

mod_perl 2.0, the Next Generation

by Stas Bekman
<http://stason.org/>
<stas@stason.org>
TicketMaster

O'Reilly Open Source Convention
Thursday, July 25 2002
San Diego, CA

This talk is available from:
<http://stason.org/talks/>

Last modified Fri Jul 19 20:56:37 2002 GMT

1 The Next Generation: mod_perl 2.0

1.1 About

- Why rewrite?
- What's new in Apache 2.0
- What's new in Perl 5.6.0 - 5.8.0
- What's new in mod_perl 2.0
- Installing mod_perl 2.0
- Configuring mod_perl 2.0
- Migrating from 1.0 to 2.0

- New Phases
- Protocol Handlers
- Filter Handlers

1.2 Versioning Convention

- To make things simple here and in the new docs:
- `mod_perl`:
 - `mod_perl 1.0` (not `mod_perl 1.xx`)
 - `mod_perl 2.0` (not `mod_perl 2.0.xx`)
- Apache:
 - Apache 1.3
 - Apache 2.0

1.3 Why the 2.0 Rewrite?

- Too patchy (6 years!), backward compatibility with:
 - Apache 1.3.0 - 1.3.27
 - Perl 5.003 - 5.8.0
- mod_perl 2.0 starts afresh with:
 - Apache 2.0 (incompatible with Apache 1.3)
 - Perl 5.6.0 (has semi-thread-safe Perl Interpreters)
 - Threaded mpms: 5.8.0 (really thread-safe)

- A new build system autogenerates the code used to
 - autogenerates the code that is used to ...
 - ...
 - which generates the final code ...
 - ... and it all works
- Automatically supports new Apache APIs

1.3.1 *The Apache::Test Framework*

- The core:

`All tests successful.`

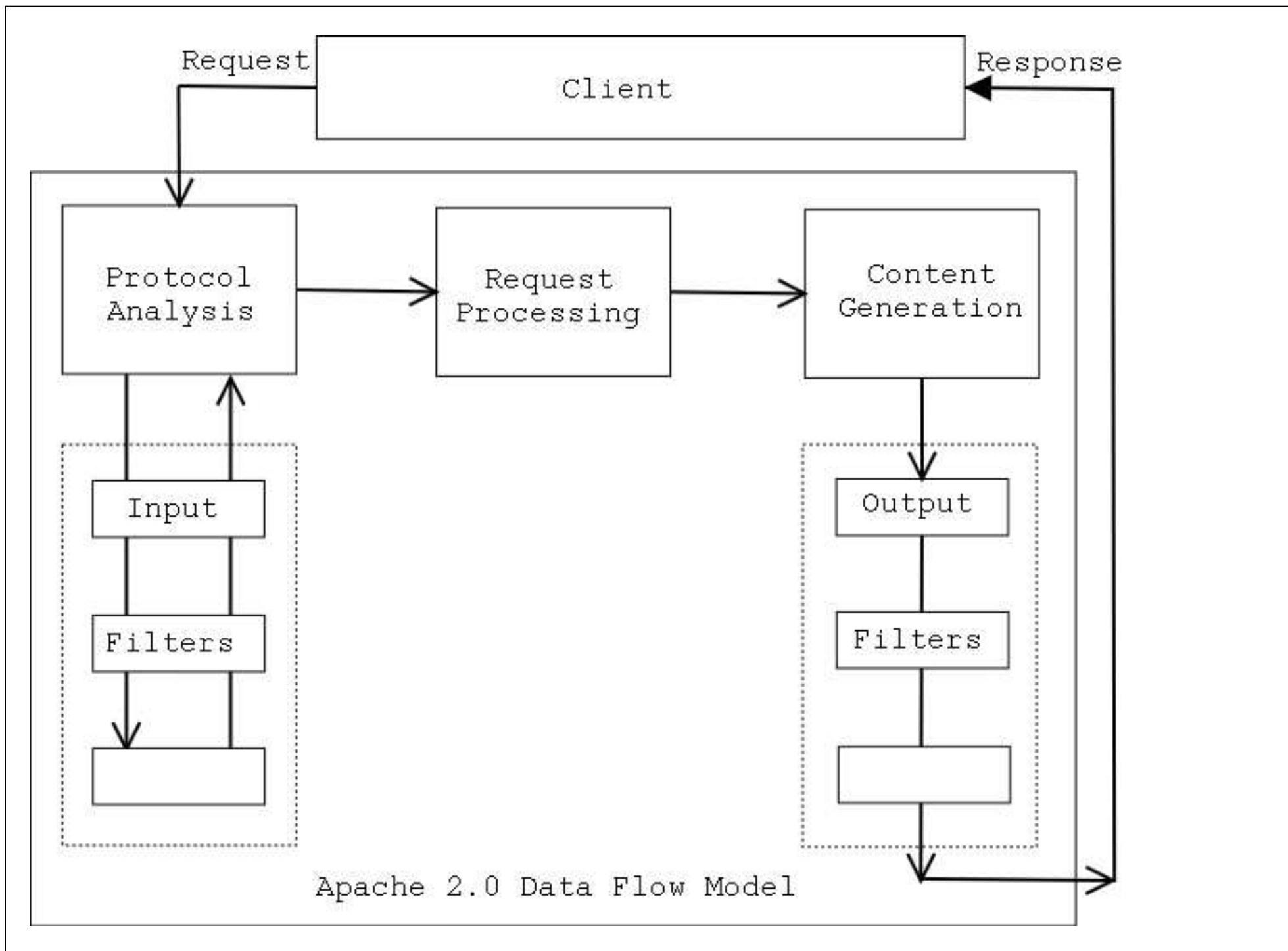
`Files=75, Tests=504, 65 wallclock secs ...`

