The syslog-ng Open Source Edition 3.1 Administrator Guide

Fifth Edition

Publication date September 23, 2010

This manual is the primary documentation of the syslog-ng Open Source Edition 3.1 application.

The syslog-ng Open Source Edition 3.1 Administrator Guide

Product Marketing and Documentation Department

Revision History
Fifth Revision September 23, 2010
SDATA description corrections, corrected a syslog driver example, and other minor corrections.
Fourth Revision July 5, 2010
Numerous corrections and clarifications.
Third Revision April 30, 2010
Parameter list in of the Reference chapter have been changed to a list of sections; added a separate *index for parameters*, documented the missing *FACILITY_NUM and LEVEL_NUM macros*, and several other minor corrections and typos.
Second Revision March 25, 2010
More detailed documentation for message statistics. Minor corrections and new man pages.
First Revision January 8, 2010
Initial release

Table of Contents

List of Examples

List of Procedures

Preface

Welcome to the syslog-ng Open Source Edition 3.1 Administrator Guide!

This document describes how to configure and manage syslog-ng. Background information for the technology and concepts used by the product is also discussed.

1. Summary of contents

Chapter 1, Introduction to syslog-ng (p. 1) describes the main functionality and purpose of syslog-ng OSE.

Chapter 2, The concepts of syslog-ng (p. 4) discusses the technical concepts and philosophies behind syslog-ng OSE.

Chapter 3, Installing syslog-ng (p. 26) describes how to install syslog-ng OSE on various UNIX-based platforms using the precompiled binaries.

Chapter 4, Configuring syslog-ng (p. 42) provides detailed description on configuring and managing syslog-ng OSE as a client or a server.

Chapter 5, Best practices and examples (p. 86) gives recommendations to configure special features of syslog-ng.

Chapter 6, Reference (p. 91) is a reference guide of syslog-ng OSE, describing all available parameters and options.

Appendix 1, The syslog-ng manual pages (p. 179) contains the manual pages of the syslog-ng OSE application.

Appendix 2, GNU General Public License (p. 195) includes the text of the GPLv2 licence applicable to syslog-ng Open Source Edition.

Glossary (p. 211) provides definitions of important terms used in this guide.

Index (p. 218) provides cross-references to important terms used in this guide.

2. Target audience and prerequisites

This guide is intended for system administrators and consultants responsible for designing and maintaining logging solutions and log centers. It is also useful for IT decision makers looking for a tool to implement centralized logging in heterogeneous environments.

The following skills and knowledge are necessary for a successful syslog-ng administrator:

- At least basic system administration knowledge.
- An understanding of networks, TCP/IP protocols, and general network terminology.
- Working knowledge of the UNIX or Linux operating system.
- In-depth knowledge of the logging process of various platforms and applications.
- An understanding of the legacy syslog (BSD-syslog) protocol (see RFC 3164, available at *http://www.ietf.org/rfc/rfc3164.txt*) and the new syslog (IETF-syslog) protocol standard (see RFC 5424-5428, available at *http://tools.ietf.org/html/rfc5424*).

3. Products covered in this guide

This guide describes the use of the following syslog-ng version:

- syslog-ng Open Source Edition (OSE) 3.1.0 and later

4. Typographical conventions

Before you start using this guide, it is important to understand the terms and typographical conventions used in the documentation. For more information on specialized terms and abbreviations used in the documentation, see the *Glossary* at the end of this document.

The following kinds of text formatting and icons identify special information in the document.

Tip
Tips provide best practices and recommendations.

Note
Notes provide additional information on a topic and emphasize important facts and considerations.

Warning
Warnings mark situations where loss of data or misconfiguration of the device is possible if the instructions are not obeyed.

`Command`	Commands you have to execute.
Emphasis	Reference items, additional readings.
`/path/to/file`	File names.
`Parameters`	Parameter and attribute names.
Label	GUI output messages or dialog labels.
Menu	A submenu in the menu bar.
Button	Buttons in dialog windows.

5. Contact and support information

The syslog-ng Open Source Edition application is developed and maintained by BalaBit IT Security Ltd. We are located in Budapest, Hungary. Our address is:

BalaBit IT Security Ltd.
1464 Budapest P.O. BOX 1279

Hungary
Tel: +36 1 371-0540
Fax: +36 1 208-0875
E-mail: info@balabit.com
Web: *http://www.balabit.com/*

5.1. Sales contact

You can directly contact us with sales related topics at the e-mail address <sales@balabit.com>.

5.2. Support contact

To subscribe to the mailing list of the syslog-ng community, visit *https://lists.balabit.hu/mailman/listinfo/syslog-ng/*.

To report bugs found in syslog-ng, visit *https://bugzilla.balabit.com/*.

Product support, including 7x24 online support is available in various packages. For support options, visit the following page: *http://www.balabit.com/support/packages/*

Precompiled binary packages are available for free for the supported Linux and BSD platforms at *http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/upgrades/*. See the following link for the list of supported platforms: *http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/support/*

Support e-mail address: <support@balabit.com>.

Support hotline: +36 1 371 0540 (available from 9 AM to 5 PM CET on weekdays)

The BalaBit Online Support System is available at *https://boss.balabit.com/* and offers 24 hours technical support. This system is available only for users with a valid support contract and a MyBalaBit account. To sign up for MyBalaBit, visit the following page: *http://www.balabit.com/mybalabit*.

