1 Warnings and Errors Troubleshooting Index

1.1 Description

If you encounter an error or warning you don’t understand, check this chapter for solutions to common warnings and errors that the mod_perl community has encountered in the last few years.

1.2 General Advice

Perl’s warnings mode is immensely helpful in detecting possible problems. Make sure you always turn on warnings while you are developing code. See The Importance of Warnings.

Enabling `use diagnostics;` generally helps you to determine the source of the problem and how to solve it. See diagnostics pragma for more information.

1.3 Building and Installation

1.3.1 Bareword "gensym" not allowed while "strict subs"

When building mod_perl one may get the following failure:

```
make[1]: Entering directory ‘.../mod_perl-1.29/Symbol’
Running Mkbootstrap for Apache::Symbol ()
...
Bareword "gensym" not allowed while "strict subs" in use at
/usr/lib/perl5/5.8.3/Pod/Parser.pm line 1158.
...
make: *** [subdirs] Error 2
```

That happens when you you’ve modify `@INC` to push `"."` before all other directories, whereas it should be last. When `"."` is first in `@INC` it picks *mod_perl-1.29/Symbol/Symbol.pm* when it’s inside the directory *mod_perl-1.29/Symbol*. This shouldn’t happen if your system paths are coming first. To check your `@INC`, run:

```
% perl -V
```

and it’ll appear at the end of the output. Usually `"."` is pushed first by setting a `PERL5LIB` or a similar environment variable. Unset it and repeat the build process to solve the problem.

1.3.2 `No rule to make target ... CORE/config.h`

If while running `'make'` you get a message:

```
make: *** No rule to make target
‘/usr/lib/perl5/5.8.3/i386-linux/CORE/config.h’,
needed by ‘Makefile’.  Stop.
```

That means that your Perl installation is incomplete. Usually this is the case on package based distros, where perl is split across multiple packages. Usually you need to install the Perl-devel package to be able to build any other Perl modules that include XS extensions.

1.4 make test Issues

See make Troubleshooting and make test Troubleshooting

1.5 Configuration and Startup

This section talks about errors reported when you attempt to start the server.

1.5.1 'relocation errors' or 'undefined symbol'

Although the httpd executable was successfully built, you can still have make failures. The output could include the following errors:

```
relocation error:
undefined symbol: PL_dowarn
```

This class of errors is often due to multiple installations of Perl. Having *libperl.so* in */usr/lib/* is a great reason for lots of obscure problems, when you have one more perl installed elsewhere. That's why perl puts its *libperl.so* under its private tree (e.g., */usr/lib/perl5/5.8.3/i686/CORE/*). But some distributions decide to put it along with the rest of system libraries, not expecting that users will have extra perl installations.

To resolve the problem you need to check what perl library the application finds. This is easy to check with the help of `ldd(1)`.

If your mod_perl is statically linked with httpd, you need to check the httpd executable whether it's linked against *libperl.so*:

```
% ldd /path/to/apache/bin/httpd
```

If the output includes *libperl.so* check that the path is to the version of Perl you've built mod_perl with. If your httpd executable is reported to link against the wrong *libperl.so* file, you've found the cause of the problem. You should either ask your distributor to not put the perl library into the global system libs directory, or use some other solution to force the loading of the right library, which is usually very platform specific. For example on Linux one can preload a specific library path using the `LD_PRELOAD` environment variable. So if the wanted library is located at */usr/lib/perl5/5.8.3/i686/CORE/libperl.so* you can make httpd use it with:

```
% LD_PRELOAD=/usr/lib/perl5/5.8.3/i686/CORE/libperl.so httpd
```

for more information on Linux loader referer to the `ld.so(8)` `ldconfig(8)` manpages.

