

NAME

Sys::Syslog - Perl interface to the UNIX syslog(3) calls

VERSION

This is the documentation of version 0.33

SYNOPSIS

```
use Sys::Syslog;                               # all except setlogsock()
use Sys::Syslog qw(:standard :macros);        # standard functions & macros

openlog($ident, $logopt, $facility);          # don't forget this
syslog($priority, $format, @args);
$oldmask = setlogmask($mask_priority);
closelog();
```

DESCRIPTION

Sys::Syslog is an interface to the UNIX syslog(3) program. Call syslog() with a string priority and a list of printf() args just like syslog(3).

EXPORTS

Sys::Syslog exports the following Exporter tags:

- `:standard` exports the standard syslog(3) functions:
openlog closelog setlogmask syslog
- `:extended` exports the Perl specific functions for syslog(3):
setlogsock
- `:macros` exports the symbols corresponding to most of your syslog(3) macros and the LOG_UPTO() and LOG_MASK() functions. See *CONSTANTS* for the supported constants and their meaning.

By default, Sys::Syslog exports the symbols from the `:standard` tag.

FUNCTIONS

openlog(\$ident, \$logopt, \$facility)

Opens the syslog. \$ident is prepended to every message. \$logopt contains zero or more of the options detailed below. \$facility specifies the part of the system to report about, for example LOG_USER or LOG_LOCAL0: see *Facilities* for a list of well-known facilities, and your syslog(3) documentation for the facilities available in your system. Check *SEE ALSO* for useful links. Facility can be given as a string or a numeric macro.

This function will croak if it can't connect to the syslog daemon.

Note that openlog() now takes three arguments, just like openlog(3).

You should use openlog() before calling syslog().

Options

- `cons` - This option is ignored, since the failover mechanism will drop down to the console automatically if all other media fail.
- `ndelay` - Open the connection immediately (normally, the connection is opened when the first message is logged).
- `noeol` - When set to true, no end of line character (`\n`) will be appended to the message. This can be useful for some buggy syslog daemons.

- `nofatal` - When set to true, `openlog()` and `syslog()` will only emit warnings instead of dying if the connection to the syslog can't be established.
- `nonul` - When set to true, no NUL character (`\0`) will be appended to the message. This can be useful for some buggy syslog daemons.
- `nowait` - Don't wait for child processes that may have been created while logging the message. (The GNU C library does not create a child process, so this option has no effect on Linux.)
- `perror` - Write the message to standard error output as well to the system log (added in Sys::Syslog 0.22).
- `pid` - Include PID with each message.

Examples

Open the syslog with options `ndelay` and `pid`, and with facility `LOCAL0`:

```
openlog($name, "ndelay,pid", "local0");
```

Same thing, but this time using the macro corresponding to `LOCAL0`:

```
openlog($name, "ndelay,pid", LOG_LOCAL0);
```

`syslog($priority, $message)`

`syslog($priority, $format, @args)`

If `$priority` permits, logs `$message` or `sprintf($format, @args)` with the addition that `%m` in `$message` or `$format` is replaced with `"$!"` (the latest error message).

`$priority` can specify a level, or a level and a facility. Levels and facilities can be given as strings or as macros. When using the `eventlog` mechanism, priorities `DEBUG` and `INFO` are mapped to event type `informational`, `NOTICE` and `WARNING` to `warning` and `ERR` to `EMERG` to `error`.

If you didn't use `openlog()` before using `syslog()`, `syslog()` will try to guess the `$ident` by extracting the shortest prefix of `$format` that ends in a `":"`.

Examples

```
# informational level
syslog("info", $message);
syslog(LOG_INFO, $message);

# information level, Local0 facility
syslog("info|local0", $message);
syslog(LOG_INFO|LOG_LOCAL0, $message);
```

Note

Sys::Syslog version v0.07 and older passed the `$message` as the formatting string to `sprintf()` even when no formatting arguments were provided. If the code calling `syslog()` might execute with older versions of this module, make sure to call the function as `syslog($priority, "%s", $message)` instead of `syslog($priority, $message)`. This protects against hostile formatting sequences that might show up if `$message` contains tainted data.

