

# **1 Apache2::Log - Perl API for Apache Logging Methods**

## 1.1 Synopsis

```

# in startup.pl
#-----
use Apache2::Log;

use Apache2::Const -compile => qw(OK :log);
use APR::Const    -compile => qw(:error SUCCESS);

my $s = Apache2::ServerUtil->server;

$s->log_error("server: log_error");
$s->log_error(__FILE__, __LINE__, Apache2::Const::LOG_ERR,
             APR::Const::SUCCESS, "log_error logging at err level");
$s->log_error(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
             APR::Const::ENOTIME, "debug print");
Apache2::ServerRec->log_error("routine warning");

Apache2::ServerRec::warn("routine warning");

# in a handler
#-----
package Foo;

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

use Apache2::Log;

use Apache2::Const -compile => qw(OK :log);
use APR::Const    -compile => qw(:error SUCCESS);

sub handler {
    my $r = shift;
    $r->log_error("request: log_error");

    my $rlog = $r->log;
    for my $level qw(emerg alert crit error warn notice info debug) {
        no strict 'refs';
        $rlog->$level($package, "request: $level log level");
    }

    # can use server methods as well
    my $s = $r->server;
    $s->log_error("server: log_error");

    $r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
                 APR::Const::ENOTIME, "in debug");

    $s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_INFO,
                 APR::Const::SUCCESS, "server info");

    $s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_ERR,
                 APR::Const::ENOTIME, "fatal error");

    $r->log_reason("fatal error");

```

```

    $r->warn('routine request warning');
    $s->warn('routine server warning');

    return Apache2::Const::OK;
}
1;

# in a registry script
# httpd.conf: PerlOptions +GlobalRequest
use Apache2::ServerRec qw(warn); # override warn locally
print "Content-type: text/plain\n\n";
warn "my warning";

```

## 1.2 Description

Apache2::Log provides the Perl API for Apache logging methods.

Depending on the the current `LogLevel` setting, only logging with the same log level or higher will be loaded. For example if the current `LogLevel` is set to *warning*, only messages with log level of the level *warning* or higher (*err*, *crit*, *elert* and *emerg*) will be logged. Therefore this:

```

$r->log_error(Apache2::Log::LOG_MARK, Apache2::Const::LOG_WARNING,
             APR::Const::ENOTIME, "warning!");

```

will log the message, but this one won't:

```

$r->log_error(Apache2::Log::LOG_MARK, Apache2::Const::LOG_INFO,
             APR::Const::ENOTIME, "just an info");

```

It will be logged only if the server log level is set to *info* or *debug*. `LogLevel` is set in the configuration file, but can be changed using the `$s->loglevel()` method.

The filename and the line number of the caller are logged only if `Apache2::Const::LOG_DEBUG` is used (because that's how Apache 2.0 logging mechanism works).

Note: On Win32 Apache attempts to lock all writes to a file whenever it's opened for append (which is the case with logging functions), as Unix has this behavior built-in, while Win32 does not. Therefore `Apache2::Log` functions could be slower than Perl's `print()/warn()`.

## 1.3 Constants

Log level constants can be compiled all at once:

```

use Apache2::Const -compile => qw(:log);

```

or individually:

```

use Apache2::Const -compile => qw(LOG_DEBUG LOG_INFO);

```

## 1.3.1 *LogLevel Constants*

The following constants (sorted from the most severe level to the least severe) are used in logging methods to specify the log level at which the message should be logged:

**1.3.1.1 Apache2::Const::LOG\_EMERG**

**1.3.1.2 Apache2::Const::LOG\_ALERT**

**1.3.1.3 Apache2::Const::LOG\_CRIT**

**1.3.1.4 Apache2::Const::LOG\_ERR**

**1.3.1.5 Apache2::Const::LOG\_WARNING**

**1.3.1.6 Apache2::Const::LOG\_NOTICE**

**1.3.1.7 Apache2::Const::LOG\_INFO**

**1.3.1.8 Apache2::Const::LOG\_DEBUG**

## 1.3.2 *Other Constants*

Make sure to compile the APR status constants before using them. For example to compile `APR::Const::SUCCESS` and all the APR error status constants do:

```
use APR::Const    -compile => qw(:error SUCCESS);
```

Here is the rest of the logging related constants:

**1.3.2.1 Apache2::Const::LOG\_LEVELMASK**

used to mask off the level value, to make sure that the log level's value is within the proper bits range. e.g.:

```
$loglevel &= LOG_LEVELMASK;
```