If mod_perl is built as DSO, you will need to check the mod_perl module (and not `httpd`) with `ldd`. Confusingly, mod_perl 1.0 module name is the same as of Perl: *libperl.so*. If for example the mod_perl module is located in */path/to/apache/libexec/*, the command would be:

```
% ldd /path/to/apache/libexec/libperl.so
```

There could be another variation of the problem where a Perl used to build mod_perl is statically linked and again, during the build time a wrong static archive (e.g., */usr/lib/libperl.a*) is picked by the linker. If Perl is statically linked, running:

```
% perl -V:useshrplib
```

will say:

```
useshrplib='false'
```

Again, the solution may vary from system to system, but moving */usr/lib/libperl.a* away while building mod_perl is probably the simplest.

1.5.2 SegFaults During Startup

Possible reasons and solutions:

- You have to build mod_perl with the same compiler as Perl was built with. Otherwise you may see segmentation faults and other weird things happen.

- This could be the XSLoader vs. DynaLoader problem, where the list of dl_handles created by XSLoader is wiped out by DynaLoader. Try adding:

  ```
  use DynaLoader ();
  ```

  to your *startup.pl* before any other module is loaded.

  This has been resolved in perl 5.8.0. For earlier versions of Perl you need to comment out:

  ```
  #@dl_librefs         = ();       # things we have loaded
  #@dl_modules         = ();       # Modules we have loaded
  ```

  in *DynaLoader.pm*.

However if none of this cases applies and you still experience segmentation faults, please report the problem using the following guidelines.

1.5.3 libexec/libperl.so: open failed: No such file or directory

If when you run the server you get the following error:

```
libexec/libperl.so: open failed: No such file or directory
```

it probably means that Perl was compiled with a shared library. mod_perl does detect this and links the Apache executable to the Perl shared library (*libperl.so*).

First of all make sure you have Perl installed on the machine, and that you have *libperl.so* in *<perl-root>/<version>/<architecture>/CORE*. For example in */usr/local/lib/perl5/5.00503/sun4-solaris/CORE*.

Then make sure that directory is included in the environment variable `LD_LIBRARY_PRELOAD`. Under normal circumstances, Apache should have the path configured at compile time, but this way you can override the library path.

1.5.4 install_driver(Oracle) failed: Can't load '.../DBD/Oracle/Oracle.so' for module DBD::Oracle

```
install_driver(Oracle) failed: Can't load
'/usr/lib/perl5/site_perl/5.005/i386-linux/auto/DBD/Oracle/Oracle.so'
for module DBD::Oracle:
libclntsh.so.8.0: cannot open shared object file:
No such file or directory at
/usr/lib/perl5/5.00503/i386-linux/DynaLoader.pm line 169.
at (eval 27) line 3
Perhaps a required shared
library or dll isn't installed where expected at
/usr/local/apache/perl/tmp.pl line 11
```

On BSD style filesystems `LD_LIBRARY_PATH` is not searched for setuid programs (a.k.a., Apache). This isn't a problem for CGI script since they don't do a setuid (and are forked off), but Apache does, and mod_perl is in Apache. Therefore the first solution is to explicitly load the library from the system wide *ldconfig* configuration file:

```
# echo $ORACLE_HOME/lib >> /etc/ld.so.conf
# ldconfig
```

Another solution to this problem is to modify the resulting *Makefile* (after running `perl Makefile.PL`) as follows:

1. search for the line `LD_RUN_PATH=`

2. replace it with `LD_RUN_PATH=(my_oracle_home)/lib`

(my_oracle_home) is, of course, the home path to your oracle installation. In particular, the file libclntsh.so.8.0 should exist in that directory. (If you use CPAN, the build directory for DBD::Oracle should be in ~/.cpan/build/DBD-Oracle-1.06/ if you're logged in as root.)

Then, just type `make install`, and all should go well.

FYI, setting `LD_RUN_PATH` has the effect of hard-coding the path to *(my_oracle_home)/lib* in the resulting `Oracle.so` file generated by the `DBD::Oracle` so that at run-time, it doesn't have to go searching through `LD_LIBRARY_PATH` or the default directories used by `ld`.