`setlogmask($mask_priority)`

Sets the log mask for the current process to `$mask_priority` and returns the old mask. If the mask argument is 0, the current log mask is not modified. See *Levels* for the list of available levels. You can use the `LOG_UPTO()` function to allow all levels up to a given priority (but it only accept the numeric macros as arguments).

Examples

Only log errors:

```
setlogmask( LOG_MASK(LOG_ERR) );
```

Log everything except informational messages:

```
setlogmask( ~(LOG_MASK(LOG_INFO)) );
```

Log critical messages, errors and warnings:

```
setlogmask( LOG_MASK(LOG_CRIT)
            | LOG_MASK(LOG_ERR)
            | LOG_MASK(LOG_WARNING) );
```

Log all messages up to debug:

```
setlogmask( LOG_UPTO(LOG_DEBUG) );
```

setlogsock()

Sets the socket type and options to be used for the next call to `openlog()` or `syslog()`. Returns true on success, undef on failure.

Being Perl-specific, this function has evolved along time. It can currently be called as follow:

- `setlogsock($sock_type)`
- `setlogsock($sock_type, $stream_location)` (added in Perl 5.004_02)
- `setlogsock($sock_type, $stream_location, $sock_timeout)` (added in Sys::Syslog 0.25)
- `setlogsock(\%options)` (added in Sys::Syslog 0.28)

The available options are:

- `type` - equivalent to `$sock_type`, selects the socket type (or "mechanism"). An array reference can be passed to specify several mechanisms to try, in the given order.
- `path` - equivalent to `$stream_location`, sets the stream location. Defaults to standard Unix location, or `_PATH_LOG`.
- `timeout` - equivalent to `$sock_timeout`, sets the socket timeout in seconds. Defaults to 0 on all systems except Mac OS X where it is set to 0.25 sec.
- `host` - sets the hostname to send the messages to. Defaults to the local host.
- `port` - sets the TCP or UDP port to connect to. Defaults to the first standard syslog port available on the system.

The available mechanisms are:

- "native" - use the native C functions from your `syslog(3)` library (added in Sys::Syslog 0.15).
- "eventlog" - send messages to the Win32 events logger (Win32 only; added in Sys::Syslog 0.19).
- "tcp" - connect to a TCP socket, on the `syslog/tcp` or `syslogng/tcp` service. See also the `host`, `port` and `timeout` options.
- "udp" - connect to a UDP socket, on the `syslog/udp` service. See also the `host`, `port` and `timeout` options.
- "inet" - connect to an INET socket, either TCP or UDP, tried in that order. See also

the host, port and timeout options.

- "unix" - connect to a UNIX domain socket (in some systems a character special device). The name of that socket is given by the `path` option or, if omitted, the value returned by the `_PATH_LOG` macro (if your system defines it), `/dev/log` or `/dev/console`, whichever is writable.
- "stream" - connect to the stream indicated by the `path` option, or, if omitted, the value returned by the `_PATH_LOG` macro (if your system defines it), `/dev/log` or `/dev/console`, whichever is writable. For example Solaris and IRIX system may prefer "stream" instead of "unix".
- "pipe" - connect to the named pipe indicated by the `path` option, or, if omitted, to the value returned by the `_PATH_LOG` macro (if your system defines it), or `/dev/log` (added in Sys::Syslog 0.21). HP-UX is a system which uses such a named pipe.
- "console" - send messages directly to the console, as for the "cons" option of `openlog()`.

The default is to try `native`, `tcp`, `udp`, `unix`, `pipe`, `stream`, `console`. Under systems with the Win32 API, `eventlog` will be added as the first mechanism to try if Win32::EventLog is available.

Giving an invalid value for `$sock_type` will croak.

Examples

Select the UDP socket mechanism:

```
setlogsock("udp");
```

Send messages using the TCP socket mechanism on a custom port:

```
setlogsock({ type => "tcp", port => 2486 });
```

Send messages to a remote host using the TCP socket mechanism:

```
setlogsock({ type => "tcp", host => $loghost });
```

Try the native, UDP socket then UNIX domain socket mechanisms:

```
setlogsock(["native", "udp", "unix"]);
```

Note

Now that the "native" mechanism is supported by Sys::Syslog and selected by default, the use of the `setlogsock()` function is discouraged because other mechanisms are less portable across operating systems. Authors of modules and programs that use this function, especially its cargo-cult form `setlogsock("unix")`, are advised to remove any occurrence of it unless they specifically want to use a given mechanism (like TCP or UDP to connect to a remote host).

closelog()

Closes the log file and returns true on success.

THE RULES OF SYS::SYSLOG

The First Rule of Sys::Syslog is: You do not call `setlogsock`.

The Second Rule of Sys::Syslog is: You **do not** call `setlogsock`.

The Third Rule of Sys::Syslog is: The program crashes, dies, calls `closelog`, the log is over.

The Fourth Rule of Sys::Syslog is: One facility, one priority.

The Fifth Rule of Sys::Syslog is: One log at a time.

The Sixth Rule of Sys::Syslog is: No syslog before openlog.

The Seventh Rule of Sys::Syslog is: Logs will go on as long as they have to.

The Eighth, and Final Rule of Sys::Syslog is: If this is your first use of Sys::Syslog, you must read the doc.

EXAMPLES

An example:

```
openlog($program, 'cons,pid', 'user');
syslog('info', '%s', 'this is another test');
syslog('mail|warning', 'this is a better test: %d', time);
closelog();

syslog('debug', 'this is the last test');
```

Another example:

```
openlog("$program $$", 'ndelay', 'user');
syslog('notice', 'fooprogram: this is really done');
```

Example of use of %m:

```
#!/ = 55;
syslog('info', 'problem was %m'); # %m == ! in syslog(3)
```

Log to UDP port on \$remotehost instead of logging locally:

```
setlogsock("udp", $remotehost);
openlog($program, 'ndelay', 'user');
syslog('info', 'something happened over here');
```

CONSTANTS

Facilities

- LOG_AUDIT - audit daemon (IRIX); falls back to LOG_AUTH
- LOG_AUTH - security/authorization messages
- LOG_AUTHPRIV - security/authorization messages (private)
- LOG_CONSOLE - /dev/console output (FreeBSD); falls back to LOG_USER
- LOG_CRON - clock daemons (**cron** and **at**)
- LOG_DAEMON - system daemons without separate facility value
- LOG_FTP - FTP daemon
- LOG_KERN - kernel messages
- LOG_INSTALL - installer subsystem (Mac OS X); falls back to LOG_USER
- LOG_LAUNCHD - launchd - general bootstrap daemon (Mac OS X); falls back to LOG_DAEMON
- LOG_LFMT - logalert facility; falls back to LOG_USER
- LOG_LOCAL0 through LOG_LOCAL7 - reserved for local use

- LOG_LPR - line printer subsystem
- LOG_MAIL - mail subsystem
- LOG_NETINFO - NetInfo subsystem (Mac OS X); falls back to LOG_DAEMON
- LOG_NEWS - USENET news subsystem
- LOG_NTP - NTP subsystem (FreeBSD, NetBSD); falls back to LOG_DAEMON
- LOG_RAS - Remote Access Service (VPN / PPP) (Mac OS X); falls back to LOG_AUTH
- LOG_REMOTEAUTH - remote authentication/authorization (Mac OS X); falls back to LOG_AUTH
- LOG_SECURITY - security subsystems (firewalling, etc.) (FreeBSD); falls back to LOG_AUTH
- LOG_SYSLOG - messages generated internally by **syslogd**
- LOG_USER (default) - generic user-level messages
- LOG_UUCP - UUCP subsystem

Levels

- LOG_EMERG - system is unusable
- LOG_ALERT - action must be taken immediately
- LOG_CRIT - critical conditions
- LOG_ERR - error conditions
- LOG_WARNING - warning conditions
- LOG_NOTICE - normal, but significant, condition
- LOG_INFO - informational message
- LOG_DEBUG - debug-level message

DIAGNOSTICS

Invalid argument passed to setlogsock

(F) You gave setlogsock() an invalid value for \$sock_type.

eventlog passed to setlogsock, but no Win32 API available

(W) You asked setlogsock() to use the Win32 event logger but the operating system running the program isn't Win32 or does not provides Win32 compatible facilities.

no connection to syslog available

(F) syslog() failed to connect to the specified socket.

stream passed to setlogsock, but %s is not writable

(W) You asked setlogsock() to use a stream socket, but the given path is not writable.

stream passed to setlogsock, but could not find any device

(W) You asked setlogsock() to use a stream socket, but didn't provide a path, and Sys::Syslog was unable to find an appropriate one.

tcp passed to setlogsock, but tcp service unavailable

(W) You asked setlogsock() to use a TCP socket, but the service is not available on the system.

syslog: expecting argument %s

(F) You forgot to give `syslog()` the indicated argument.

```
syslog: invalid level/facility: %s
```

(F) You specified an invalid level or facility.

```
syslog: too many levels given: %s
```

(F) You specified too many levels.

```
syslog: too many facilities given: %s
```

(F) You specified too many facilities.

```
syslog: level must be given
```

(F) You forgot to specify a level.

```
udp passed to setlogsock, but udp service unavailable
```

(W) You asked `setlogsock()` to use a UDP socket, but the service is not available on the system.

```
unix passed to setlogsock, but path not available
```

(W) You asked `setlogsock()` to use a UNIX socket, but `Sys::Syslog` was unable to find an appropriate an appropriate device.

HISTORY

`Sys::Syslog` is a core module, part of the standard Perl distribution since 1990. At this time, modules as we know them didn't exist, the Perl library was a collection of `.pl` files, and the one for sending syslog messages with was simply `lib/syslog.pl`, included with Perl 3.0. It was converted as a module with Perl 5.0, but had a version number only starting with Perl 5.6. Here is a small table with the matching Perl and `Sys::Syslog` versions.

<code>Sys::Syslog</code>	Perl
-----	----
undef	5.0.0 ~ 5.5.4
0.01	5.6.*
0.03	5.8.0
0.04	5.8.1, 5.8.2, 5.8.3
0.05	5.8.4, 5.8.5, 5.8.6
0.06	5.8.7
0.13	5.8.8
0.22	5.10.0
0.27	5.8.9, 5.10.1 ~ 5.14.2
0.29	5.16.0, 5.16.1

SEE ALSO

Other modules

Log::Log4perl - Perl implementation of the Log4j API

Log::Dispatch - Dispatches messages to one or more outputs

Log::Report - Report a problem, with exceptions and language support

Manual Pages

syslog(3)

SUSv3 issue 6, IEEE Std 1003.1, 2004 edition,
<http://www.opengroup.org/onlinepubs/000095399/basedefs/syslog.h.html>

GNU C Library documentation on syslog,

http://www.gnu.org/software/libc/manual/html_node/Syslog.html Solaris 10 documentation on syslog,
<http://docs.sun.com/app/docs/doc/816-5168/syslog-3c?a=view>

Mac OS X documentation on syslog,
<http://developer.apple.com/documentation/Darwin/Reference/ManPages/man3/syslog.3.html>

IRIX 6.5 documentation on syslog,
<http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=man&fname=3c+syslog>

AIX 5L 5.3 documentation on syslog,
<http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/syslog.htm>

HP-UX 11i documentation on syslog, <http://docs.hp.com/en/B2355-60130/syslog.3C.html>

Tru64 5.1 documentation on syslog,
http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51_HTML/MAN/MAN3/0193____.HTM

Stratus VOS 15.1,
<http://stratadoc.stratus.com/vos/15.1.1/r502-01/wwhelp/wwhimpl/js/html/wwhelp.htm?context=r502-01&file=ch5r502-01bi.html>

RFCs

RFC 3164 - The BSD syslog Protocol, <http://www.faqs.org/rfcs/rfc3164.html> -- Please note that this is an informational RFC, and therefore does not specify a standard of any kind.

RFC 3195 - Reliable Delivery for syslog, <http://www.faqs.org/rfcs/rfc3195.html>

Articles

Syslogging with Perl, <http://lexington.pm.org/meetings/022001.html>

Event Log

Windows Event Log,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wes/wes/windows_event_log.asp

AUTHORS & ACKNOWLEDGEMENTS

Tom Christiansen <[christ\(at\)perl.com](mailto:christ(at)perl.com)> and Larry Wall <[larry\(at\)wall.org](mailto:larry(at)wall.org)>.

UNIX domain sockets added by Sean Robinson <[robinson_s\(at\)sc.maricopa.edu](mailto:robinson_s(at)sc.maricopa.edu)> with support from Tim Bunce <[Tim.Bunce\(at\)ig.co.uk](mailto:Tim.Bunce(at)ig.co.uk)> and the `perl5-porters` mailing list.

Dependency on `syslog.ph` replaced with XS code by Tom Hughes <[tom\(at\)compton.nu](mailto:tom(at)compton.nu)>.

Code for `constant()` regenerated by Nicholas Clark <[nick\(at\)ccl4.org](mailto:nick(at)ccl4.org)>.

Failover to different communication modes by Nick Williams <[Nick.Williams\(at\)morganstanley.com](mailto:Nick.Williams(at)morganstanley.com)>.

Extracted from core distribution for publishing on the CPAN by Sébastien Aperghis-Tramoni <[sebastien\(at\)aperghis.net](mailto:sebastien(at)aperghis.net)>.

XS code for using native C functions borrowed from `Unix::Syslog`, written by Marcus Harnisch <[marcus.harnisch\(at\)gmx.net](mailto:marcus.harnisch(at)gmx.net)>.

Yves Orton suggested and helped for making `Sys::Syslog` use the native event logger under Win32 systems.

Jerry D. Hedden and Reini Urban provided greatly appreciated help to debug and polish `Sys::Syslog` under Cygwin.

BUGS

Please report any bugs or feature requests to `bug-sys-syslog` (at) `rt.cpan.org`, or through the web interface at <http://rt.cpan.org/Public/Dist/Display.html?Name=Sys-Syslog>. I will be notified, and then you'll automatically be notified of progress on your bug as I make changes.

SUPPORT

You can find documentation for this module with the `perldoc` command.

```
perldoc Sys::Syslog
```

You can also look for information at:

* AnnoCPAN: Annotated CPAN documentation

<http://annocpan.org/dist/Sys-Syslog>

* CPAN Ratings

<http://cpanratings.perl.org/d/Sys-Syslog>

* RT: CPAN's request tracker

<http://rt.cpan.org/Dist/Display.html?Queue=Sys-Syslog>

* Search CPAN

<http://search.cpan.org/dist/Sys-Syslog/>

* MetaCPAN

<https://metacpan.org/module/Sys::Syslog>

* Perl Documentation

<http://perldoc.perl.org/Sys/Syslog.html>

COPYRIGHT

Copyright (C) 1990-2012 by Larry Wall and others.

LICENSE

This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

Notes for the future maintainer (even if it's still me..) - - - - -

Using Google Code Search, I search who on Earth was relying on `$host` being public. It found 5 hits:

* First was inside Indigo Star Perl2exe documentation. Just an old version of Sys::Syslog.

* One real hit was inside DalWeathDB, a weather related program. It simply does a

```
$Sys::Syslog::host = '127.0.0.1';
```

- <http://www.gallistel.net/nparker/weather/code/>

* Two hits were in TPC, a fax server thingy. It does a

```
$Sys::Syslog::host = $TPC::LOGHOST;
```

but also has this strange piece of code:

```
# work around perl5.003 bug
sub Sys::Syslog::hostname {}
```

I don't know what bug the author referred to.

- <http://www.tpc.int/> - <ftp://ftp-usa.tpc.int/pub/tpc/server/UNIX/>

* Last hit was in Filefix, which seems to be a FIDOnet mail program (!). This one does not use \$host, but has the following piece of code:

```
sub Sys::Syslog::hostname
{
    use Sys::Hostname;
    return hostname;
}
```

I guess this was a more elaborate form of the previous bit, maybe because of a bug in Sys::Syslog back then?

- <ftp://ftp.kiae.su/pub/unix/fido/>

Links ---- Linux Fast-STREAMS - <http://www.openss7.org/streams.html>

II12021: SYSLOGD HOWTO TCPIPINFO (z/OS, OS/390, MVS) -
<http://www-1.ibm.com/support/docview.wss?uid=isg1II12021>

Getting the most out of the Event Viewer - <http://www.codeproject.com/dotnet/evtvwr.asp?print=true>

Log events to the Windows NT Event Log with JNI -
<http://www.javaworld.com/javaworld/jw-09-2001/jw-0928-ntmessages.html>