**1.3.2.2 Apache2::Const::LOG\_TOCLIENT**

used to give content handlers the option of including the error text in the `ErrorDocument` sent back to the client. When `Apache2::Const::LOG_TOCLIENT` is passed to `log_rerror()` the error message will be saved in the `$r`'s notes table, keyed to the string `"error-notes"`, if and only if the severity level of the message is `Apache2::Const::LOG_WARNING` or greater and there are no other `"error-notes"` entry already set in the request record's notes table. Once the `"error-notes"` entry is set, it is up to the error handler to determine whether this text should be sent back to the client. For example:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(ENOTIME);
$r->log_rerror(Apache2::Log::LOG_MARK,
              Apache2::Const::LOG_ERR|Apache2::Const::LOG_TOCLIENT,
              APR::Const::ENOTIME,
              "request log_rerror");
```

now the log message can be retrieved via:

```
$r->notes->get("error-notes");
```

Remember that client-generated text streams sent back to the client **MUST** be escaped to prevent CSS attacks.

### 1.3.2.3 Apache2::Const::LOG\_STARTUP

is useful for startup message where no timestamps, logging level is wanted. For example:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(SUCCESS);
$s->log_serror(Apache2::Log::LOG_MARK,
              Apache2::Const::LOG_INFO,
              APR::Const::SUCCESS,
              "This log message comes with a header");
```

will print:

```
[Wed May 14 16:47:09 2003] [info] This log message comes with a header
```

whereas, when `Apache2::Const::LOG_STARTUP` is binary ORed as in:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(SUCCESS);
$s->log_serror(Apache2::Log::LOG_MARK,
              Apache2::Const::LOG_INFO|Apache2::Const::LOG_STARTUP,
              APR::Const::SUCCESS,
              "This log message comes with no header");
```

then the logging will be:

```
This log message comes with no header
```

## 1.4 Server Logging Methods

### 1.4.1 `$s->log`

get a log handle which can be used to log messages of different levels.

```
my $slog = $s->log;
```

- **obj:** `$s` ( `Apache2::ServerRec` object )
- **ret:** `$slog` ( `Apache2::Log::Server` object )

`Apache2::Log::Server` object to be used with `LogLevel` methods.

- **since:** 2.0.00

## 1.4.2 `$s->log_error`

just logs the supplied message to `error_log`

```
$s->log_error(@message);
```

- **obj:** `$s` ( `Apache2::ServerRec` object )
- **arg1:** `@message` ( strings ARRAY )

what to log

- **ret:** no return value
- **since:** 2.0.00

For example:

```
$s->log_error("running low on memory");
```

## 1.4.3 `$s->log_serror`

This function provides a fine control of when the message is logged, gives an access to built-in status codes.

```
$s->log_serror($file, $line, $level, $status, @message);
```

- **obj:** `$s` ( `Apache2::ServerRec` object )
- **arg1:** `$file` ( string )

The file in which this function is called

- **arg2:** `$line` ( number )

The line number on which this function is called

- **arg3:** `$level` ( `Apache2::LOG_*` constant )

The level of this error message

- **arg4:** `$status` ( `APR::Const` status constant )

The status code from the last command (similar to `#!` in perl), usually `APR::Const` constant or coming from an exception object.

- **arg5: @message ( strings ARRAY )**

The log message(s)

- **ret: no return value**
- **since: 2.0.00**

For example:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(ENOTIME SUCCESS);
$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_ERR,
              APR::Const::SUCCESS, "log_serror logging at err level");

$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
              APR::Const::ENOTIME, "debug print");
```

### 1.4.4 *\$s->warn*

```
$s->warn(@warnings);
```

is the same as:

```
$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_WARNING,
              APR::Const::SUCCESS, @warnings)
```

- **obj: \$s ( Apache2::ServerRec object )**
- **arg1: @warnings ( strings ARRAY )**

array of warning strings

- **ret: no return value**
- **since: 2.0.00**

For example:

```
$s->warn('routine server warning');
```

## 1.5 Request Logging Methods

### 1.5.1 *\$r->log*

get a log handle which can be used to log messages of different levels.

```
$rlog = $r->log;
```

- **obj: \$r ( Apache2::RequestRec object )**
- **ret: \$rlog ( Apache2::Log::Request object )**

`Apache2::Log::Request` object to be used with `LogLevel` methods.