For more information see the *ld* manpage and an essay on `LD_LIBRARY_PATH`: http://www.visi.com/~barr/ldpath.html

1.5.5 Invalid command 'PerlHandler'...

```
Syntax error on line 393 of /etc/httpd/conf/httpd.conf: Invalid
command 'PerlHandler', perhaps mis-spelled or defined by a module
not included in the server configuration [FAILED]
```

This can happen when you have a mod_perl enabled Apache compiled with DSO (generally it's an installed RPM or other binary package) but the mod_perl module isn't loaded. In this case you have to tell Apache to load mod_perl by adding:

```
AddModule mod_perl.c
```

in your *httpd.conf*.

This can also happen when you try to run a non-mod_perl Apache server using the configuration from a mod_perl server.

1.5.6 symbol __floatdisf: referenced symbol not found

This problem is experienced by users on certain Solaris versions. When the server is built with modules that use the `__floatdisf` symbol it can't be started. e.g.:

```
Cannot load /usr/local/apache/libexec/libproxy.so into server:
ld.so.1: ../bin/httpd: fatal: relocation error: file
/usr/local/apache/libexec/libproxy.so: symbol __floatdisf: referenced
symbol not found
```

The missing symbol is in *libgcc.a*. Use

```
% gcc -print-libgcc-file-name
```

to see where that file is. Once found you have to relink the module with that file. You can also look for it in the gcc tree, e.g. under *gcc-3.2.1/gcc*.

First, configure and install Apache. Next, relink *mod_proxy.so* or *mod_negotiation.so*, or whatever the module that reports the problem with *libgcc.a*.

```
% cd apache_1.3.27/src/modules
% ld -G -o mod_proxy.so mod_proxy.lo /pathto/libgcc.a
```

(adjust the */pathto/* to point to the right file from the gcc output stage.)

You can now verify with `nm` that *mod_proxy.so* includes that symbol.

1.5.7 RegistryLoader: Translation of uri [...] to filename failed

```
RegistryLoader: Translation of uri [/home/httpd/perl/test.pl] to filename
        failed [tried: /home/httpd/docs/home/httpd/perl/test.pl]
```

This error shows up when `Apache::RegistryLoader` fails to translate the URI into the corresponding filesystem path. Most failures happen when one passes a file path instead of URI. (A reminder: */home/httpd/perl/test.pl* is a file path, while */perl/test.pl* is a URI). In most cases all you have to do is to pass something that `Apache::RegistryLoader` expects to get - the URI, but there are more complex cases. `Apache::RegistryLoader`'s man page shows how to handle these cases as well (look for the trans() sub).

1.5.8 "Apache.pm failed to load!"

If your server startup fails with:

```
Apache.pm failed to load!
```

try adding this to *httpd.conf*:

```
PerlModule Apache
```

1.6 Code Parsing and Compilation

1.6.1 Value of $x will not stay shared at - line 5

my () Scoped Variable in Nested Subroutines.

1.6.2 Value of $x may be unavailable at - line 5.

my () Scoped Variable in Nested Subroutines.

1.6.3 Can't locate loadable object for module XXX

There is no object built for this module. e.g. when you see:

```
Can't locate loadable object for module Apache::Util in @INC...
```

make sure to give mod_perl's `Makefile.PL` `PERL_UTIL_API=1`, `EVERYTHING=1` or `DYNAMIC=1` parameters to enable and build all the components of `Apache::Util`.

1.6.4 Can't locate object method "get_handlers"...

```
Can't locate object method "get_handlers" via package "Apache"
```

You need to rebuild your mod_perl with stacked handlers, i.e. `PERL_STACKED_HANDLERS=1` or more simply `EVERYTHING=1`.

1.6.5 Missing right bracket at line ...

Most often you will find that you really do have a syntax error. However the other reason might be that a script running under `Apache::Registry` is using `__DATA__` or `__END__` tokens. Learn why.

1.6.6 Can't load '.../auto/DBI/DBI.so' for module DBI

Check that all your modules are compiled with the same Perl that is compiled into mod_perl. Perl 5.005 and 5.004 are not binary compatible by default.

Other known causes of this problem:

OS distributions that ship with a broken binary Perl installation.

The 'perl' program and 'libperl.a' library are somehow built with different binary compatibility flags.

The solution to these problems is to rebuild Perl and any extension modules from a fresh source tree. Tip for running Perl's Configure script: use the '-des' flags to accepts defaults and '-D' flag to override certain attributes:

```
% ./Configure -des -Dcc=gcc ... && make test && make install
```

Read Perl's INSTALL document for more details.

Solaris OS specific:

"`Can't load DBI`" or similar error for the IO module or whatever dynamic module mod_perl tries to pull in first. The solution is to re-configure, re-build and re-install Perl and dynamic modules with the following flags when Configure asks for "`additional LD flags`":

```
-Xlinker --export-dynamic
```

or

```
-Xlinker -E
```

This problem is only known to be caused by installing gnu ld under Solaris.

1.7 Runtime

1.7.1 print() doesn’t send anything on Mac OS X

On certain Mac OS X builds mod_perl doesn’t seem to able to print() anything to the client. That’s because `STDOUT` is not tied to the `Apache` module. Most likely some core module on Mac OS X untie’s/re-tie’s `STDOUT` after mod_perl had it tied. The workaround is to add:

```
PerlHeaderParserHandler "sub { tie *STDOUT, 'Apache' unless tied *STDOUT; }"
```

Another solution is not to use un unqualified `print()`, but `$r->print()`, for example replace:

```
print "Hello";
```

with:

```
$r->print("Hello");
```

If you don’t have `$r`, you can obtain it with:

```
my $r = Apache->request;
```

1.7.2 "exit signal Segmentation fault (11)" with mysql

If you build mod_perl and mod_php in the same binary, you might get Segmentation fault followed by this error:

```
exit signal Segmentation fault (11)
```

The solution is to not rely on PHP’s built-in MySQL support, and instead build mod_php with your local MySQL support files by adding `--with-mysql=/path/to/mysql` during `./configure`.

1.7.3 foo ... at /dev/null line 0

Under mod_perl you may receive a warning or an error in the *error_log* which specifies */dev/null* as the source file, and line 0 as an line number where the printing of the message was triggered. This is absolutely normal if the code is executed from within a handler, because there is no actual file associated with the handler. Therefore `$0` is set to */dev/null* and that’s what you see.

1.7.4 Preventing mod_perl Processes From Going Wild

See the sections "Non-Scheduled Emergency Log Rotation" and "All RAM Consumed"

1.7.5 Segfaults when using XML::Parser

If you have some of the processes segfault when using `XML::Parser` you should use

```
--disable-rule=EXPAT
```

during the Apache configuration step.

Starting from mod_perl version 1.23 this option is disabled by default.

1.7.6 My CGI/Perl Code Gets Returned as Plain Text Instead of Being Executed by the Webserver

See My CGI/Perl Code Gets Returned as Plain Text Instead of Being Executed by the Webserver.

1.7.7 Incorrect line number reporting in error/warn log messages

See Use of uninitialized value at (eval 80) line 12.

1.7.8 rwrite returned -1

This message happens when the client breaks the connection while your script is trying to write to the client. With Apache 1.3.x, you should only see the rwrite messages if `LogLevel` is set to `debug`.

There was a bug that reported this debug message regardless of the value of the `LogLevel` directive. It was fixed in mod_perl 1.19_01.

Generally `LogLevel` is either `debug` or `info`. `debug` logs everything, `info` is the next level, which doesn't include debug messages. You shouldn't use "debug" mode on your production server. At the moment there is no way to prevent users from aborting connections.

1.7.9 Can't upgrade that kind of scalar ...

Fixed in mod_perl 1.23.

