

1 Getting Help

1.1 Description

If your question isn't answered by reading this guide, check this section for information on how to get help on mod_perl, Apache, or other topics discussed here.

1.2 READ ME FIRST

If, after reading this guide, the general docs and the other documents listed on this site, you still don't have the answers, please ask for help on the mod_perl users mailing list. But please, first try to browse the mailing list archive. Most of the time you will find the answers to your questions by searching the archive, since it is very likely that someone else has already encountered the same problem and found a solution for it. If you ignore this advice, you should not be surprised if your question is left unanswered -- it bores people to answer the same question again and again. This does not mean that you should avoid asking questions, but you should not abuse the available help and you should *RTFM* before you call for *HELP*. (Remember the fable of the shepherd boy and the wolves).

Another possibility is to look in the general Getting Help section of this site for a commercial training or consulting company.

1.2.1 Please Ask Only Questions Related to mod_perl

If you have general Apache questions, please refer to: <http://httpd.apache.org/lists.html>.

If you have general Perl questions, please refer to: <http://lists.perl.org/>.

For other remotely related to mod_perl questions see the references to other documentation.

Finally, if you are posting to the list for the first time, please refer to the mod_perl mailing lists' Guidelines.

1.3 How to Report Problems

Make sure to include a good subject like explaining the problem in a few words. Also please mention that this a problem with mod_perl 1.0 (since now we have mod_perl 2.0 too). Here is an example of a good subject:

```
Subject: [mpl] response handler randomly segfaults
```

The most important thing to understand is that you should try hard to provide **all** the information that may assist to understand and reproduce the problem. When you prepare a bug report, put yourself in the position of a person who is going to try to help you, realizing that a guess-work on behalf of that helpful person, more often doesn't work than it does. Unfortunately most people don't realize that, and it takes several emails to squeeze the needed details from the person reporting the bug, a process which may drag for days.

Always send the following details:

- Anything in the *error_log* file that looks suspicious and possibly related to the problem.
- Output of `perl -V`
- Version of `mod_perl` (hint: it's logged into the *error_log* file when the server has just started)
- Version of `apache` (hint: it's logged into the *error_log* file when the server has just started)
- Options given to `mod_perl's` `Makefile.PL` while building it. If you are using a pre-compiled binary (e.g., rpm), find the source package that was used to build this binary and retrieve this information from there.
- Server configuration details (that's the relevant parts of your *httpd.conf*, usually just the relevant `mod_perl` configuration sections).
- Relevant sections of your `ErrorLog` (make test's is: `t/logs/error_log`)
- If some other code doesn't work, minimize that code to a minimal size while it reproduces the problem and attach it to the report.
- If you build from source, make sure that `make test` passes 100%.

If `make test` fails, run the failing tests separately, using the verbose mode and attach the output of the run and the relevant sections of your `ErrorLog` to the report.

If this is a script which doesn't use `mod_perl` api, try to test under `mod_cgi` if applicable

You should try to isolate the problem and send the smallest possible code snippet, that reproduces the problem.

Remember, that if we cannot reproduce the problem, we might not be able to solve it.

To further increase the chances that bugs your code exposes will be investigated, try using `Apache-Test` to create a self-contained environment that core developers can use to easily reproduce your bug. To get you started, an `Apache-Test` bug skeleton has been created:

<http://perl.apache.org/~geoff/bug-reporting-skeleton-mp1.tar.gz>

Detailed instructions are contained within the `README`.

- **Getting the Backtrace From Core Dumps**

If you get a *core* file dump (*Segmentation fault*), please send a backtrace if possible. Before you try to produce it, re-build `mod_perl` with:

```
panic% perl Makefile.PL PERL_DEBUG=1
```

which will:

- **add `-g` to `EXTRA_CFLAGS`**
- **turn on `PERL_TRACE`**
- **set `PERL_DESTRUCT_LEVEL=2` (additional checks during Perl cleanup)**
- **link against *libperl* if it exists**

Here is a summary of how to get a backtrace:

```
% cd mod_perl-x.xx
% touch t/conf/srm.conf
% gdb ../apache_x.xx/src/httpd
(gdb) run -X -f 'pwd'/t/conf/httpd.conf -d 'pwd'/t
[now make request that causes core dump]
(gdb) bt
```

So you go to the `mod_perl` source directory, create an empty `srm.conf` file, and start `gdb` with a path to the `httpd` binary, which is at least located in the Apache source tree after you built it. (Of course replace `x` with real version numbers). Next step is to start the `httpd` from within `gdb` and issue a request, which causes a core dump. when the code has died with `SEGV` signal, run `bt` to get the backtrace.

Alternatively you can also attach to an already running process like so:

```
% gdb httpd <process id number>
```

If the dump is happening in *libperl* you have to rebuild Perl with `-DDEBUGGING` enabled. A quick way to this is to go to your Perl source tree and run these commands:

```
% rm *.lo
% make LIBPERL=libperl.a
% cp libperl.a $Config{archlibexp}/CORE
```

where `$Config{archlibexp}` is:

```
% perl -V:archlibexp
```

If the trace includes Apache calls and you see no arguments, you need to rebuild Apache with `--without-execstrip`. If building a static `mod_perl`, you need to rebuild it with:

```
% perl Makefile.PL ... APACI_ARGS='--without-execstrip'
```

● Spinning Processes

The `gdb` attaching to the live process approach is helpful when debugging a *spinning* process. You can also get a Perl stacktrace of a *spinning* process by install a `$SIG{USR1}` handler in your code:

```
use Carp ();  
$SIG{USR1} = \&Carp::confess;
```

While the process is spinning, send it a *USR1* signal:

```
% kill -USR1 <process id number>
```

and the Perl stack trace will be printed.

alternatively you can use gdb to find which Perl code is causing the spin:

```
% gdb httpd <pid of spinning process>  
(gdb) where  
(gdb) source mod_perl-x.xx/.gdbinit  
(gdb) curinfo
```

After loading the special macros file (*.gdbinit*) you can use the *curinfo* gdb macro to figure out the file and line number the code stuck in.

Finally send all these details to the modperl mailing list.

1.4 Help on Related Topics

When developing with mod_perl, you often find yourself having questions regarding other projects and topics like Apache, Perl, SQL, etc. This document will help you find the right resource where you can find the answers to your questions.

1.5 Maintainers

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

- Stas Bekman <stas (at) stason.org>

1.6 Authors

- Stas Bekman <stas (at) stason.org>

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

Table of Contents:

1	Getting Help	1
1.1	Description	2
1.2	READ ME FIRST	2
1.2.1	Please Ask Only Questions Related to mod_perl	2
1.3	How to Report Problems	2
1.4	Help on Related Topics	5
1.5	Maintainers	5
1.6	Authors	5