   my %buq;
    for my $col ('bz id column', 'bz assigned to column',
'bz short desc column']) {
     mv $kev = $col;
      key = ~ s/bz (w*) column/$1/;
     my $e = $bug item->look down( class => qr/\b$col\b/);
     bug\{key\} = e->as text;
   push @bugs, \%bug;
 return @bugs;
```

Logging added

```
use Log::Lager;
sub get bug list {
 my $sp = shift;
 my \url = join '', \BASE, map \/#\SP##/ ? \$sp : \$ , \ @BUGLIST URL;
  TRACE 'Getting bug list', URL => $url, SP => $sp;
 my $html = get url($url);
 DEBUG $html;
  my $t = HTML::TreeBuilder->new from content($html);
  if( $t->look down( class => gr/\bzero results\b/ ) ) {
    WARN 'Zero bugs found';
    return:
 my $buglist = $t->look down( class => gr/\bbz buglist\b/);
  my @buglist = $buglist->look down(class => qr/\bbz bugitem\b/);
  my @buqs;
  for my $bug item (@buglist) {
    TRACE 'Extracting bug from result set';
    GUTS $bug item;
    my %buq;
    for my $col ('bz id column', 'bz assigned to column',
'bz short desc column']) {
```

Add lexical controls

```
sub get bug list {
 use Log::Lager 'enable DTI stack T pretty D';
 my $sp = shift;
 my \url = join '', \BASE, map \/##SP##/ ? \$sp : \$ , \ @BUGLIST URL;
  TRACE 'Getting bug list', URL => $url, SP => $sp;
 my $html = get url($url);
 DEBUG $html;
 my $t = HTML::TreeBuilder->new from content($html);
 my $t = HTML::TreeBuilder->new from content($html);
  if( t->look down( class => qr/\bzero results\b/ ) ) {
   WARN 'Zero bugs found';
   return;
 my $buglist = $t->look down( class => gr/\bbz buglist\b/);
 my @buglist = $buglist->look down(class => gr/\bbz bugitem\b/);
 my @buas;
  for my $bug item (@buglist) {
   my %bug;
    TRACE 'Extracting bug from result set';
    GUTS $bug item;
    for my $col ('bz id column', 'bz assigned to column',
'bz short desc column' ]) {
     no Log::Lager 'D';
```

Sample Code...

```
sub supersub {
   my $foo = {
        name => 'Bob Smith',
        birthdate => '1965/12/21',
        address => bless({
             number => '1009',
             street => 'Maple St.',
             city => 'Springfield',
             state => 'Oregon'
        },'My::Address'),
    };
    DEBUG 'Adding address info to object: ', $foo;
```

Sample Output - Compact

Single line JSON – A bit hard to read, but very easy to parse.

```
[["2011-07-19 23:10:40 Z", "DEBUG", "tiny", 8929, 0, "test.pl",
"test.pl", 19, "main", "main::supersub"], "Adding address info to
object: ", {"address": {"My::Address": {"city": "Springfield",
"number": "1009", "state": "Oregon", "street": "Maple St."}},
"birthdate": "1965/12/21", "name": "Bob Smith"}]
```

Sample Output - Pretty

Multi-line JSON – Easy to read, but tougher to parse (yet still easy).

```
"Adding address info to object: ",
   "2011-07-19 22:58:23 Z",
                                     "address": {
   "DEBUG",
                                        "My::Address": {
   "tiny",
                                           "city": "Springfield",
   8915,
                                           "number": "1009",
   0,
                                           "state": "Oregon",
   "test.pl",
                                           "street": "Maple St."
  "test.pl",
   19,
                                     },
   "main",
                                     "birthdate": "1965/12/21",
   "main::supersub"
                                     "name": "Bob Smith"
],
```

Production

buggetter-log.conf:

```
enable FEW disable IDTG
nostack FEWIDTG
compact FEWIDTG
nofatal FEWIDTG
END
```

Disable lexical controls

buggetter-log.conf:

```
enable FEW disable IDTG
nostack FEWIDTG
compact FEWIDTG
nofatal FEWIDTG
lexoff
END
```

Control output

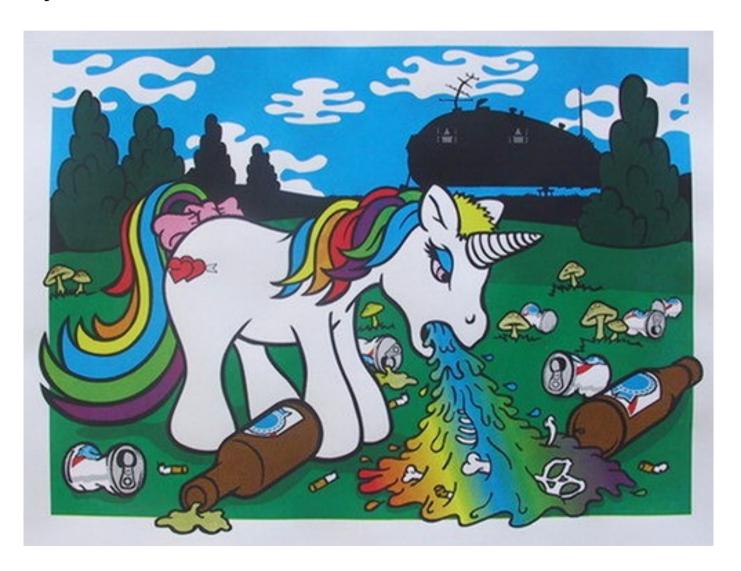
```
buggetter-log.conf:
   enable FEW disable IDTG
   nostack FEWIDTG
   compact FEWIDTG
   nofatal FEWIDTG
   lexoff
   syslog buggetter LOG LOCALO
   END
```

Target a package or sub

```
buggetter-log.conf:
  enable FEW disable IDTG
  nostack FEWIDTG
  compact FEWIDTG
  lexoff
  syslog buggetter LOG LOCALO
  package AT:: Bugfest enable DIT
  sub AT::Bugfest::broke enable G stack T
  END
```

Because I promised it

Rainbows and a pony!



Questions