1.7.10 caught SIGPIPE in process

```
[modperl] caught SIGPIPE in process 1234
[modperl] process 1234 going to Apache::exit with status...
```

That's the `$SIG{PIPE}` handler installed by mod_perl/`Apache::SIG`, which is called if a connection times out or if the client presses the 'Stop' button. It gives you an opportunity to do cleanups if the script was aborted in the middle of its execution. See Handling the 'User pressed Stop button' case for more info.

If your mod_perl version is earlier than 1.17 you might also get the message in the following section...

1.7.11 Client hit STOP or Netscape bit it!

```
Client hit STOP or Netscape bit it!
Process 2493 going to Apache::exit with status=-2
```

You may see this message in mod_perl versions less than 1.17. See also caught SIGPIPE in process.

1.7.12 Global symbol "$foo" requires explicit package name

The script below will print a warning like that above, moreover it will print the whole script as a part of the warning message:

```
#!/usr/bin/perl -w
use strict;
print "Content-type: text/html\n\n";
print "Hello $undefined";
```

The warning:

```
Global symbol "$undefined" requires
explicit package name at /usr/apps/foo/cgi/tmp.pl line 4.
        eval 'package Apache::ROOT::perl::tmp_2epl;
use Apache qw(exit);sub handler {
#line 1 /usr/apps/foo/cgi/tmp.pl
BEGIN {$^W = 1;}#!/usr/bin/perl -w
use strict;
print "Content-type: text/html\\n\\n";
print "Hello $undefined";


}
;' called at
/usr/apps/lib/perl5/site_perl/5.005/aix/Apache/Registry.pm
line 168
        Apache::Registry::compile('package
  Apache::ROOT::perl::tmp_2epl;use Apache qw(exit);sub han...')
  called at
      /usr/apps/lib/perl5/site_perl/5.005/aix/Apache/Registry.pm
      line 121
      Apache::Registry::handler('Apache=SCALAR(0x205026c0)')
      called at /usr/apps/foo/cgi/tmp.pl line 4
        eval {...} called at /usr/apps/foo/cgi/tmp.pl line 4
[Sun Nov 15 15:15:30 1998] [error] Undefined subroutine
&Apache::ROOT::perl::tmp_2epl::handler called at
/usr/apps/lib/perl5/site_perl/5.005/aix/Apache/Registry.pm
line 135.

[Sun Nov 15 15:15:30 1998] [error] Goto undefined subroutine
&Apache::Constants::SERVER_ERROR at
/usr/apps/lib/perl5/site_perl/5.005/aix/Apache/Constants.pm
line 23.
```

The error is simple to fix. When you use the `use strict;` pragma (and you should...), Perl will insist that all variables are defined before being used, so the error will not arise.

The bad thing is that sometimes the whole script (possibly, thousands of lines) is printed to the *error_log* file as code that the server has tried to `eval()`uate.

May be you have a `$SIG{__DIE__}` handler installed (`Carp::confess()`?). If so that's what's expected.

You might wish to try something more terse such as "local $SIG{__WARN__} = \&Carp::cluck;" The confess method is *very* verbose and will tell you more than you might wish to know including full source.

1.7.13 Use of uninitialized value at (eval 80) line 12.

Your code includes some undefined variable that you have used as if it was already defined and initialized. For example:

```
$param = $q->param('test');
print $param;
```

vs.

```
$param = $q->param('test') || '';
print $param;
```

In the second case, `$param` will always be *defined*, either with `$q->param('test')`'s return value or the default value (`''` empty string in our example).

Also read about Finding the Line Which Triggered the Error or Warning.

1.7.14 Undefined subroutine &Apache::ROOT::perl::test_2epl::some_function called at

See Names collisions with Modules and libs.

1.7.15 Callback called exit

Callback called exit is just a generic message when some unrecoverable error occurs inside Perl during `perl_call_sv()` (which mod_perl uses to invoke all handler subroutines. Such problems seem to occur far less with 5.005_03 than 5.004.