5.3. Training

BalaBit IT Security Ltd. holds courses for advanced GNU/Linux system administrators. Our experienced system engineers give lectures on syslog-ng administration.

6. About this document

This guide is a work-in-progress document with new versions appearing periodically.

The latest version of this document can be downloaded from the BalaBit website at *http://www.balabit.com/support/documentation/*.

For news and update notifications about the syslog-ng documentation, visit the BalaBit Documentation Blog at *http://robert.blogs.balabit.com*.

6.1. What is new in this main edition of The syslog-ng Administrator Guide?

The syslog-ng Open Source Edition 3.1 Administrator Guide contains the following main changes compared to earlier editions:

- The contents of the guide have been updated for syslog-ng Open Source Edition 3.1.
- Earlier editions of the The syslog-ng Administrator Guide covered both the open source and the commercial versions of syslog-ng. Starting with The syslog-ng 3.1 Administrator Guide, they are discussed in separate documents called The syslog-ng Open Source Edition Administrator Guide, The syslog-ng Premium Edition Administrator Guide, and The syslog-ng Agent for Windows Administrator Guide.
- The order of chapters has changed: installation and compiling information is now before the configuration and reference chapters.
- Message statistics are described in more detail.
- Manual pages for the `loggen`, `pdbtool`, and `syslog-ng-ctl` utilities are now included.

6.2. Feedback

Any feedback is greatly appreciated. General comments, errors found in the text, and any suggestions about how to improve the documentation is welcome at <documentation@balabit.com>.

6.3. Acknowledgments

BalaBit would like to express its gratitude to the syslog-ng users and the syslog-ng community for their invaluable help and support.

Special thanks to Nate Campi for organizing and hosting the syslog-ng FAQ (*http://campin.net/syslog-ng/faq.html*) and for his permission to reproduce parts of his work in this guide.

Chapter 1. Introduction to syslog-ng

This chapter introduces the syslog-ng Open Source Edition application in a non-technical manner, discussing how and why is it useful, and the benefits it offers to an existing IT infrastructure.

1.1. What syslog-ng is

The syslog-ng application is a flexible and highly scalable system logging application that is ideal for creating centralized and trusted logging solutions. The main features of syslog-ng are summarized below.

- *Reliable log transfer*: The syslog-ng application enables you to send the log messages of your hosts to remote servers using the latest protocol standards. The logs of different servers can be collected and stored centrally on dedicated log servers. Transferring log messages using the TCP protocol ensures that no messages are lost.
- *Secure logging using TLS*: Log messages may contain sensitive information that should not be accessed by third parties. Therefore, syslog-ng uses the Transport Layer Security (TLS) protocol to encrypt the communication. TLS also allows the mutual authentication of the host and the server using X.509 certificates.
- *Direct database access*: Storing your log messages in a database allows you to easily search and query the messages and interoperate with log analyzing applications. The syslog-ng application supports the following databases: MSSQL, MySQL, Oracle, PostgreSQL, and SQLite.
- *Heterogeneous environments*: The syslog-ng application is the ideal choice to collect logs in massively heterogeneous environments using several different operating systems and hardware platforms, including Linux, Unix, BSD, Sun Solaris, HP-UX, Tru64, and AIX.
- *Filter and classify*: The syslog-ng application can sort the incoming log messages based on their content and various parameters like the source host, application, and priority. Directories, files, and database tables can be created dynamically using macros. Complex filtering using regular expressions and boolean operators offers almost unlimited flexibility to forward only the important log messages to the selected destinations.
- *Parse and rewrite*: The syslog-ng application can segment log messages to named fields or columns, and also modify the values of these fields.
- *IPv4 and IPv6 support*: The syslog-ng application can operate in both IPv4 and IPv6 network environments; it can receive and send messages to both types of networks.

1.2. What syslog-ng is not

The syslog-ng application is not log analysis software. It can filter log messages and select only the ones matching certain criteria. It can even convert the messages and restructure them to a predefined format, or parse the messages and segment them into different fields. But syslog-ng cannot interpret and analyze the meaning behind the messages, or recognize patterns in the occurrence of different messages.

1.3. Why is syslog-ng needed?

Log messages contain information about the events happening on the hosts. Monitoring system events is essential for security and system health monitoring reasons.

The original syslog protocol separates messages based on the priority of the message and the facility sending the message. These two parameters alone are often inadequate to consistently classify messages, as many applications might use the same facility — and the facility itself is not even included in the log message. To make things worse, many log messages contain unimportant information. The syslog-ng application helps you to select only the really interesting messages, and forward them to a central server.

Company policies or other regulations often require log messages to be archived. Storing the important messages in a central location greatly simplifies this process.

For details on how can you use syslog-ng to comply with various regulations, see the *Regulatory compliance and system logging* whitepaper available at *http://www.balabit.com/support/documentation/*

1.4. What is new in syslog-ng Open Source Edition 3.1?

Version 3.1 of syslog-ng Open Source Edition includes the following main features:

- syslog-ng Open Source Edition 3.1 uses a new pattern database format dubbed V3 that has several improvements over the older V1 format.

1.5. Who uses syslog-ng?

The syslog-ng application is used worldwide by companies and institutions who collect and manage the logs of several hosts, and want to store them in a centralized, organized way. Using syslog-ng is particularly advantageous for:

- Internet Service Providers;
- Financial institutions and companies requiring policy compliance;
- Server, web, and application hosting companies;
- Datacenters;
- Wide area network (WAN) operators;
- Server farm administrators.