- Any Perl module needing mod_perl 1.0 or 2.0
- Any Apache module (both 1.3 and 2.0), PHP, Python, C...
- Already used by httpd-test to test Apache!!!

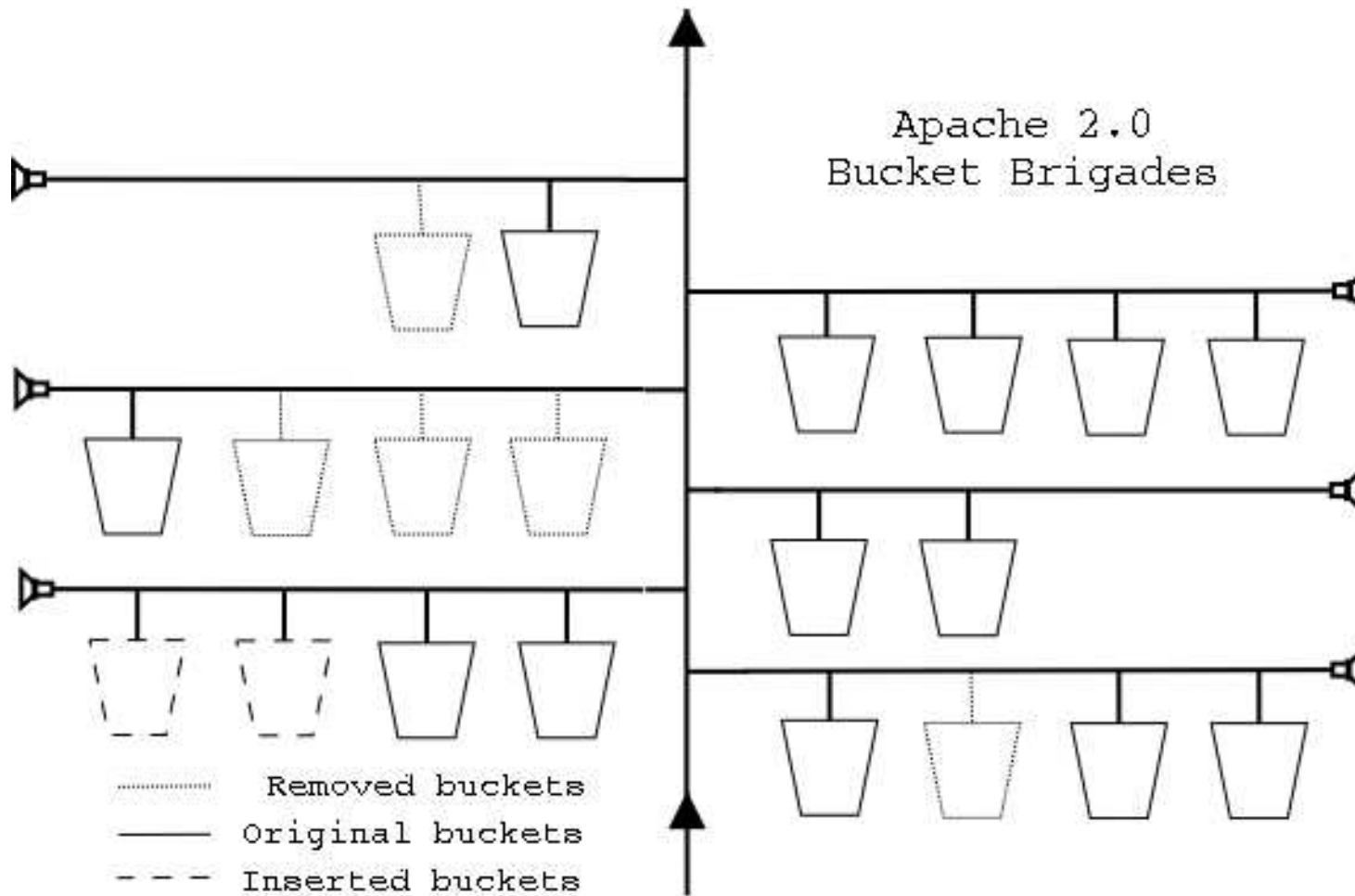
1.4 New in Apache 2.0

- Apache Portable Runtime
- Multi Processing Model modules (MPMs).
 - processes: prefork
 - threads: worker, leader, perchild...
 - os: mpmt_os2, netware, winnt, beos...
- Protocol Modules (HTTP, POP3, SMTP...)

- I/O Filtering
- Bucket Brigades
- Parsed Configuration Tree
- New Hook Scheme (Flexible, Order-able)
- Optional Functions



Apache 2.0 Bucket Brigades



1.5 New in Perl m/5\.(6|8)\.\d/

- Thread-safe Interpreter (5.8.0) via `perl_clone()`
- Subroutine attributes:

```
sub handler : FilterRequestHandler { ... }
```

- `CORE::GLOBAL::` subs overriding `CORE::`
- `PerlIO` layers => `APR::PerlIO`:

```
open my $fh, "<:APR", $file, $r;
```

- I18n: Unicode, UTF...

1.6 New in mod_perl 2.0

- All the new Apache 2.0 and Perl 5.6.0+ features
- Plus its own new features

1.6.1 Threads Support

- Thread Interpreters Pool
- `scalar @perl_interpreters != scalar @apache_threads`
 - no need for front-end/back-end separation

- Two classes of interpreters: *parent* and *clone*
- parent: preload modules and *perl_clone()* clones
- clones: do the real work
 - mutable data is copied by the clone
 - read-only data such as the syntax tree is shared
 - clone pools are FIFO => memory re-use

1.6.2 Thread-safety

- Manipulating Perl data is thread-safe (5.8.0)

`push()`, `map()`, `chomp()`, ...

- The rest, depends on the underlying implementation

`localtime()`, `readdir()`, `srand()`, ...

- Thread-safe but Process-scoped

`chdir()`, `umask()`, `chroot()`, ...

- See `perlthrtut(3)`

1.7 Installing Apache

```
% cd httpd-2.0.xx  
% ./configure --prefix=/home/httpd/httpd-2.0 --with-mpm=prefork  
% make && make install
```

1.8 Installing Perl

- Perl:

```
% cd perl-5.8.0  
% ./Configure -des -Dprefix=$HOME/perl/perl-5.8.0 -Dusethreads  
% make && make test && make install
```

- Prerequisites

```
% $HOME/perl/perl-5.8.0/bin/perl -MCPAN -e 'install("LWP")'
```

1.9 Installing mod_perl 2.0

```
% cd mod_perl-2.0.x  
% perl Makefile.PL MP_AP_PREFIX=/home/stas/src/httpd-2.0.xx  
% make && make test && make install
```

- for win32 binaries see <http://perl.apache.org/download/>

1.10 Configuring mod_perl 2.0

- DSO:

```
LoadModule perl_module modules/mod_perl.so
```

- Static: nada

1.10.1 Accessing the Modules

- mod_perl 2.0 Perl libs go to *Apache2/*
- Adjust @INC:

```
use Apache2 ( );
```

```
# before:
```

```
/usr/lib/perl5/site_perl/5.8.0/i686-linux-thread-multi
```

```
# after:
```

```
/usr/lib/perl5/site_perl/5.8.0/i686-linux-thread-multi/Apache2
```


1.10.2 *PerlRequire'd Startup File*

```
use Apache2 ();  
# use Apache::compat (); # 1.0 compat  
  
use lib qw(/home/httpd/perl);  
  
use ModPerl::Util (); #for CORE::GLOBAL::exit  
  
use Apache::RequestRec ();  
use Apache::RequestIO ();  
use Apache::RequestUtil ();  
  
use Apache::Server ();  
use Apache::ServerUtil ();  
use Apache::Connection ();  
use Apache::Log ();
```

```
use APR::Table ();

use ModPerl::Registry ();

use Apache::Const -compile => ':common';
use APR::Const -compile => ':common';

1;
```

1.10.3 Perl's Command Line Switches

- PerlSwitches passes any Perl switches
- e.g., enable warnings and taint checking:

PerlSwitches -wT

- adjust @INC values:

PerlSwitches -I/home/stas/modperl

1.10.4 mod_perl 2.0 Core Handlers

SetHandler perl-script

- As in mod_perl 1.0
- Unless set to -GlobalRequest assumes

PerlOptions +GlobalRequest

- Unless set to -SetupEnv assumes

PerlOptions +SetupEnv

- Tied STDOUT and STDERR

```
my $line = <STDIN>;  
print "Dahuuuuut!";
```

- on each request restores %ENV, @INC, \$/, STDOUT's \$| and END blocks

SetHandler modperl

- calls the Perl*Handler's callback func.
- sets MOD_PERL, GATEWAY_INTERFACE, PATH and TZ env vars.
- No tied IO handles:

```
$r->read($line, $len, $offset);  
$r->print("Dahuuuuut!");
```

1.10.4.1 A Simple Response Handler Example

- Printout environment variables ala `perl-script` core handler

```
PerlModule Apache::PrintEnv1  
<Location /print_env1>  
    SetHandler perl-script  
    PerlResponseHandler Apache::PrintEnv1  
</Location>
```

```
package Apache::PrintEnv1;
use strict;
use Apache::RequestRec (); # for $r->content_type
use Apache::Const -compile => 'OK';

sub handler {
    my $r = shift;

    $r->content_type('text/plain');
    for (sort keys %ENV){
        print "$_ => $ENV{$_}\n";
    }

    return Apache::OK;
}
1;
```


- Printout environment variables ala `modperl` core handler

```
PerlModule Apache::PrintEnv2
```

```
<Location /print_env2>
```

```
    SetHandler modperl
```

```
    PerlResponseHandler Apache::PrintEnv2
```

```
</Location>
```

```
package Apache::PrintEnv2;
use strict;
use Apache::RequestRec (); # for $r->content_type
use Apache::RequestIO ();  # for $r->print
use Apache::Const -compile => 'OK';

sub handler {
    my $r = shift;

    $r->content_type('text/plain');
    $r->subprocess_env;
    for (sort keys %ENV){
        $r->print("$_ => $ENV{$_}\n");
    }
    return Apache::OK;
}
1;
```