Sometimes you discover that your server is not responding and its error_log has filled up the remaining space on the file system. When you get to see the contents of the error_log -- it includes millions of lines, like:

```
Callback called exit at -e line 33, <HTML> chunk 1.
```

Why the looping?

Perl can get *very* confused inside an infinite loop in your code. It doesn't necessarily mean that your code did call `exit()`. Perl's malloc went haywire and called `croak()`, but no memory is left to properly report the error, so Perl is stuck in a loop writing that same message to stderr.

Perl 5.005+ plus is recommended for its improved malloc.c and other features that improve mod_perl and are turned on by default.

See also Out of memory

1.7.16 Out of memory!

If something goes really wrong with your code, Perl may die with an "Out of memory!" message and/or "Callback called exit". Common causes of this are never-ending loops, deep recursion, or calling an undefined subroutine.

If you are using perl 5.005 or later, and perl is compiled to use it's own malloc rutines, you can trap out of memory errors by setting aside an extra memory pool in the special variable `$^M`. By default perl uses the operating system malloc for many popular systems, so unless you build perl with 'usemymalloc=y' you probably wont be able to use `$^M`. Run:

```
% perl -V:usemymalloc
```

if your mod_perl was compiled against perl which uses internal `malloc()` the answer will be 'y'.

Here is an explanation of `$^M` from the `perlvar` manpage:

```
By default, running out of memory is an untrap- pable, fatal
error.  However, if suitably built, Perl can use the contents of
$^M as an emergency memory pool after die()ing.  Suppose that
your Perl were compiled with -DPERL_EMERGENCY_SBRK and used
Perl's malloc.  Then

    $^M = 'a' x (1 << 16);

would allocate a 64K buffer for use in an emer- gency.  See the
INSTALL file in the Perl distribu- tion for information on how
to enable this option.  To discourage casual use of this
advanced feature, there is no English long name for this
variable.
```

If your perl installation supports $^M and you add '`use Apache::Debug level => 4;`' to your Perl script, it will allocate the `$^M` emergency pool and the `$SIG{__DIE__}` handler will call `Carp::confess`, giving you a stack trace which should reveal where the problem is. See the `Apache::Resource` module for ways to control httpd processes.

Note that Perl 5.005 and later have `PERL_EMERGENCY_SBRK` turned on by default.

Another trick is to have a startup script initialize `Carp::confess`, like so:

```
use Carp ();
eval { Carp::confess("init") };
```

this way, when the real problem happens, `Carp::confess` doesn't eat memory in the emergency pool (`$^M`).

Some other mod_perl users have reported that this works well for them:

```
# Allocate 64K as an emergency memory pool for use in out of
# memory situation
$^M = 0x00 x 65536;

# Little trick to initialize this routine here so that in the case
# of OOM, compiling this routine doesn't eat memory from the
# emergency memory pool $^M
use CGI::Carp ();
eval { CGI::Carp::confess('init') };

# Importing CGI::Carp sets $main::SIG{__DIE__} = \&CGI::Carp::die;
# Override that to additionally give a stack backtrace
$main::SIG{__DIE__} = \&CGI::Carp::confess;
```

Discussion of `$^M` has come up on PerlMonks, and there is speculation that `$^M` is a forgotten feature that's not well supported. See http://perlmonks.org/index.pl?node_id=287850 for more information.

1.7.17 server reached MaxClients setting, consider raising the MaxClients setting

See Choosing MaxClients.

1.7.18 syntax error at /dev/null line 1, near "line arguments:"

```
syntax error at /dev/null line 1, near "line arguments:"
Execution of /dev/null aborted due to compilation errors.
parse: Undefined error: 0
```

There is a chance that your `/dev/null` device is broken. Try:

```
% echo > /dev/null
```

Alternatively you should try to remove this special file and recreate it:

```
# rm /dev/null
# mknod /dev/null c 1 3
# chmod a+rw /dev/null
```

1.7.19 Can’t call method "register_cleanup" (CGI.pm)

```
Can’t call method "register_cleanup" on an
undefined value at /usr/lib/perl5/5.00503/CGI.pm line 263.
```

caused by this code snippet in *CGI.pm*:

```
if ($MOD_PERL) {
  Apache->request->register_cleanup(\&CGI::_reset_globals);
  undef $NPH;
}
```

One solution is to add to *httpd.conf*:

```
PerlPostReadRequestHandler ’sub { Apache->request(shift) }’
```

But even better, switch to `Apache::Cookie`:

```
use Apache;
use Apache::Cookie;

sub handler {
  my $r = shift;
  my $cookies = Apache::Cookie->new($r)->parse;
  my %bar = $cookies->{foo}->value;
}
```

1.7.20 readdir() not working

If readdir() either fails with an exception, or in the list context it returns the correct number of items but each item as an empty string, you have a binary compatibility between mod_perl and Perl problem. Most likely the two have been built against different *glibc* versions, which have incompatible `struct dirent`.

To solve this problem rebuild mod_perl and Perl against the same *glibc* version or get new binary packages built against the same *glibc* version.

1.8 Shutdown and Restart

1.8.1 Evil things might happen when using PerlFreshRestart

`PerlFreshRestart` affects the graceful restart process for non-DSO mod_perl builds. If you have mod_perl enabled Apache built as DSO and you restart it, the whole Perl interpreter is completely torn down (perl_destruct())and restarted. The value of `PerlFreshRestart` is irrelevant at this point.

Unfortunately, not all perl modules are robust enough to survive reload. For them this is an unusual situation. `PerlFreshRestart` does not much more than:

```
while (my ($k,$v) = each %INC) {
  delete $INC{$k};
  require $k;
}
```

Besides that, it flushes the `Apache::Registry` cache, and empties any dynamic stacked handlers (e.g. `PerlChildInitHandler`).

Some users, who had turned `PerlFreshRestart` **On**, reported having segfaults, others have seen no problems starting the server, no errors written to the logs, but no server running after a restart. Most of the problems have gone away when it was turned `Off`.

It doesn't mean that you shouldn't use it, if it works for you. Just beware of the dragons...

1.8.2 Constant subroutine XXX redefined

That's a mandatory warning inside Perl which happens only if you modify your script and Apache::Registry reloads it. Perl is warning you that the subroutine(s) were redefined. It is mostly harmless. If you don't like seeing these warnings, just `kill -USR1` (graceful restart) Apache when you modify your scripts.

You aren't supposed to see these warnings if you don't modify the code with perl 5.004_05 or 5.005+.and higher. If you still experience a problem with code within a CGI script, moving all the code into a module (or a library) and require()ing it should solve the problem.

1.8.3 Can't undef active subroutine

```
Can't undef active subroutine at
/usr/apps/lib/perl5/site_perl/5.005/aix/Apache/Registry.pm line 102.
Called from package Apache::Registry, filename
/usr/apps/lib/perl5/site_perl/5.005/aix/Apache/Registry.pm, line 102
```

This problem is caused when a client drops the connection while httpd is in the middle of a write. httpd times out, sending a SIGPIPE, and Perl (in that child) is stuck in the middle of its eval context. This is fixed by the Apache::SIG module which is called by default. This should not happen unless you have code that is messing with **$SIG{PIPE}**. It's also triggered only when you've changed your script on disk and mod_perl is trying to reload it.

1.8.4 [warn] child process 30388 did not exit, sending another SIGHUP

From mod_perl.pod: With Apache versions 1.3.0 and higher, mod_perl will call the perl_destruct() Perl API function during the child exit phase. This will cause proper execution of `END` blocks found during server startup as well as invoking the `DESTROY` method on global objects which are still alive. It is possible that this operation may take a long time to finish, causing problems during a restart. If your code does not contain any `END` blocks or `DESTROY` methods which need to be run during child server shutdown, this destruction can be avoided by setting the `PERL_DESTRUCT_LEVEL` environment variable to `-1`. Be aware that 'your code' includes any modules you use and *they* may well have `DESTROY` and `END` blocks...