The following is a list of public references — companies who use syslog-ng in their production environment:

- Allianz Hungary Insurance Co. (*http://www.allianz.hu/*)
- Navisite Inc. (*http://www.navisite.com/*)
- Svenska Handelsbanken AB (*http://www.handelsbanken.com/*)
- Swedish National Debt Office (*http://www.riksgalden.se*)

1.6. Supported platforms

The syslog-ng Open Source Edition application is highly portable and is known to run on a wide range of hardware architectures (x86, x86_64, SUN Sparc, PowerPC 32 and 64, Alpha) and operating systems, including Linux, BSD, Solaris, IBM AIX, HP-UX, Mac OS X, Cygwin, Tru64, and others.

- The source code of syslog-ng Open Source Edition is released under the GPLv2 license and is available at *http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/upgrades/#any*
- Precompiled binary packages provided by BalaBit are available for free for the supported Linux and BSD platforms at *http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/upgrades/*.
- For syslog-ng Open Source Edition packages for Solaris 8-10, visit *http://www.sunfreeware.com/*
- For syslog-ng Open Source Edition packages for IBM AIX 5 and later, visit *http://www.perzl.org/aix/index.php?n=Main.Syslog-ng*
- For syslog-ng Open Source Edition packages for HP-UX, visit *http://hpux.connect.org.uk/hppd/cgi-bin/search?package=one&description=one&term=syslog-ng&Search=Search*
- For syslog-ng Open Source Edition packages for Mac OS X, visit *http://syslog-ng.darwinports.com/*
- Packages for routers running OpenWRT or a similar embedded Linux distribution are available at *http://www.openwrt.org/*

Chapter 2. The concepts of syslog-ng

This chapter discusses the technical concepts of syslog-ng.

2.1. The philosophy of syslog-ng

Typically, syslog-ng is used to manage log messages and implement centralized logging, where the aim is to collect the log messages of several devices on a single, central log server. The different devices — called syslog-ng clients — all run syslog-ng, and collect the log messages from the various applications, files, and other *sources*. The clients send all important log messages to the remote syslog-ng server, where the server sorts and stores them.

2.2. Logging with syslog-ng

The syslog-ng application reads incoming messages and forwards them to the selected *destinations*. The syslog-ng application can receive messages from files, remote hosts, and other *sources*.

Log messages enter syslog-ng in one of the defined sources, and are sent to one or more *destinations*.

Sources and destinations are independent objects; *log paths* define what syslog-ng does with a message, connecting the sources to the destinations. A log path consists of one or more sources and one or more destinations; messages arriving to a source are sent to every destination listed in the log path. A log path defined in syslog-ng is called a *log statement*.

Optionally, log paths can include *filters*. Filters are rules that select only certain messages, for example, selecting only messages sent by a specific application. If a log path includes filters, syslog-ng sends only the messages satisfying the filter rules to the destinations set in the log path.

Other optional elements that can appear in log statements are *parsers* and *rewriting rules*. Parsers segment messages into different fields to help processing the messages, while rewrite rules modify the messages by adding, replacing, or removing parts of the messages.

The following procedure illustrates the route of a log message from its source on the syslog-ng client to its final destination on the central syslog-ng server.

2.2.1. Procedure – The route of a log message in syslog-ng

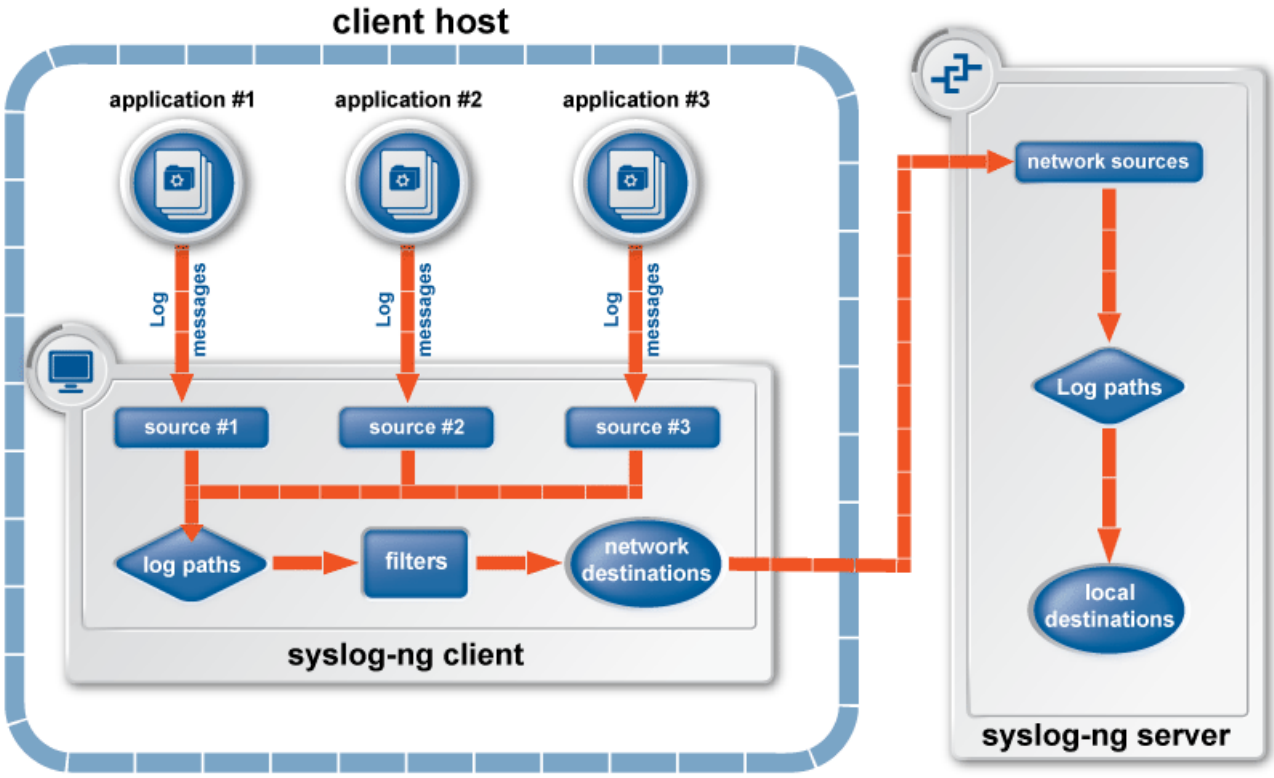

Figure 2.1. The route of a log message

Step 1. A device or application sends a log message to a source on the syslog-ng client. For example, an Apache web server running on Linux enters a message into the `/var/log/apache` file.

Step 2. The syslog-ng client running on the web server reads the message from its `/var/log/apache` source.

Step 3. The syslog-ng client processes the first log statement that includes the `/var/log/apache` source.

Step 4. The syslog-ng client performs optional operations (message filtering, parsing, and rewriting) on the message; for example, it compares the message to the filters of the log statement (if any). If the message complies with all filter rules, syslog-ng sends the message to the destinations set in the log statement, for example, to the remote syslog-ng server.

Warning
Message filtering, parsing, and rewriting is performed in the order that the operations appear in the log statement.

Note
The syslog-ng client sends a message to *all* matching destinations by default. As a result, a message may be sent to a destination more than once, if the destination is used in multiple log statements. To prevent such situations, use the `final` flag in the destination statements. See *Table 6.1, Log statement flags (p. 150)* for details.

Step 5. The syslog-ng client processes the next log statement that includes the `/var/log/apache` source, repeating Steps 3-4.

Step 6. The message sent by the syslog-ng client arrives to a source set in the syslog-ng server.

Step 7. The syslog-ng server reads the message from its source and processes the first log statement that includes that source.

Step 8. The syslog-ng server performs optional operations (message filtering, parsing, and rewriting) on the message; for example, it compares the message to the filters of the log statement (if any). If the message complies with all filter rules, syslog-ng sends the message to the destinations set in the log statement.

Warning
Message filtering, parsing, and rewriting is performed in the order that the operations appear in the log statement.

Step 9. The syslog-ng server processes the next log statement, repeating Steps 7-9.

Note
The syslog-ng application can stop reading messages from its sources if the destinations cannot process the sent messages. This feature is called flow-control and is detailed in *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16).*

2.2.2. Embedded log statements

Starting from version 3.0, syslog-ng can handle embedded log statements (also called log pipes). Embedded log statements are useful for creating complex, multi-level log paths with several destinations and use filters, parsers, and rewrite rules.

For example, if you want to filter your incoming messages based on the facility parameter, and then use further filters to send messages arriving from different hosts to different destinations, you would use embedded log statements.

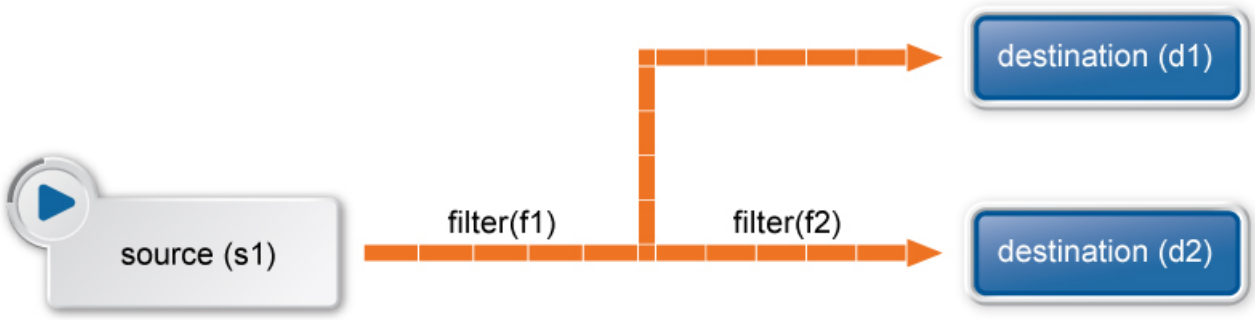

Figure 2.2. Embedded log statement

Embedded log statements include sources — and usually filters, parsers, rewrite rules, or destinations — and other log statements that can include filters, parsers, rewrite rules, and destinations. The following rules apply to embedded log statements:

- Only the beginning (also called top-level) log statement can include sources.
- Embedded log statements can include multiple log statements on the same level (i.e., a top-level log statement can include two or more log statements).
- Embedded log statements can include several levels of log statements (i.e., a top-level log statement can include a log statement that includes another log statement, and so on).
- Only another log statement can follow an embedded log statement, filters or other rules cannot.
- Embedded log statements that are on the same level receive the same messages from the higher-level log statement. For example, if the top-level log statement includes a filter, the lower-level log statements receive only the messages that pass the filter.

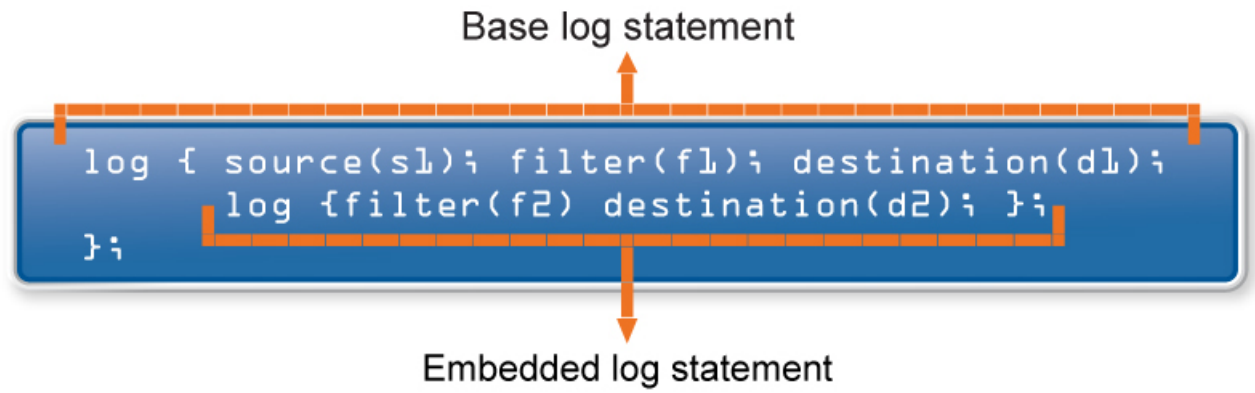

Figure 2.3. Embedded log statements

Embedded log filters can be used to optimize the processing of log messages, for example, to re-use the results of filtering and rewriting operations.

2.3. Modes of operation

The syslog-ng Open Source Edition application has three typical operation scenarios: *Client*, *Server*, and *Relay*.

2.3.1. Client mode

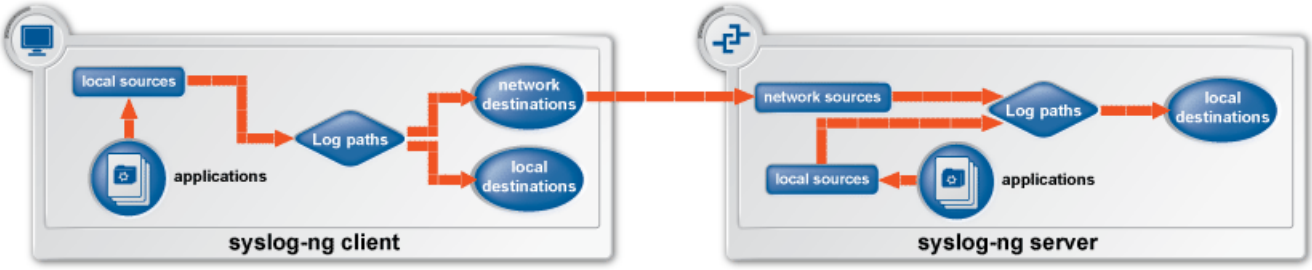

Figure 2.4. Client-mode operation

In client mode, syslog-ng collects the local logs generated by the host and forwards them through a network connection to the central syslog-ng server or to a relay. Clients often also log the messages locally into files.

2.3.2. Relay mode

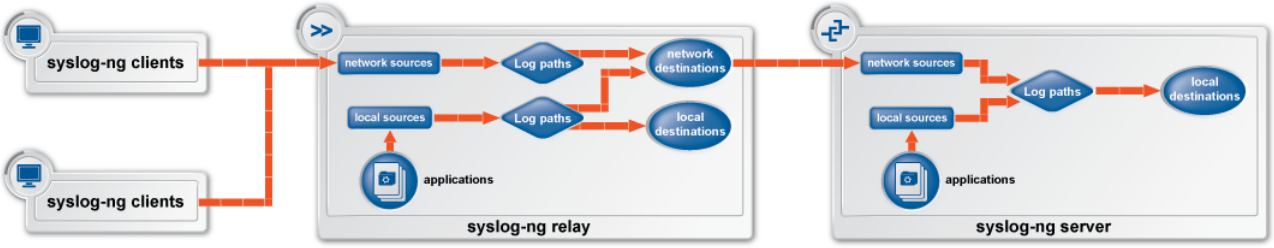

Figure 2.5. Relay-mode operation

In relay mode, syslog-ng receives logs through the network from syslog-ng clients and forwards them to the central syslog-ng server using a network connection. Relays also log the messages from the relay host into a local file, or forward these messages to the central syslog-ng server.

2.3.3. Server mode

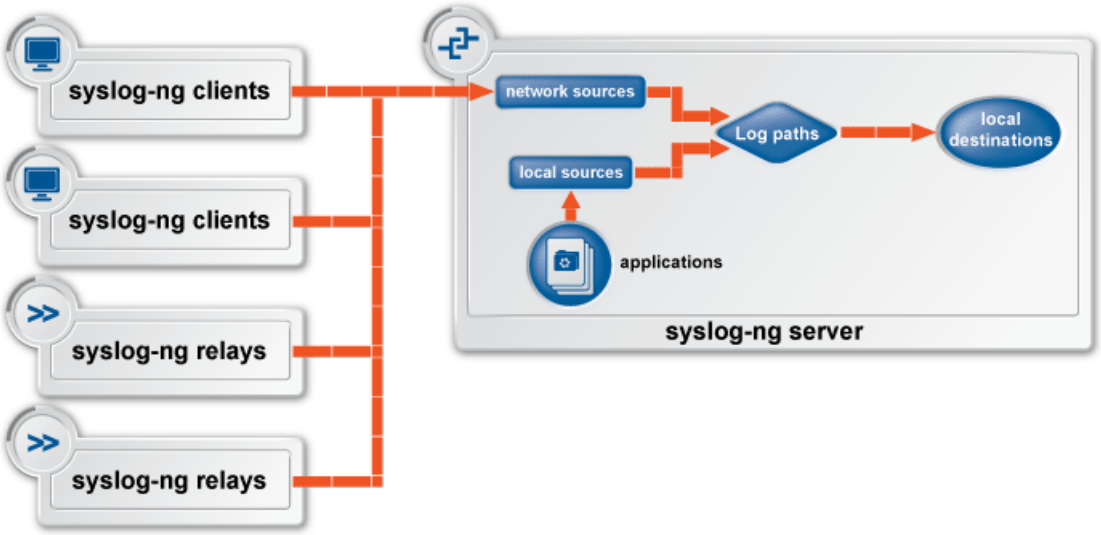

Figure 2.6. Server-mode operation

In server mode, syslog-ng acts as a central log-collecting server. It receives messages from syslog-ng clients and relays over the network, and stores them locally in files, or passes them to other applications, e.g., log analyzers.

2.4. Global objects

The syslog-ng application uses the following objects:

- *Source driver*: A communication method used to receive log messages. For example, syslog-ng can receive messages from a remote host via TCP/IP, or read the messages of a local application from a file.
- *Source*: A named collection of configured source drivers.
- *Destination driver*: A communication method used to send log messages. For example, syslog-ng can send messages to a remote host via TCP/IP, or write the messages into a file or database.
- *Destination*: A named collection of configured destination drivers.
- *Filter*: An expression to select messages. For example, a simple filter can select the messages received from a specific host.
- *Macro*: An identifier that refers to a part of the log message. For example, the `$HOST` macro returns the name of the host that sent the message. Macros are often used in templates and filenames.
- *Parser*: A rule that segments messages into separate columns at a predefined separator character (for example a comma). Every column has a unique name that can be used as a macro.

- *Rewrite rule*: A rule modifies a part of the message, for example, replaces a string, or sets a field to a specified value.

- *Log paths*: A combination of sources, destinations, and other objects like filters, parsers, and rewrite rules. The syslog-ng application sends messages arriving to the sources of the log paths to the defined destinations, and performs filtering, parsing, and rewriting of the messages. Log paths are also called log statements. Log statements can include other (embedded) log statements to create complex log paths.

- *Template*: A template is a set of macros that can be used to restructure log messages or automatically generate file names. For example, a template can add the hostname and the date to the beginning of every log message.

- *Option*: Options set global parameters of syslog-ng, like the parameters of name resolution and timezone handling.

For details on the above objects, see *Section 4.2, Defining global objects (p. 43)*.

2.5. Timezone handling

The syslog-ng application supports messages originating from different timezones. The original syslog protocol does not include timezone information, but syslog-ng provides a solution by extending the syslog protocol to include the timezone in the log messages. The syslog-ng application also enables administrators to supply timezone information for legacy devices which do not support the protocol extension.

Timezone information is associated with messages entering syslog-ng is selected using the following algorithm:

Step 1. The sender application (for example the syslog-ng client) or host specifies the timezone of the messages. If the incoming message includes a timezone it is associated with the message. Otherwise, the local timezone is assumed.

Step 2. Specify the `time_zone()` parameter for the source driver that reads the message. This timezone will be associated with the messages only if no timezone is specified within the message itself. Each source defaults to the value of the `recv_time_zone()` global option.

Step 3. Specify the timezone in the destination driver using the `time_zone()` parameter. Each destination driver might have an associated timezone value; syslog-ng converts message timestamps to this timezone before sending the message to its destination (file or network socket). Each destination defaults to the value of the `send_time_zone()` global option.

Note
A message can be sent to multiple destination zones. The syslog-ng application converts the timezone information properly for every individual destination zone.

Step 4. If the timezone is not specified, the message is left unchanged.

Step 5. When macro expansions are used in the destination filenames, the local timezone is used.

2.6. Daylight saving changes

The syslog-ng application receives the timezone and daylight saving information from the operating system it is installed on. If the operating system handles daylight saving correctly, so does syslog-ng.

2.7. Secure logging using TLS

The syslog-ng application can send and receive log messages securely over the network using the Transport Layer Security (TLS) protocol. TLS is an encryption protocol over the TCP/IP network protocol, so it can be used only with TCP-based sources and destinations (`tcp()` and `tcp6()`).

TLS uses certificates to authenticate and encrypt the communication, as illustrated on the following figure:

Figure 2.7. Certificate-based authentication

The client authenticates the server by requesting its certificate and public key. Optionally, the server can also request a certificate from the client, thus mutual authentication is also possible.

In order to use TLS encryption in syslog-ng, the following elements are required:

- A certificate on the syslog-ng server that identifies the syslog-ng server.
- The certificate of the Certificate Authority that issued the certificate of the syslog-ng server must be available on the syslog-ng client.

When using mutual authentication to verify the identity of the clients, the following elements are required:

- A certificate must be available on the syslog-ng client. This certificate identifies the syslog-ng client.
- The certificate of the Certificate Authority that issued the certificate of the syslog-ng client must be available on the syslog-ng server.

Mutual authentication ensures that the syslog-ng server accepts log messages only from authorized clients.

See *Section 4.12, Encrypting log messages with TLS (p. 77)* for details on configuring TLS communication in syslog-ng.

2.8. Formatting messages, filenames, directories, and tablenames

The syslog-ng application can dynamically create filenames, directories, or names of database tables using macros that help you organize your log messages. Macros refer to a property or a part of the log message, for example, the `$HOST` macro refers to the name or IP address of the client that sent the log message, while `$DAY` is the day of the month when syslog-ng has received the message. Using these macros in the path of the destination log files allows you for example to collect the logs of every host into separate files for every day.

A set of macros can be defined as a template object and used in multiple destinations.

Another use of macros and templates is to customize the format of the syslog message, for example to add elements of the message header to the message text. Note that if a message uses the IETF-syslog format, only the text of the message can be customized, the structure of the header is fixed.

For details on using templates and macros, see *Section 4.7, Templates and macros (p. 70)* and *Section 6.5, Macros (p. 154)*.

2.9. Segmenting messages

The filters and default macros of syslog-ng work well on the headers and metainformation of the log messages, but are rather limited when processing the content of the messages. Parsers can segment the content of the messages into name-value pairs, and these names can be used as user-defined macros. Subsequent filtering or other type of processing of the message can use these custom macros to refer to parts of the message.

Parsers are global objects most often used together with filters and rewrite rules. For details on using parsers, see *Section 4.8, Parsing messages (p. 71)* and *Section 6.6, Message parsers (p. 158)*.

2.10. Modifying messages

The syslog-ng application can rewrite parts of the messages using rewrite rules. Rewrite rules are global objects similar to parsers and filters and can be used in log paths. The syslog-ng application has two methods to rewrite parts of the log messages: replacing (setting) a part of the message to a fix value, and a general search-and-replace mode.

Substitution completely replaces a specific part of the message that is referenced using a built-in or user-defined macro.

General rewriting searches for a string in the entire message (or only a part of the message specified by a macro) and replaces it with another string. Optionally, this replacement string can be a template that contains macros.

For details on using rewrite rules, see *Section 4.10, Rewriting messages (p. 75)* and *Section 6.7, Rewriting messages (p. 167)*.

2.11. Classifying log messages

The syslog-ng application can compare the contents of the received log messages to predefined message patterns. By comparing the messages to the known patterns, syslog-ng is able to identify the exact type of the messages, and sort them into message classes. The message classes can be used to classify the type of the event described in the log message. The message classes can be customized, and for example can label the messages as user login, application crash, file transfer, etc. events.

To find the pattern that matches a particular message, syslog-ng uses a method called longest prefix match radix tree. This means that syslog-ng creates a tree structure of the available patterns, where the different characters available in the patterns for a given position are the branches of the tree.

To classify a message, syslog-ng selects the first character of the message (the text of message, not the header), and selects the patterns starting with this character, other patterns are ignored for the rest of the process. After that, the second character of the message is compared to the second character of the selected patterns. Again, matching patterns are selected, and the others discarded. This process is repeated until a single pattern completely matches the message, or no match is found. In the latter case, the message is classified as unknown, otherwise the class of the matching pattern is assigned to the message.

To make the message classification more flexible and robust, the patterns can contain pattern parsers: elements that match on a set of characters. For example, the NUMBER parser matches on any integer or hexadecimal number (for example 1, 123, 894054, 0xFFFF, etc.). Other pattern parsers match on various strings and IP addresses. For the details of available pattern parsers, see *Section 6.6.2.1, Using pattern parsers (p. 161)*.

The functionality of the pattern database is similar to that of the logcheck project, but it is much easier to write and maintain the patterns used by syslog-ng, than the regular expressions used by logcheck. Also, it is much easier to understand syslog-ng pattens than regular expressions.

Pattern matching based on regular expressions is computationally very intensive, especially when the number of patterns increases. The solution used by syslog-ng can be performed real-time, and is independent from the number of patterns, so it scales much better. The following patterns describe the same message: `Accepted password for bazsi from 10.50.0.247 port 42156 ssh2`

A regular expression matching this message from the logcheck project: `Accepted \ (gssapi(-with-mic|-keyex)?|rsa|dsa|password|publickey|keyboard-interactive/pam) \ for [^[:space:]]+ from [^[:space:]]+ port [0-9]+( (ssh|ssh2))?`

A syslog-ng database pattern for this message: `Accepted @QSTRING:auth_method: @ for@QSTRING:username: @from\ @QSTRING:client_addr: @port @NUMBER:port:@ ssh2`

For details on using pattern databases to classify log messages, see *Section 4.9, Classifying messages (p. 72)* and *Section 6.6.2, Pattern databases (p. 161)*.