- Instead of calling:

```
$r->subprocess_env;
```

- could configure the location with:

```
PerlOptions +SetupEnv
```

1.10.5 *PerlOptions Directive*

- Disable mod_perl for a given VirtualHost:

```
<VirtualHost ...>  
    PerlOptions -Enable  
</VirtualHost>
```

- Give the VirtualHost its own interpreter pool.

```
<VirtualHost ...>  
    PerlOptions +Clone  
    PerlInterpStart 2  
    PerlInterpMax 2  
</VirtualHost>
```

- Run different versions of the same module:

```
<VirtualHost ...>  
    ServerName dev1  
    PerlOptions +Parent  
    PerlSwitches -Mblib=/home/dev1/lib/perl  
</VirtualHost>
```

```
<VirtualHost ...>  
    ServerName dev2  
    PerlOptions +Parent  
    PerlSwitches -Mblib=/home/dev2/lib/perl  
</VirtualHost>
```

- disallow certain handlers/options

```
<VirtualHost ...>  
    PerlOptions -Authen -Authz -Access -Sections  
</VirtualHost>
```

- Or maybe everything but the response handler:

```
<VirtualHost ...>  
    PerlOptions None +Response  
</VirtualHost>
```

- Resolve Perl*Handlers at startup time:

`PerlOptions +Autoload`

`PerlResponseHandler Apache::Magick`

- Disable the global `request_rec` (`Apache->request`)

`<Location ...>`

`SetHandler perl-script`

`PerlOptions -GlobalRequest`

`...`

`</Location>`

- s/PerlSendHeader On/PerlOptions +ParseHeaders/
- s/PerlSetupEnv Off/PerlOptions -SetupEnv/
- Merge Handlers

PerlFixupHandler Apache::FixupA

<Location /inside>

PerlOptions +MergeHandlers

PerlFixupHandler Apache::FixupB

</Location>

1.10.6 Threads Mode Specific Directives

- PerlInterpStart
- PerlInterpMax
- PerlInterpMinSpare
- PerlInterpMaxSpare
- PerlInterpMaxRequests

- PerlInterpScope
- Use for the lifetime of the request (default):

PerlInterpScope request

- Use a separate interpreter in subrequests:

PerlInterpScope subrequest

- Use a separate interpreter for each handler:

PerlInterpScope handler

1.10.7 Retrieving Server Startup Options

```
% httpd -DONE_PROCESS
```

- Retrieve:

```
if (Apache::exists_config_define("ONE_PROCESS")) {  
    print "Running in a single mode";  
}
```

1.11 New Apache Phases

- `PerlOpenLogsHandler`
- `PerlPostConfigHandler`
- `PerlPreConnectionHandler`
- `PerlProcessConnectionHandler`
- `PerlResponseHandler`
- `PerlInputFilterHandler`
- `PerlOutputFilterHandler`

1.11.1 The Shortest Migration Path from 1.0

```
use Apache2;  
use Apache::compat;
```

- Certain Configuration directives and APIs have changed
- See <http://perl.apache.org/docs/2.0/user/compat/compat.html>

1.11.2 *ModPerl::Registry* Family

- `s/Apache::Registry/ModPerl::Registry/`

```
Alias /perl/ /home/httpd/perl/  
<Location /perl>  
    SetHandler perl-script  
    PerlResponseHandler ModPerl::Registry  
    Options +ExecCGI  
    PerlOptions +ParseHeaders  
</Location>
```

- Cook your own registry with `Apache::RegistryCooker`

1.11.3 *Method Handlers*

- \$\$ doesn't work since some callbacks accept more than 2 args

```
package Bird;
@ISA = qw(Eagle);

sub handler : method {
    my($class, $r) = @_;
    ...;
}
```

- See the *attributes* manpage.