- **since: 2.0.00**

## 1.5.2 `$r->log_error`

just logs the supplied message (similar to `$s->log_error`).

```
$r->log_error(@message);
```

- **obj:** `$r` (`Apache2::RequestRec` object)
- **arg1:** `@message` (strings ARRAY)

what to log

- **ret:** no return value
- **since: 2.0.00**

For example:

```
$r->log_error("the request is about to end");
```

## 1.5.3 `$r->log_reason`

This function provides a convenient way to log errors in a preformatted way:

```
$r->log_reason($message);
$r->log_reason($message, $filename);
```

- **obj:** `$r` (`Apache2::RequestRec` object)
- **arg1:** `$message` (string)

the message to log

- **opt arg2:** `$filename` (string)

where to report the error as coming from (e.g. `__FILE__`)

- **ret:** no return value
- **since: 2.0.00**

For example:

```
$r->log_reason("There is no enough data");
```

will generate a log entry similar to the following:

```
[Fri Sep 24 11:58:36 2004] [error] access to /someuri
failed for 127.0.0.1, reason: There is no enough data.
```

### 1.5.4 *\$r->log\_rerror*

This function provides a fine control of when the message is logged, gives an access to built-in status codes.

```
$r->log_rerror($file, $line, $level, $status, @message);
```

arguments are identical to `$s->log_serror`.

- **since: 2.0.00**

For example:

```
use Apache2::Const -compile => qw(:log);
use APR::Const -compile => qw(ENOTIME SUCCESS);
$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_ERR,
              APR::Const::SUCCESS, "log_rerror logging at err level");

$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
              APR::Const::ENOTIME, "debug print");
```

### 1.5.5 *\$r->warn*

```
$r->warn(@warnings);
```

is the same as:

```
$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_WARNING,
              APR::Const::SUCCESS, @warnings)
```

- **obj: \$r ( Apache2::RequestRec object )**
- **arg1: @warnings ( strings ARRAY )**

array of warning strings

- **ret: no return value**
- **since: 2.0.00**

For example:

```
$r->warn('routine server warning');
```

## 1.6 Other Logging Methods

## 1.6.1 *LogLevel Methods*

after getting the log handle with `$s->log` or `$r->log`, use one of the following methods (corresponding to the LogLevel levels):

```
emerg(), alert(), crit(), error(), warn(), notice(), info(), debug()
```

to control when messages should be logged:

```
$s->log->emerg(@message);  
$r->log->emerg(@message);
```

- **obj:** `$slog` ( server or request log handle )
- **arg1:** `@message` ( strings ARRAY )
- **ret:** no return value
- **since:** 2.0.00

For example if the LogLevel is `error` and the following code is executed:

```
my $slog = $s->log;  
$slog->debug("just ", "some debug info");  
$slog->warn(@warnings);  
$slog->crit("dying");
```

only the last command's logging will be performed. This is because `warn`, `debug` and other logging command which are listed right to `error` will be disabled.

### 1.6.2 *alert*

See LogLevel Methods.

### 1.6.3 *crit*

See LogLevel Methods.

### 1.6.4 *debug*

See LogLevel Methods.

### 1.6.5 *emerg*

See LogLevel Methods.

### ***1.6.6 error***

See LogLevel Methods.

### ***1.6.7 info***

See LogLevel Methods.

### ***1.6.8 notice***

See LogLevel Methods.

Though Apache treats `notice()` calls as special. The message is always logged regardless the value of `ErrorLog`, unless the error log is set to use `syslog`. (For details see `httpd-2.0/server/log.c`.)

### ***1.6.9 warn***

See LogLevel Methods.

## **1.7 General Functions**

### ***1.7.1 LOG\_MARK***

Though looking like a constant, this is a function, which returns a list of two items: (`__FILE__`, `__LINE__`), i.e. the file and the line where the function was called from.

```
my ($file, $line) = Apache2::Log::LOG_MARK();
```

- **ret1:** `$file` (string)
- **ret2:** `$line` (number)
- **since:** 2.0.00

It's mostly useful to be passed as the first argument to those logging methods, expecting the filename and the line number as the first arguments (e.g., `$s->log_error` and `$r->log_error`).

## **1.8 Virtual Hosts**

Code running from within a virtual host needs to be able to log into its `ErrorLog` file, if different from the main log. Calling any of the logging methods on the `$r` and `$s` objects will do the logging correctly.

