1 Apache::Log - Perl API for Apache Logging Methods

1.1 Synopsis

```
#in startup.pl
#-------------
use Apache::Log;

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

my $s = Apache->server;

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

Apache->warn("routine warning");
Apache::warn("routine warning");
Apache::Server->warn("routine warning");

#in a handler
#------------
use Apache::Log;

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

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

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

    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(Apache::Log::LOG_MARK, Apache::LOG_DEBUG,
                   APR::ENOTIME, "in debug");

    $s->log_serror(Apache::Log::LOG_MARK, Apache::LOG_INFO,
                   APR::SUCESS, "server info");

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

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

    return Apache::OK;
}
```

1.2 Description

`Apache::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_rerror(Apache::Log::LOG_MARK, Apache::LOG_WARNING,
               APR::ENOTIME, "warning!");
```

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

```
$r->log_rerror(Apache::Log::LOG_MARK, Apache::LOG_INFO,
               APR::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 `Apache::LOG_DEBUG` is used (because that's how Apache 2.0 logging mechanism works).

1.3 Constants

Log level constants can be compiled all at once:

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

or individually:

```
use Apache::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 `Apache::LOG_EMERG`

1.3.1.2 Apache::LOG_ALERT

1.3.1.3 Apache::LOG_CRIT

1.3.1.4 Apache::LOG_ERR

1.3.1.5 Apache::LOG_WARNING

1.3.1.6 Apache::LOG_NOTICE

1.3.1.7 Apache::LOG_INFO

1.3.1.8 Apache::LOG_DEBUG

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

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

1.3.2 Other Constants

1.3.2.1 Apache::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 Apache::LOG_TOCLIENT

used to give content handlers the option of including the error text in the ErrorDocument sent back to the client. When Apache::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 Apache::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:

```
$r->log_rerror(Apache::Log::LOG_MARK, Apache::LOG_ERR|Apache::LOG_TOCLIENT,
               APR::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 `Apache::LOG_STARTUP`

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

```
$s->log_serror(Apache::Log::LOG_MARK, Apache::LOG_INFO,
               APR::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 `Apache::LOG_STARTUP` is binary ORed as in:

```
$s->log_serror(Apache::Log::LOG_MARK, Apache::LOG_INFO|Apache::LOG_STARTUP,
               APR::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_error`

just logs the supplied message to *error_log*

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

- **arg1: `$s` (`Apache::Server`)**
- **arg2: `@message` (ARRAY)**

  what to log

- **ret: (XXX)**

For example:

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

1.4.2 `$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);
```

- **arg1: `$s` (`Apache::Server`)**
- **arg2: `$file` (string)**

The file in which this function is called

- **arg3: `$line` (number)**

  The line number on which this function is called

- **arg4: `$level` (`Apache::LOG_* constant`)**

  The level of this error message

- **arg5: `$status` (integer)**

  The status code from the last command (similar to $! in perl, usually APR:: status constant)

- **arg6: `@message` (ARRAY)**

  The log message

- **ret: (XXX)**

For example:

```
$s->log_serror(Apache::Log::LOG_MARK, Apache::LOG_ERR,
               APR::SUCCESS, "log_serror logging at err level");

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

1.4.3 `$s->log`

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

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

- **arg1: `$s` (`Apache::Server`)**
- **ret: `$slog` (scalar)**

1.5 Request Logging Methods

1.5.1 `$r->log_error`

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

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

- **arg1: `$r` (`Apache::RequestRec`)**
- **arg2: `@message` (ARRAY)**

what to log

- **ret: (XXX)**

For example:

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

1.5.2 $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`.

For example:

```
$r->log_rerror(Apache::Log::LOG_MARK, Apache::LOG_ERR,
               APR::SUCCESS, "log_rerror logging at err level");

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

1.5.3 $r->log

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

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

- **arg1: `$r` (`Apache::RequestRec`)**
- **ret: `$rlog` (scalar)**

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);
```

- **arg1: `$slog` (log handle)**
- **arg2: `@message` (ARRAY)**

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 `emerg`

See LogLevel Methods.

1.6.3 `alert`

See LogLevel Methods.

1.6.4 `crit`

See LogLevel Methods.

1.6.5 `error`

See LogLevel Methods.

1.6.6 `warn`

See LogLevel Methods.

1.6.7 `notice`

See LogLevel Methods.

1.6.8 `info`

See LogLevel Methods.

1.6.9 debug

See LogLevel Methods.

1.7 General Functions

1.7.1 Apache::Log::LOG_MARK

```
($file, $line) = Apache::Log::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.

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_serror and $r->log_rerror).

1.7.2 Apache::Log::log_pid

Log the current pid of the parent process

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

- **arg1: $p (APR::Pool)**

  The pool to use for logging

- **arg2: $fname (APR::Pool)**

  The name of the file to log to

- **ret: no return value**

1.8 Aliases

1.8.1 $s->warn

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

is the same as:

```
$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_WARNING,
              APR::SUCCESS, @warnings)
```

For example:

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

1.8.2 Apache->warn

1.8.3 Apache::warn

```
Apache->warn(@warnings);
```

1.9 See Also

mod_perl 2.0 documentation.

1.10 Copyright

1.11 Authors

The mod_perl development team and numerous contributors.

Table of Contents:

Table of Contents: