syslog-ng reference manual

Balázs Scheidler

syslog-ng reference manual
by Balázs Scheidler

Table of Contents

1. Introduction to syslog-ng ... 1

2. Message paths ... 2

2.1. Sources ... 2
2.2. Filters ... 3
2.3. Destinations ... 3
2.4. Log paths ... 4
2.5. Options ... 5

3. Reference ... 6

3.1. Source drivers ... 6
3.1.1. internal() ... 6
3.1.2. unix-stream() and unix-dgram() ... 6
3.1.3. tcp() and udp() ... 7
3.1.4. file() ... 8
3.1.5. pipe() ... 9
3.1.6. sun-streams() driver ... 9
3.2. Destination drivers ... 10
3.2.1. file() ... 10
3.2.2. pipe() ... 13
3.2.3. unix-stream() & unix-dgram() ... 14
3.2.4. udp() & tcp() ... 15
3.2.5. usertty() ... 17
3.2.6. program() ... 17
3.3. Filter functions ... 18
3.4. Options ... 18

4. Performance tuning in syslog-ng ... 21

4.1. Setting garbage collector parameters ... 21
4.1.1. gc_idle_threshold() ... 21
4.1.2. gc_busy_threshold() ... 21
4.2. Setting output queue size ... 21
4.3. Setting sync parameter ... 21

List of Tables

2-1. Communication method between syslogd and its clients 2
2-2. Available source drivers in syslog-ng 2
2-3. Available destination drivers in syslog-ng .... **??**
2-4. Log statement flags 4
3-1. Available options for unix-stream & unix-dgram 6
3-2. Available options for udp() & tcp() 7
3-3. Available options for file 8
3-4. Available options for pipe 9
3-5. Available options for sun-streams 10
3-6. Available macros in filename expansion 10
3-7. Available options for file() 11
3-8. Available options for pipe() 13
3-9. Available options for unix-stream() & unix-dgram() 14
3-10. Available options for udp() & tcp() 15
3-11. Additional options for tcp() 16
3-12. Available options for usertty() 17
3-13. Available options for program() 18
3-14. Available filter functions in syslog-ng 18
3-15. List of supported global options in syslog-ng 19

List of Examples

2-1. Source statement on a Linux based operating system 2
2-2. A filter statement finding the messages containing the word deny coming from the host blurp 3
3-1. Using the internal() driver 6
3-2. Using the unix-stream() and unix-dgram() drivers 7
3-3. Using the udp() and tcp() drivers 8
3-4. example script to feed a growing logfile into syslog-ng 8
3-5. Using the file() driver 9
3-6. Using the pipe() driver 9
3-7. Using the sun-streams() driver 10
3-8. Using the file() driver 13
3-9. Using the file() driver with macros in the file name and a template for the message 13
3-10. Using the pipe() driver 14
3-11. Using the unix-stream() driver 15
3-12. Using the tcp() driver 16
3-13. Using the usertty() driver 17
3-14. Using the program() destination driver 18

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. A message is entered to syslog-ng in one of its sources, if that message matches the filtering rules it goes out using the destinations. Note that a message goes to _all_ matching destinations by default, although this behaviour can be changed.

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
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 are required, others are optional. The required parameters are positional, meaning that they must be specified in a defined order. A unix-stream() driver has a single required argument, the name of the socket to listen to, and several optional parameters, which follow the socket name. Optional arguments can be specified in any order and must have the form `option(value)`.

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

Name	Description
internal()	Messages generated internally in syslog-ng
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 Chapter 3

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 parentheses, the boolean operators "and", "or" and "not", and any of the functions listen in Table 3-14.

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"); };
```

For a complete description on the above functions, see Chapter 3.

In earlier revisions of syslog-ng there was a special filter identifier, "DEFAULT", which matched all not-yet-matched messages. This could make your configuration much simpler and easier to manage. This feature was removed in syslog-ng 1.5.x, and a more powerful idea was introduced. For more details consult Section 2.4.

2.3. Destinations

A destination is where log is sent if filtering rules match. Similarly to sources, destinations are comprised of one or more drivers, each of which define how messages are handled. 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 2-3. 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)
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 Chapter 3.

2.4. Log paths

In the previous chapters we learnt how to define sources, filters and destinations. We'll need to connect those components together, which is accomplished by the log statement. The needed syntax is here:

```
log { source(s1); source(s2); ...
      filter(f1); filter(f2); ...
      destination(d1); destination(d2); ...
      flags(flag1[, flag2...]); };
```

Any message coming from any of the listed sources, matching the all the filters are sent to all listed destinations. Log statements are processed in the order they appear in the config file.

By default all matching log statements are processed, therefore a single log message might be sent to the same destination several times, given that destination is listed on several log statements.

This default behaviour can be changed by the flags() parameter.

Table 2-4. Log statement flags

Flag	Description
final	This flag means that the processing of log statements ends here. Note that this doesn't necessarily mean that matching messages will be stored once, as they can be matching log statements processed prior the current one.
fallback	This flag makes a log statement 'fallback'. Being a fallback statement means that only messages not matching any 'non-fallback' log statements will be dispatched.

Flag	**Description**
catchall	This flag means that the source of the message is ignored, only the filters are taken into account when matching messages.

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 3. The general syntax is here:

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

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

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 statements.

```
Declaration: internal()
```

Syslog-ng will issue a warning upon startup, 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()

These 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 simultaneously accepted connections should be limited. This can be achieved using the max-connections() parameter. The default value of this parameter is quite strict, you might have to increase it on a busy system.

Both unix-stream and unix-dgram has a single required positional argument, specifying the filename of the socket to create, and several optional parameters.

```
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

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

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

```
# source declaration on Linux
source s_stream { unix-stream("/dev/log" max-connections(10)); };

# source declaration on BSDs
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 as transport.

UDP is a simple datagram oriented protocol, which provides "best effort 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 a flow-controlled message pipeline. In this pipeline, each message is acknowledged, and retransmission is done for lost packets. Generally it’s safer to use TCP, because lost connections can be detected, and no messages get lost, but traditionally the syslog 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, port 514. 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 udp() & tcp()

Name	Type	Description	Default
ip() or localip()	string	The IP address to bind to. Note that this is not the address where messages are accepted from.	0.0.0.0
port() or localport()	number	The port number to bind to.	514
keep-alive()	yes or no	Available for tcp() only, and specifies whether to close connections upon the receival of a SIGHUP signal.	yes
tcp-keep-alive()	yes or no	Available for tcp() only, and specifies whether to enable TCP keep alive messages using the SO_KEEPALIVE socket option.	no
max-connections()	number	Specifies the maximum number of simultaneous connections.	10

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 logfile | logger -p local4.info
```

The file driver has a single required parameter specifying the file to open and the following options:

Table 3-3. Available options for file

Name	Type	Description	Default

Name	Type	Description	Default
log_prefix()	string	The string to prepend log messages. Useful for logging kernel messages as it is not prefixed by ’kernel:’ by default	empty string

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

Example 3-5. Using the file() driver

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

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.

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.

The pipe driver has a single required parameter, specifying the filename of the pipe to open, and the following options:

Table 3-4. Available options for pipe

Name	Type	Description	Default
pad_size()	number	Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes)	0

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

The sun-streams() driver has a single required argument, specifying the STREAMS device to open and a single option.

Example 3-7. Using the sun-streams() driver

```
source s_stream { sun-streams("/dev/log" door("/etc/.syslog_door"); };
```

Table 3-5. Available options for sun-streams

Name	Type	Description	Default
door()	string	Specifies the filename of a door to open, needed on Solaris above 2.5.1.	none

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 messages 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 will 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 3-6. Available macros in filename expansion

Name	Description
FACILITY	The name of the facility, the message is tagged as coming from.
PRIORITY or LEVEL	The priority of the message.
TAG	The priority and facility encoded as a 2 digit hexadecimal number.
DATE	
FULLDATE	
ISODATE	
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.
WEEKDAY	The 3-letter name of the day of week the message was sent, e.g. ’Thu’.
HOUR	The hour of day the message was sent.
MIN	The minute the message was sent.
SEC	The second the message was sent.
TZOFFSET	The time-zone as hour offset from GMT. e.g. ’-0700’
TZ	The time zone or name or abbreviation. e.g. ’PDT’
FULLHOST	
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.
PROGRAM	The name of the program the message was sent by.
MSG or MESSAGE	Message contents including the programname and pid.
MSGONLY	Message contents without the program name.

Table 3-7. Available options for file()

Name	Type	Description	Default
log_fifo_size()	number	The number of entries in the output fifo.	Use global setting.
fsync()	yes or no	Forces an fsync() call on the destination fd after each write. Note: this may degrade performance seriously	
sync_freq()	number	The logfile is synced when this number of messages has been written to it.	Use global setting.

Name	Type	Description	Default
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
create_dirs()	yes or no	Enable creating non-existing directories.	no
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
dir_owner()	string	The owner of directories created by syslog-ng.	root
dir_group()	string	The group of directories created by syslog-ng.	root
template()	string	Specifies a template which defines the logformat to be used in this file. Possible macros are the same as with destination file().	a format conforming to the default logfile format.
template_escape()	yes or no	Turns on escaping ’ and " in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of your log message don’t get interpreted as commands to the SQL server.	yes

Name	Type	Description	Default
remove_if_older()	number	If set to a value higher than 0, before writing to a file, syslog-ng checks whether this file is older than the specified amount of time (specified in seconds). If so, it removes the existing file and the line to be written is the first line in a new file with the same name. In combination with e.g. the $WEEKDAY macro, this is can be used for simple log rotation, in case not all history need to be kept.	Do never remove existing files, but append (= 0).

Example 3-8. Using the file() driver

```
destination d_file { file("/var/log/messages" ); };
```

Example 3-9. Using the file() driver with macros in the file name and a template for the message

```
destination d_file {
        file("/var/log/$YEAR.$MONTH.$DAY/messages"
                template("$HOUR:$MIN:$SEC $TZ $HOST [$LEVEL] $MSG $MSG\n")
                template_escape(no)
        );
};
```

3.2.2. pipe()

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

The pipe driver has a single required parameter, specifying the filename of the pipe to open.

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

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

Table 3-8. Available options for pipe()

Name	Type	Description	Default
owner()	string	Set the owner of the pipe to the one specified.	root
group()	string	Set the group of the pipe to the one specified.	root

Name	Type	Description	Default
perm()	number	The permission mask of the pipe.	0600
template()	string	Specifies a template which defines the logformat to be used. Possible macros are the same as with destination file().	a format conforming to the default logfile format.
template_escape()	yes or no	Turns on escaping ' and " in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of your log message don't get interpreted as commands to the SQL server.	yes

Example 3-10. Using the pipe() driver

```
destination d_pipe { pipe("/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.

Both drivers have a single required argument specifying the name of the socket to connect to.

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

Table 3-9. Available options for unix-stream() & unix-dgram()

Name	Type	Description	Default
template()	string	Specifies a template which defines the logformat to be used. Possible macros are the same as with destination file().	a format conforming to the default logfile format.

Name	Type	Description	Default
template_escape()	yes or no	Turns on escaping ’ and " in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of your log message don’t get interpreted as commands to the SQL server.	yes

Example 3-11. Using the unix-stream() driver

```
    destination d_unix_stream { unix-stream("/var/run/logs"); };
```

3.2.4. udp() & tcp()

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

Both drivers have a single required argument specifying the destination host address, where messages should be sent, and several optional parameters. Note that this differs from source drivers, where local bind address is implied, and none of the parameters are required.

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

Table 3-10. Available options for udp() & tcp()

Name	Type	Description	Default
localip()	string	The IP address to bind to before connecting to target.	0.0.0.0
localport()	number	The port number to bind to.	0
port() or destport()	number	The port number to connect to.	514
template()	string	Specifies a template which defines the logformat to be used. Possible macros are the same as with destination file().	a format conforming to the default logfile format.

Name	Type	Description	Default
template_escape()	yes or no	Turns on escaping ’ and " in templated output. This is useful for generating SQL statements and quoting string contents so that parts of your log message don’t get interpreted as commands to the SQL server.	yes
tcp-keep-alive()	yes or no	Available for tcp() only, and specifies whether to enable TCP keep alive messages using the SO_KEEPALIVE socket option.	no
spoof_source	yes or no	Enables source address spoofing. This means that the host running syslog-ng generates UDP packets with the source IP address matching the original sender of the message. It is useful when you want to perform some kind of preprocessing via syslog-ng then forward messages to your central log management solution with the source address of the original sender. This option only works for UDP destinations though the original message can be received by TCP as well. This option is only available if syslog-ng was compiled using the --enable-spoof-source configure option.	no

Table 3-11. Additional options for tcp()

Name	Type	Description	Default
sync()	number	The messages are sent to the remote host when this number of messages have been collected.	0

Example 3-12. Using the tcp() driver

```
    destination d_tcp { tcp("10.1.2.3" port(1999); localport(999)); };
```

3.2.5. usertty()

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

The usertty driver has a single required argument, specifying a username who should receive a copy of matching messages, and no optional arguments.

```
  Declaration:
    usertty(username);
```

Table 3-12. Available options for usertty()

Name	Type	Description	Default
template()	string	Specifies a template which defines the logformat to be used. Possible macros are the same as with destination file().	a format conforming to the default logfile format.
template_escape()	yes or no	Turns on escaping ’ and " in templated output. This is useful for generating SQL statements and quoting string contents so that parts of your log message don’t get interpreted as commands to the SQL server.	yes

Example 3-13. Using the usertty() driver

```
    destination d_usertty { usertty("root"); };
```

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.

The program driver has a single required parameter, specifying a program name to start and no options. The program is executed with the help of the current shell, so the command may include both file patterns and I/O redirection, they will be processed.

```
  Declaration:
    program(commandtorun);
```

NOTE: the program is executed once at startup, and kept running until SIGHUP or exit. The reason is to prevent starting up a large number of programs for messages, which would imply an easy DoS.

Table 3-13. Available options for program()

Name	Type	Description	Default
template()	string	Specifies a template which defines the logformat to be used. Possible macros are the same as with destination file().	a format conforming to the default logfile format.
template_escape()	yes or no	Turns on escaping ’ and " in templated output. This is useful for generating SQL statements and quoting string contents so that parts of your log message don’t get interpreted as commands to the SQL server.	yes

Example 3-14. Using the program() destination driver

```
destination d_prg { program("/bin/cat >/dev/null"); };
```

3.3. Filter functions

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

Table 3-14. Available filter functions in syslog-ng

Name	Synopsis	Description
facility	facility(faciliy[,facility])	Match messages h
level() or priority()	level(pri[,pri1..pri2[,pri3]])	Match messages b
program()	program(regexp)	Match messages b
host()	host(regexp)	Match messages b
match()	match(regexp)	Tries to match a re
filter()	filter(filtername)	Call another filter
netmask()	netmask(ip/mask)	Check the sender’s

3.4. Options

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

Table 3-15. 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()	number	The number of lines buffered before written to file
mark()	number	The number of seconds between two MARK lines. NOTE: not implemented yet.
stats()	number	The number of seconds between two STATS.
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.
keep_hostname()	yes or no	Enable or disable hostname rewriting.
check_hostname()	yes or no	Enable or disable whether the hostname contains valid characters.
bad_hostname()	regular expression	A regexp which matches hostnames which should not be taken as such.
create_dirs()	yes or no	Enable or disable directory creation for destination files.
owner()	userid	.
group()	groupid	.
perm()	permission value	.
dir_owner()	userid	.
dir_group()	groupid	.
dir_perm()	permission value	.
use_time_recvd()	yes or no	Use the time a message is received instead of the one specified in the message.
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.

Name	**Accepted values**	**Description**
dns_cache()	yes or no	Enable or disable DNS cache usage.
dns_cache_size()	number	Number of hostnames in the DNS cache.
dns_cache_expire()	number	Number of seconds while a successful lookup is cached.
dns_cache_expire_failed()	number	Number of seconds while a failed lookup is cached.
log_msg_size()	number	Maximum length of message in bytes.
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 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

```
destination 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.