1.8.5 Processes Get Stuck on Graceful Restart

If you see a process stuck in "G" (Gracefully finishing) after a doing a graceful restart (sending kill -SIGUSR1) it means that the process is hanging in perl_destruct() while trying to cleanup. This cleanup normally isn't a requirement, you can disable it by setting the PERL_DESTRUCT_LEVEL environment variable to -1. See the section "Speeding up the Apache Termination and Restart" for more information.

1.8.6 httpd keeps on growing after each restart

See the *HUP Signal* explanation at the section: Server Stopping and Restarting

Also, be aware that `<Perl>` sections can also cause leaks during graceful restarts. See the (sub)thread: http://aspn.activestate.com/ASPN/Mail/Message/modperl/304620 (META: we don't have code samples, so if you have such please let us know).

1.9 OS Specific Notes

1.9.1 RedHat Linux

1.9.1.1 RH 8 and 9 Locale Issues

RedHat 8 and 9 ship Perl with a locale setting that breaks perl. To solve the problem either change the locale to en_US.ISO8859-1 or simply remove the UTF8 part.

For more information refer to: http://twiki.org/cgi-bin/view/Codev/UsingPerl58OnRedHat8 http://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=87682

1.9.2 OpenBSD

1.9.2.1 Issues Caused by httpd being `chroot`'ed

Since OpenBSD 3.2, the Apache `httpd(8)` server has been `chroot(2)`ed by default.

A mod_perl enabled OpenBSD httpd will NOT work as is. A proper `chroot(2)` environment must be setup for the default chrooted behavior to work. The `-u` option to httpd disables the chroot behavior, and returns the httpd to the expanded "unsecure" behavior. See the OpenBSD `httpd(8)` man page.

For more information see *section 10.16 of the OpenBSD FAQ*.
http://www.openbsd.org/faq/faq10.html#httpdchroot.

1.9.3 Win FU

1.9.3.1 Apache::DBI

`Apache::DBI` causes the server to exit when it starts up, with:

```
[Mon Oct 25 15:06:11 1999] file .\main\http_main.c, line 5890,
assertion "start_mutex" failed
```

Solution: build mod_perl with `PERL_STARTUP_DONE_CHECK` set (e.g. insert

```
#define PERL_STARTUP_DONE_CHECK 1
```

at the top of *mod_perl.h* or add it to the defines in the MSVC++ and similar applications' `Options` dialog).

Apache loads all Apache modules twice, to make sure the server will successfully restart when asked to. This flag disables all `PerlRequire` and `PerlModule` statements on the first load, so they can succeed on the second load. Without that flag, the second load fails.

1.9.4 HP-UX

1.9.4.1 `Apache::Status` Dumps Core Under HP-UX 10.20

HP-UX 10.20 may dump core when accessing *perl-status?sig*. This issue is known to happen when perl's *./Configure* doesn't detect that `SIGRTMAX` is defined, but not implemented on that platform.

One solution is to upgrade to a recent version of Perl (5.8.0?) that properly detects the implementation of that signal.

Another solution is to modify *Apache/Status.pm* to skip that broken signal by replacing the line:

```
sort keys %SIG),
```

with:

```
sort grep { $_ ne 'RTMAX' } keys %SIG),
```

1.10 Problematic Perl Modules

Here is the list of Perl Modules that are known to have problems under mod_perl 1.0 and possible workarounds, solutions.

- **`IPC::Open*`**

  use `IPC::Run` instead. It provides the same functionality as the `IPC::Open*` family and more... and it works fine with mod_perl.

1.11 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- Stas Bekman [http://stason.org/]

1.12 Authors

- Stas Bekman [http://stason.org/]

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: