syslog-ng reference manual

Balázs Scheidler

syslog-ng reference manual

by Balázs Scheidler

Table of Contents

List of Tables

List of Examples

Chapter 1. Introduction to syslog-ng

One of the most neglected area of Unix is handling system events. Daily checks for system messages is crucial for the security and health conditions of a computer system.

System logs contain much "noise" - messages which have no importance - and on the contrary important events, which should not be lost in the load of messages. With current tools it’s difficult to select which messages we are interested in.

A message is sent to different destinations based on the assigned facility/priority pair. There are 12+8 (12 real and 8 local) predefined facilities (mail, news, auth etc.), and 8 different priorities (ranging from alert to debug).

One problem is that there are facilities which are too general (daemon), and these facilities are used by many programs, even if they do not relate each other. It is difficult to find the interesting bits from the enourmous amount of messages.

A second problem is that there are very few programs which allow setting their "facility code" to log under. It’s at best a compile time parameter.

So using facilities as a means of filtering is not the best way. For it to be a good solution would require runtime option for all applications, which specifies the log facility to log under, and the ability to create new facilities in syslogd. Neither of these are available, and the first is neither feasible.

One of the design principles of syslog-ng was to make message filtering much more finegrained. syslog-ng is able to filter messages based on the contents of messages in addition to the priority/facility pair. This way only the messages we are really interested in get to a specific destination. Another design principle was to make logforwarding between firewalled segments easier: long hostname format, which makes it easy to find the originating and chain of forwarding hosts even if a log message traverses several computers. And last principle was a clean and powerful configuration file format.

Chapter 2. Message paths

In syslog-ng a message path (or message route) consist of one or more sources, one or more filtering rules and one or more destinations (sinks). A message is entered to syslog-ng in one of its sources, if that message matches the filtering rules it goes out using one of the destinations.

2.1. Sources

A source is a collection of source drivers, which collect messages using a given method. For instance there's a source driver for AF_UNIX, SOCK_STREAM style sockets, which is used by the Linux syslog() call.

To declare a source, you'll need to use the source statement in the configuration file with the following syntax:

```
  source <identifier> { source-driver(params); source-
driver(params); ... };
```

The identifier has to uniquely identify this given source and of course may not clash with any of the reserved words (in case you had a nameclash, simply enclose the identifier in quotation marks)

You can control exactly which drivers are used to gather log messages, thus you'll have to know how your system and its native syslogd communicate. Here's a introduction to the inner workings of syslogd on some of the platforms I tested:

Table 2-1. Communication method between syslogd and its clients

Platform	Method
Linux	A SOCK_STREAM unix socket named /dev/log

Platform	Method
BSD flavors	A SOCK_DGRAM unix socket named /var/run/log
Solaris (2.5 or below)	An SVR4 style STREAMS device named /dev/log
Solaris (2.6 or above)	In addition to the STREAMS device used in versions below 2.6, uses a new multithreaded IPC method called door. By default the door used by syslogd is /etc/.syslog_door

Each possible communication mechanism has the corresponding source driver in syslog-ng. For instance to open a unix socket with SOCK_DGRAM style communication you use the driver unix-dgram, the same with SOCK_STREAM style - as used under Linux - is called unix-stream.

Example 2-1. Source statement on a Linux based operating system

```
  source src { unix-
stream("/dev/log"); internal(); udp(ip(0.0.0.0) port(514)); };
```

Each driver may take parameters, some of them required, some of them optional. The required parameters are usually positional, which means that they have to come first. See the unix-stream driver specification above, as it refers to the file /dev/log.

Table 2-2. Available source drivers in syslog-ng

Name	Description
internal	Messages generated internally in syslog-ng

Name	**Description**
unix-stream	Opens the specified unix socket in SOCK_STREAM mode, and listens for messages.
unix-dgram	Opens the specified unix socket in SOCK_DGRAM mode, and listens for messages.
file	Opens the specified file, and reads messages.
pipe, fifo	Opens the specified named pipe and reads messages
udp	Listens on the specified UDP port for messages.
tcp	Listens on the specified TCP port for messages.
sun-stream, sun-streams	Opens the specified STREAMS device on Solaris systems, and reads messages.

For a complete descriptions on the above drivers, see the reference section.

2.2. Filters

Filters perform log routing inside syslog-ng. You can write a boolean expression using internal functions, which has to evaluate to true for the message to pass.

Filters have also a uniquely identifying name, so you can refer to filters in your log statements. Syntax for the filter statement:

```
filter <identifier> { expression; };
```

An expression may contain the operators "and", "or" and "not", and any of the functions listed below.

Example 2-2. A filter statement finding the messages containing the word deny coming from the host blurp

```
filter f_blurp_deny { host("blurp") and match("deny"); };
```

Table 2-3. Available filter functions in syslog-ng

Function	Description
facility()	Selects messages based on their facility code
level() or priority()	Selects messages based on their priority
program()	Tries to match a regular expression to the program name field of log messages
host()	Tries to match a regular expression to the hostname field of log messages
match()	Tries to match a regular expression to the message itself.
filter()	Call another filter rule and evaluate its value

For a complete description on the above functions, see the Reference chapter.

There's a special filter identifier "DEFAULT" which allows you to catch not-yet-handled messages. For example, consider the following configuration:

```
options { keep_hostname(yes); };

source src { unix-stream("proba2"); internal(); };

destination ftpd { file("ftplog"); };
destination named { file("namedlog"); };
destination daemon { file("daemonlog"); };
```

```
  filter f_ftpd { match("ftp"); };
  filter f_named { match("named"); };
  filter f_daemon { facility(daemon); };

  log { source(src); filter(f_ftpd); destination(ftpd); };
  log { source(src); filter(f_named); destination(named); };
  log { source(src); filter(f_daemon); filter(DEFAULT); destina-
tion(daemon); };
```

The default filter above catches all facility=daemon messages which are not caught by the filter f_ftpd and f_named.

2.3. Destinations

A destination is a message sink, where log is sent if filtering rules match. Similarly to sources, destinations may include several drivers which define how messages are dispatched. To declare a destination in the configuration file, you'll need a destination statement, whose syntax is as following:

```
  destination <identifier> { destination-
driver(params); destination-driver(params); ... };
```

Table 2-4. Available destination drivers in syslog-ng

Name	Description>
file	Writes messages to the given file
fifo, pipe	Writes messages to the given named pipe
unix-stream	Sends messages to the given unix socket in SOCK_STREAM style (Linux)

Name	**Description>**
unix-dgram	Sends messages to the given unix socket in SOCK_DGRAM style (BSD)
udp	Sends messages to specified host and UDP port
tcp	Sends messages to specified host and TCP port
usertty	Sends messages to specified user if logged in
program	Forks and launches given program, and sends messages to its standard input.

For detailed list of the supported drivers, see the Reference chapter.

2.4. Log paths

In the previous chapters we learnt how to define sources, filters and destinations. We'll need to connect those together, which is accomplished by the log statement. Any message coming from one of the listed sources, matching the filters (each of them) are sent to the listed destinations. The needed syntax is here:

```
log { source(s1); source(s2); ...
filter(f1); filter(f2); ...
destination(d1); destination(d2); ... };
```

Members on the logpath are evaluated in order, e.g. the only filter invokations applied to a source are those which are after the source reference.

2.5. Options

There are several options you can specify, which modifies the behaviour of syslog-ng. For an exact list of possible options see the chapter Reference. The general syntax is here:

```
options { option1(params); option2(params); ... };
```

Each option may have parameters, just like in driver specification.

Table 2-5. List of supported global options in syslog-ng

Name	Accepted values	Description
time_reopen()	number	The time to wait before a died connection is reestablished
time_reap()	number	The time to wait before an idle destination file is closed.
sync_freq()	number	The number of lines buffered before written to file
mark_freq()	number	The number of seconds between two MARK lines. NOTE: not implemented yet.
log_fifo_size()	number	The number of lines fitting to the output queue
chain_hostnames()	yes or no	Enable or disable the chained hostname format.
use_time_recvd()	yes or no	Use the time a message is received instead of the one specified in the message.

Name	**Accepted values**	**Description**
use_dns()	yes or no	Enable or disable DNS usage. syslog-ng blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all hosts, which may get to syslog-ng is resolvable.
use_fqdn()	yes or no	Add Fully Qualified Domain Name instead of short hostname.
gc_idle_threshold()	number	Sets the threshold value for the garbage collector, when syslog-ng is idle. GC phase starts when the number of allocated objects reach this number. Default: 100.
gc_busy_threshold()	number	Sets the threshold value for the garbage collector, when syslog-ng is busy. GC phase starts when the number of allocated objects reach this number. Default: 3000.

Chapter 3. Reference

This chapter documents the drivers and options you may specify in the configuration file.

3.1. Source drivers

The following drivers may be used in the source statement, as described in the previous chapter.

3.1.1. internal()

All internally generated messages "come" from this special source. If you want warnings, errors and notices from syslog-ng itself, you have to include this source in one of your source statement.

```
Declaration: internal()
```

Syslog-ng will print you a warning, if this driver is not referenced.

Example 3-1. Using the internal() driver

```
source s_local { internal(); };
```

3.1.2. unix-stream() and unix-dgram()

This two drivers behave similarly: they open the given AF_UNIX socket, and start listening on them for messages. unix-stream() is primarily used on Linux, and uses SOCK_STREAM semantics (connection oriented, no messages are lost), unix-dgram()

is used on BSDs, and uses SOCK_DGRAM semantics, this may result in lost local messages, if the system is overloaded.

To avoid denial of service attacks when using connection-oriented protocols, the number of simoultaneously accepted connections should be limited. This can be achieved using the max-connections() parameter.

```
Declaration:
  unix-stream(filename [options]);
  unix-dgram(filename [options]);
```

The following options can be specified:

Table 3-1. Available options for unix-stream & unix-dgram

Name	Type	Description	Default
owner()	string	Set the uid of the socket.	root
group()	string	Set the gid of the socket. Default: root.	root
perm()	number	Set the permission mask. For octal numbers prefix the number with ’0’, e.g. use 0755 for rwxr-xr-x.	0666
keep-alive()	yes or no	Selects whether to keep connections opened when syslog-ng is restarted, can be used only with unix-stream(). Default: yes.	yes

Name	Type	Description	Default
max-connections()	number	Limits the number of simoultaneously opened connections. Can be used only with unix-stream().	10

Example 3-2. Using the unix-stream() and unix-dgram() drivers

```
    source s_stream { unix-stream("/dev/log" max-
connections(10)); };
    source s_dgram { unix-dgram("/var/run/log"); };
```

3.1.3. tcp() and udp()

These drivers let you receive messages from the network, and as the name of the drivers show, you can use both UDP and TCP.

UDP is a simple datagram protocol, which provides "best possible service" to transfer messages between hosts. It may lose messages, and no attempt is made to retransmit such lost messages at the protocol level.

TCP provides connection-oriented service, which basically means flow-controlled message pipeline. In this pipeline, each message is acknowledged, and retransmission is done for each lost packet. Generally it’s safer to use TCP, because lost connections can be detected, and no messages get lost, but traditionally syslogd protocol uses UDP.

None of tcp() and udp() drivers require positional parameters. By default they bind to 0.0.0.0:514, which means that syslog-ng will listen on all available interfaces. To limit accepted connections to one interface only, use the localip() parameter as described below.

NOTE: the tcp port 514 is reserved for use with rshell, so you have to pick another port if you intend to use syslog-ng and rshell at the same time.

```
  Declaration:
    tcp([options]);
    udp([options]);
```

The following options are valid for udp() and tcp()

Table 3-2. Available options for unix-stream & unix-dgram

Name	Type	Description	Default
ip or localip	string	The IP address to bind to.	0.0.0.0

Example 3-3. Using the udp() and tcp() drivers

```
    source s_tcp { tcp(ip(127.0.0.1) port(1999); max-
connections(10)); };
    source s_udp { udp(); };
```

3.1.4. file()

Usually the kernel presents its messages in a special file (/dev/kmsg on BSDs, /proc/kmsg on Linux), so to read such special files, you'll need the file() driver. Please note that you can't use this driver to follow a file like tail -f does. To feed a growing logfile into syslog-ng (HTTP access.log for instance), use a script like this:

Example 3-4. example script to feed a growing logfile into syslog-ng

```
    #!/bin/sh
    tail -f | logger -p local4.info
```

NOTE: on Linux, the klogd daemon reads kernel messages, and forwards them to the syslogd process. klogd preprocesses kernel messages and replaces addresses with symbolic names (from /boot/System.map). If you don't want to lose this functionality you'll have to run klogd with syslog-ng as well.

```
Declaration:
  file(filename);
```

Example 3-5. Using the file() driver

```
source s_file { file("/proc/kmsg"); };
```

3.1.5. pipe()

The pipe driver opens a named pipe with the specified name, and listens for messages. It's used as the native message getting protocol on HP-UX.

```
Declaration:
  pipe(filename);
```

NOTE: you'll need to create this pipe using mkfifo(1).

Example 3-6. Using the pipe() driver

```
source s_pipe { pipe("/dev/log"); };
```

3.1.6. sun-streams() driver

Solaris uses its STREAMS API to send messages to the syslogd process. You'll have to compile syslog-ng with this driver compiled in (see ./configure --help).

Newer versions of Solaris (2.5.1 and above), in addition to STREAMS uses a new IPC called door to confirm delivery of a message. Syslog-ng supports this new IPC mechanism with the door() option (see below).

3.2. Destination drivers

Destination drivers output log messages to somewhere outside syslog-ng: a file or a network socket.

3.2.1. file()

The file driver is one of the most important destination drivers in syslog-ng. It allows you to output logmessages to the named file, or as you'll see to a set of files.

The destination filename may include macros which gets expanded when the message is written, thus a simple file() driver may result in several files to be created. Macros can be included by prefixing the macro name with a '$' sign (without the quotes), just like in Perl/PHP.

If the expanded filename refers to a directory which doesn't exist, it'll be created depending on the create_dirs() setting (both global and a per destination option)

Warning: since the state of each created file must be tracked by syslog-ng, it consumes some memory for each file. If no new messages are written to a file within 60 seconds (controlled by the time_reap global option), it's closed, and its state is freed.

Exploiting this a DoS attack can be mounted against your system. If the number of possible destination files and its needed memory is more than the amount your logserver has.

The most suspicious macro is $PROGRAM, where the possible variations is quite high, so in untrusted environments $PROGRAM usage should be avoided.

Table 3-3. Available macros in filename expansion

Name	Description
HOST	The name of the source host where the message is originated from. If the message traverses several hosts, and chain_hostnames() is on, the first one is used.
FACILITY	The name of the facility, the message is tagged as coming from.
PRIORITY or LEVEL	The priority of the message.
PROGRAM	The name of the program the message was sent by.
YEAR	The year the message was sent. Time expansion macros can either use the time specified in the log message, e.g. the time the log message is sent, or the time the message was received by the log server. This is controlled by the use_time_recvd() option.
MONTH	The month the message was sent.
DAY	The day of month the message was sent.
HOUR	The hour of day the message was sent.
MIN	The minute the message was sent.
SEC	The second the message was sent.

Table 3-4. Available options for file()

Name	Type	Description	Default

Name	Type	Description	Default
log_fifo_size()	number	The number of entries in the output fifo.	Use global setting.
sync_freq()	number	The logfile is synced when this number of messages has been written to it.	Use global setting.
encrypt()	yes or no	Encrypt the resulting file. NOTE: this is not implemented as of 1.3.14.	Use global setting.
compress()	yes or no	Compress the resulting logfile using zlib. NOTE: this is not implemented as of 1.3.14.	Use global setting.
owner()	string	Set the owner of the created filename to the one specified.	root
group()	string	Set the group of the created filename to the one specified.	root
perm()	number	The permission mask of the file if it is created by syslog-ng.	0600

Name	Type	Description	Default
dir_perm()	number	The permission mask of directories created by syslog-ng. Log directories are only created if a file after macro expansion refers to a non-existing directory, and dir creation is enabled using create_dirs().	0600
create_dirs()	yes or no	Enable creating non-existing directories.	no

3.2.2. pipe()

This driver sends messages to a named pipe like /dev/xconsole

3.2.3. unix-stream() & unix-dgram()

This driver sends messages to a unix socket in either SOCK_STREAM or SOCK_DGRAM mode.

3.2.4. udp() & tcp()

This driver sends messages to another host on the local intranet or internet using either UDP or TCP protocol.

3.2.5. usertty()

This driver writes messages to the terminal of a logged-in user.

3.2.6. program()

This driver fork()'s executes the given program with the given arguments and sends messages down to the stdin of the child.

3.3. Filter functions

The following functions may be used in the filter statement, as described in the previous chapter.

3.4. Options

The following options can be specified in the options statement, as described in the previous chapter.

Chapter 4. Performance tuning in syslog-ng

There are several settings available you can finetune the behaviour of syslog-ng. The defaults should be adequate for a single server or workstation installation, but for a central loghost receiving the logs from multiple computers it may not be enough.

4.1. Setting garbage collector parameters

Syslog-ng uses a garbage collector internally, and while the garbage collector is running it does not accept messages. This may cause problems if some non-connection oriented transport protocol is used, like unix-dgram() or udp(). There are two settings which control the garbage collection phase:

4.1.1. gc_idle_threshold()

With this option you can specify the idle threshold of the gc. If the number of allocated objects reach this number, and the system is idle (no message arrived within 100msec), a gc phase starts. Since the system is idle, presumably no messages will be lost if the gc is ran. Therefore this value should be low, but higher than the minimally allocated objects. The minimum number of objects allocated depends on your configuration, but you can get exact numbers by specifying the -v command line option.

4.1.2. gc_busy_threshold()

This threshold is used when syslog-ng is busy accepting messages (this means that within 100msec an I/O event occured), however to prevent syslog-ng eating all your memory, gc should be ran in these cases as well. Set this value high, so that your log bursts don't get interrupted by the gc.

4.2. Setting output queue size

Syslog-ng always reads its incoming log channels to prevent your running daemons from blocking. This may result in lost messages if the output queue is full. It’s therefore important to set the output queue size (termed in number of messages), which you can do globally, or on a per destination basis.

```
  options { log_fifo_size(1000); };
```

or

```
  destina-
tion d_messages { file("/var/log/messages" log_fifo_size(1000); };
```

You should set your fifo size to the estimated number of messages in a message burst. If bursts extend the bandwidth of your destination pipe, syslog-ng can feed messages into the destination pipe after the burst has collapsed.

Of course syslog-ng cannot widen your network bandwidth, so if your destination host lives on a noisy network, and your logtraffic extends the bandwidth of this network, syslog-ng can’t do anything. It’ll do its best however.

4.3. Setting sync parameter

The sync parameter doesn’t exactly do what you might expect. As you have seen messages to be sent are buffered in an output queue. The sync parameter specifies the number of messages held in this buffer before anything is written.

Note that it doesn’t write all buffered messages in one single chunk, it writes each distinct message with a single write() system call.