1.11.4 Apache::Eliza Protocol Module

- An example of a protocol working directly with a socket
- Very simple implementation, but cannot use filters

- Configuration:

```
Listen 8084
```

```
<VirtualHost _default_:8084>
```

```
    PerlModule Apache::Eliza
```

```
    PerlProcessConnectionHandler Apache::Eliza
```

```
</VirtualHost>
```

- And we give it a whirl:

```
% telnet localhost 8084
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
Hello Eliza
How do you do. Please state your problem.

How are you?
Oh, I?

Why do I have core dumped?
You say Why do you have core dumped?

I feel like writing some tests today, you?
```

I'm not sure I understand you fully.

Good bye, Eliza

Does talking about this bother you?

Connection closed by foreign host.

```
package Apache::Eliza;

use strict;
use warnings FATAL => 'all';

use Apache::Connection ();
use APR::Socket ();

require Chatbot::Eliza;

use Apache::Const -compile => 'OK';

use constant BUFF_LEN => 1024;

my $eliza = new Chatbot::Eliza;
```

```

sub handler {
    my Apache::Connection $c = shift;
    my APR::Socket $socket = $c->client_socket;

    my $buff;
    my $last = 0;
    while (1) {
        my($rlen, $wlen);
        $rlen = BUFF_LEN;
        $socket->recv($buff, $rlen);
        last if $rlen <= 0;

        # \r is sent instead of \n if the client is talking over telnet
        $buff =~ s/[\r\n]*$//;
        $last++ if $buff =~ /good bye/i;
        $buff = $eliza->transform( $buff ) . "\n\n";
        $socket->send($buff, length $buff);
        last if $last;
    }

    Apache::OK;
}
1;

```

1.11.5 Apache::Eliza2 Protocol Module

- An example of a protocol working directly with bucket brigades
- Complicated implementation, but can use filters

```
package Apache::Eliza2;

use strict;
use warnings FATAL => 'all';

use Apache::Connection ();
use APR::Bucket ();
use APR::Brigade ();
use APR::Util ();

require Chatbot::Eliza;

use APR::Const -compile => qw(SUCCESS EOF);
use Apache::Const -compile => qw(OK MODE_GETLINE);

my $eliza = new Chatbot::Eliza;
```

```
sub handler {
    my Apache::Connection $c = shift;

    my $bb_in  = APR::Brigade->new($c->pool, $c->bucket_alloc);
    my $bb_out = APR::Brigade->new($c->pool, $c->bucket_alloc);
    my $last = 0;

    while (1) {
        my $rv = $c->input_filters->get_brigade($bb_in,
                                                Apache::MODE_GETLINE);

        if ($rv != APR::SUCCESS or $bb_in->empty) {
            my $error = APR::strerror($rv);
            unless ($rv == APR::EOF) {
                warn "[eliza] get_brigade: $error\n";
            }
            $bb_in->destroy;
            last;
        }
    }
}
```



```

while (!$bb_in->empty) {
    my $bucket = $bb_in->first;

    $bucket->remove;

    if ($bucket->is_eos) {
        $bb_out->insert_tail($bucket);
        last;
    }

    my $data;
    my $status = $bucket->read($data);
    return $status unless $status == APR::SUCCESS;

    if ($data) {
        $data =~ s/[\r\n]*$//;
        $last++ if $data =~ /good bye/i;
        $data = $eliza->transform( $data ) . "\n\n";
        $bucket = APR::Bucket->new($data);
    }

    $bb_out->insert_tail($bucket);

```

```
    }

    my $b = APR::Bucket::flush_create($c->bucket_alloc);
    $bb_out->insert_tail($b);
    $c->output_filters->pass_brigade($bb_out);
    last if $last;
}

Apache::OK;
}
```

```
use base qw(Apache::Filter);
use constant BUFF_LEN => 1024;

sub lowercase : FilterConnectionHandler {
    my $filter = shift;

    while ($filter->read(my $buffer, BUFF_LEN)) {
        $filter->print(lc $buffer);
    }

    return Apache::OK;
}

1;
```

- Configuration

```
Listen 8085
```

```
<VirtualHost _default_:8085>
```

```
    PerlModule Apache::Eliza2
```

```
    PerlProcessConnectionHandler Apache::Eliza2
```

```
    PerlOutputFilterHandler Apache::Eliza2::lowercase
```

```
</VirtualHost>
```

1.11.6 I/O Filtering Phases

- `mod_perl` provides two interfaces to filtering:
 - a direct mapping to buckets and bucket brigades
 - and a simpler, stream-oriented interface

- mod_perl can do connection and request filtering.
- (Apache distinguish between more types)
- subroutine attributes set the filter type

```
sub handler : FilterRequestHandler { ... }  
sub handler : FilterConnectionHandler { ... }
```

1.11.6.1 PerlInputFilterHandler

- poor man's s/GET/HEAD/ rewrite
- The handler looks for data like:

```
GET /perl/test.pl HTTP/1.1
```

- and turns it into:

```
HEAD /perl/test.pl HTTP/1.1
```

```

package Apache::InputFilterGET2HEAD;

use strict;
use warnings;

use base qw(Apache::Filter);

use Apache::RequestRec ();
use Apache::RequestIO ();
use APR::Brigade ();
use APR::Bucket ();

use Apache::Const -compile => 'OK';
use APR::Const -compile => ':common';

sub handler : FilterConnectionHandler {
    my($filter, $bb, $mode, $block, $readbytes) = @_;

    my $c = $filter->c;
    my $ctx_bb = APR::Brigade->new($c->pool, $c->bucket_alloc);
    my $rv = $filter->next->get_brigade($ctx_bb, $mode, $block, $readbytes);
    return $rv unless $rv == APR::SUCCESS;

    while (!$ctx_bb->empty) {
        my $bucket = $ctx_bb->first;

        $bucket->remove;
    }
}

```



```

        if ($bucket->is_eos) {
            $bb->insert_tail($bucket);
            last;
        }

        my $data;
        my $status = $bucket->read($data);
        return $status unless $status == APR::SUCCESS;

        if ($data and $data =~ s|^GET|HEAD|) {
            $bucket = APR::Bucket->new($data);
        }

        $bb->insert_tail($bucket);
    }

    Apache::OK;
}
1;

```

```
Listen 8005
<VirtualHost _default_:8005>
    PerlInputFilterHandler +Apache::InputFilterGET2HEAD

    <Location />
        SetHandler modperl
        PerlResponseHandler +Apache::RequestType
    </Location>
</VirtualHost>
```

1.11.6.2 PerlOutputFilterHandler

- A stream oriented output filter
- `Apache::ROT13` implements the simple Caesar-cypher encryption
- so that *"mod_perl 2.0 rules!"* becomes *"zbq_crey 2.0 ehryf!"*
- `sizeof(English) == 26 =>` the ROT13 encryption is self-inverse,
- so the same code can be used for encoding and decoding

```
package Apache::ROT13;
use strict;

use Apache::RequestRec ();
use Apache::RequestIO ();
use Apache::Filter ();
use Apache::Const -compile => 'OK';
use constant BUFF_LEN => 1024;

sub handler {
    my $filter = shift;

    while ($filter->read(my $buffer, BUFF_LEN)) {
        $buffer =~ y/A-Za-z/N-ZA-Mn-za-m/;
        $filter->print($buffer);
    }
}
```

```
    return Apache::OK;  
}  
1;
```

```
PerlModule Apache::ROT13
Alias /perl-rot13/ /home/httpd/perl/
<Location /perl-rot13>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    PerlOutputFilterHandler Apache::ROT13
    Options +ExecCGI
    #PerlOptions +ParseHeaders
</Location>
```

1.12 References

- All the information can be found at:

`http://perl.apache.org/docs/`

- Further Questions?

`modperl@perl.apache.org`