If the core `warn()` is called, it'll be always logged to the main log file. Here is how to make it log into the vhost `error_log` file. Let's say that we start with the following code:

```
warn "the code is smoking";
```

1. First, we need to use `mod_perl`'s logging function, instead of `CORE::warn`

Either replace `warn` with `Apache2::ServerRec::warn`:

```
use Apache2::Log ();
Apache2::ServerRec::warn("the code is smoking");
```

or import it into your code:

```
use Apache2::ServerRec qw(warn); # override warn locally
warn "the code is smoking";
```

or override `CORE::warn`:

```
use Apache2::Log ();
*CORE::GLOBAL::warn = \&Apache2::ServerRec::warn;
warn "the code is smoking";
```

Avoid using the latter suggestion, since it'll affect all the code running on the server, which may break things. Of course you can localize that as well:

```
use Apache2::Log ();
local *CORE::GLOBAL::warn = \&Apache2::ServerRec::warn;
warn "the code is smoking";
```

Chances are that you need to make the internal Perl warnings go into the vhost's `error_log` file as well. Here is how to do that:

```
use Apache2::Log ();
local $SIG{__WARN__} = \&Apache2::ServerRec::warn;
eval q[my $x = "aaa" + 1;]; # this issues a warning
```

Notice that it'll override any previous setting you may have had, disabling modules like `CGI::Carp` which also use `$SIG{__WARN__}`

2. Next we need to figure out how to get hold of the vhost's server object.

Inside HTTP request handlers this is possible via `Apache2->request`. Which requires either `PerlOptions +GlobalRequest` setting or can be also done at runtime if `$r` is available:

```
use Apache2::RequestUtil ();
sub handler {
    my $r = shift;
    Apache2::RequestUtil->request($r);
    ...
}
```

Outside HTTP handlers at the moment it is not possible, to get hold of the vhost's `error_log` file. This shouldn't be a problem for the code that runs only under `mod_perl`, since the always available `$s` object can invoke a plethora of methods supplied by `Apache2::Log`. This is only a problem for modules, which are supposed to run outside `mod_perl` as well.

META: To solve this we think to introduce 'PerlOptions +GlobalServer', a big brother for 'PerlOptions +GlobalRequest', which will be set in `modperl_hook_pre_connection`.

## 1.9 Unsupported API

`Apache2::Log` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the `mod_perl` development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

### 1.9.1 `log_pid`

META: what is this method good for? it just calls `getpid` and logs it. In any case it has nothing to do with the logging API. And it uses static variables, it probably shouldn't be in the Apache public API.

Log the current pid

```
Apache2::Log::log_pid($pool, $fname);
```

- **obj: \$p (APR::Pool object)**

The pool to use for logging

- **arg1: \$fname (file path)**

The name of the file to log to

- **ret: no return value**
- **since: subject to change**

## 1.10 See Also

`mod_perl 2.0` documentation.

## 1.11 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 2.0.

## 1.12 Authors

The `mod_perl` development team and numerous contributors.



## Table of Contents:

1	Apache2::Log - Perl API for Apache Logging Methods	1
1.1	Synopsis	2
1.2	Description	3
1.3	Constants	3
1.3.1	LogLevel Constants	4
1.3.1.1	Apache2::Const::LOG_EMERG	4
1.3.1.2	Apache2::Const::LOG_ALERT	4
1.3.1.3	Apache2::Const::LOG_CRIT	4
1.3.1.4	Apache2::Const::LOG_ERR	4
1.3.1.5	Apache2::Const::LOG_WARNING	4
1.3.1.6	Apache2::Const::LOG_NOTICE	4
1.3.1.7	Apache2::Const::LOG_INFO	4
1.3.1.8	Apache2::Const::LOG_DEBUG	4
1.3.2	Other Constants	4
1.3.2.1	Apache2::Const::LOG_LEVELMASK	4
1.3.2.2	Apache2::Const::LOG_TOCLIENT	4
1.3.2.3	Apache2::Const::LOG_STARTUP	5
1.4	Server Logging Methods	5
1.4.1	\$s->log	5
1.4.2	\$s->log_error	6
1.4.3	\$s->log_serror	6
1.4.4	\$s->warn	7
1.5	Request Logging Methods	7
1.5.1	\$r->log	7
1.5.2	\$r->log_error	8
1.5.3	\$r->log_reason	8
1.5.4	\$r->log_rerror	9
1.5.5	\$r->warn	9
1.6	Other Logging Methods	9
1.6.1	LogLevel Methods	10
1.6.2	alert	10
1.6.3	crit	10
1.6.4	debug	10
1.6.5	emerg	10
1.6.6	error	11
1.6.7	info	11
1.6.8	notice	11
1.6.9	warn	11
1.7	General Functions	11
1.7.1	LOG_MARK	11
1.8	Virtual Hosts	11
1.9	Unsupported API	13
1.9.1	log_pid	13
1.10	See Also	13

Table of Contents:

1.11 Copyright . . . . .	13
1.12 Authors . . . . .	13