2.11.1. The structure of the pattern database

The pattern database is organized as follows:

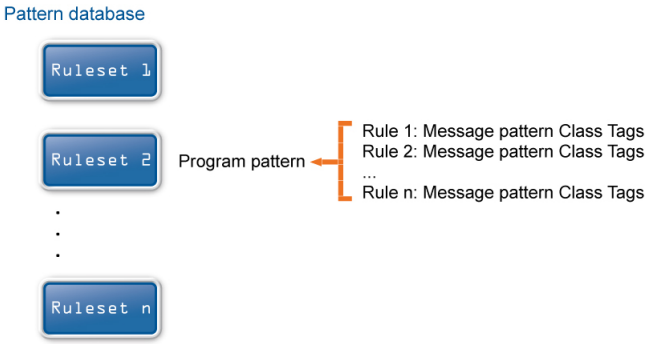

Figure 2.8. The structure of the pattern database

- The pattern database consists of rulesets. A ruleset consists of a Program Pattern and a set of rules: the rules of a ruleset are applied to log messages if the name of the application that sent the message matches the Program Pattern of the ruleset. The name of the application (the content of the $PROGRAM macro) is compared to the Program Patterns of the available rulesets, and then the rules of the matching rulesets are applied to the message.
- The Program Pattern can be a string that specifies the name of the appliation or the beginning of its name (e.g., to match for sendmail, the program pattern can be sendmail, or just send), and the Program Pattern can contain pattern parsers. Note that pattern parsers are completely independent from the syslog-ng parsers used to segment messages. Additionally, every rule has a unique identifier: if a message matches a rule, the identifier of the rule is stored together with the message.
- Rules consist of a message pattern and a class. The Message Pattern is similar to the Program Pattern, but is applied to the message part of the log message (the content of the $MESSAGE macro). If a message pattern matches the message, the class of the rule is assigned to the message (e.g., Security, Violation, etc.).
- Rules can also contain additional information about the matching messages, such as the description of the rule, an URL, name-value pairs, or free-form tags.
- Patterns can consist of literals (keywords, or rather, keycharacters) and pattern parsers.

Note
If the $PROGRAM part of a message is empty, rules with an empty Program Pattern are used to classify the message.

If the same Program Pattern is used in multiple rulesets, the rules of these rulesets are merged, and every rule is used to classify the message. Note that message patterns must be unique within the merged rulesets, but the currently only one ruleset is checked for uniqueness.

2.11.2. How pattern matching works

Figure 2.9. Applying patterns

The followings describe how patterns work. This information applies to program patterns and message patterns alike, even though message patterns are used to illustrate the procedure.

Patterns can consist of literals (keywords, or rather, keycharacters) and pattern parsers. Pattern parsers attempt to parse a sequence of characters according to certain rules.

Note
Wildcards and regular expressions cannot be used in patterns. The `@` character must be escaped, i.e., to match for this character, you have to write `@@` in your pattern. This is required because pattern parsers of syslog-ng are enclosed between `@` characters.

When a new message arrives, syslog-ng attempts to classify it using the pattern database. The available patterns are organized alphabetically into a tree, and syslog-ng inspects the message character-by-character, starting from the beginning. This approach ensures that only a small subset of the rules must be evaluated at any given step, resulting in high processing speed. Note that the speed of classifying messages is practically independent from the total number of rules.

For example, if the message begins with the `Apple` string, only patterns beginning with the character `A` are considered. In the next step, syslog-ng selects the patterns that start with `Ap`, and so on, until there is no more specific pattern left.

Note that literal matches take precedence over pattern parser matches: if at a step there is a pattern that matches the next character with a literal, and another pattern that would match it with a parser, the pattern with the literal match is selected. Using the previous example, if at the third step there is the literal pattern `Apport` and a pattern parser `Ap@STRING@`, the `Apport` pattern is matched, even if the pattern parser would result in a better match.

If there are two parsers at the same level (e.g., `Ap@STRING@` and `Ap@QSTRING@`), it is random which pattern is applied (technically, the one that is loaded first). However, if the selected parser cannot parse at least one character of the message, the other parser is used. But having two different parsers at the same level is extremely rare, so the impact of this limitation is much less than it appears.

2.11.3. Artificial ignorance

Artificial ignorance is a method to detect anomalies. When applied to log analysis, it means that you ignore the regular, common log messages - these are the result of the regular behavior of your system, and therefore are not too interesting. However, new messages that have not appeared in the logs before can sign important events, and should be therefore investigated. "By definition, something we have never seen before is anomalous" (Marcus J. Ranum).

The syslog-ng application can classify messages using a pattern database: messages that do not match any pattern are classified as unknown. This provides a way to use artificial ignorance to review your log messages. You can periodically review the unknown messages — syslog-ng can send them to a separate destination - and add patterns for them to the pattern database. By reviewing and manually classifying the unknown messages, you can iteratively classify more and more messages, until the only the really anomalous messages show up as unknown.

Obviously, for this to work, a large number of message patterns are required. The radix-tree matching method used for message classification is very effective, can be performed very fast, and scales very well; basically the time required to perform a pattern matching is independent from the number of patterns in the database.

To simplify the building of pattern databases, BalaBit has released (and will continue to release) sample databases. Currently the sample pattern databases are available at the *BalaBit Download page*.

2.12. Managing incoming and outgoing messages with flow-control

This section describes the internal message-processing model of syslog-ng, as well as the flow-control feature that can prevent message losses. To use flow-control, the `flow-control` flag must be enabled for the particular log path.

The syslog-ng application monitors (polls) the sources defined in its configuration file, periodically checking each source for messages. When a log message is found in one of the sources, syslog-ng polls every source and reads the available messages. These messages are processed and put into the output buffer of syslog-ng (also called fifo). From the output buffer, the operating system sends the messages to the appropriate destinations.

In large-traffic environments many messages can arrive during a single poll loop, therefore syslog-ng reads only a fixed number of messages from each source. The `log_fetch_limit()` option specifies the number of messages read during a poll loop from a single source.

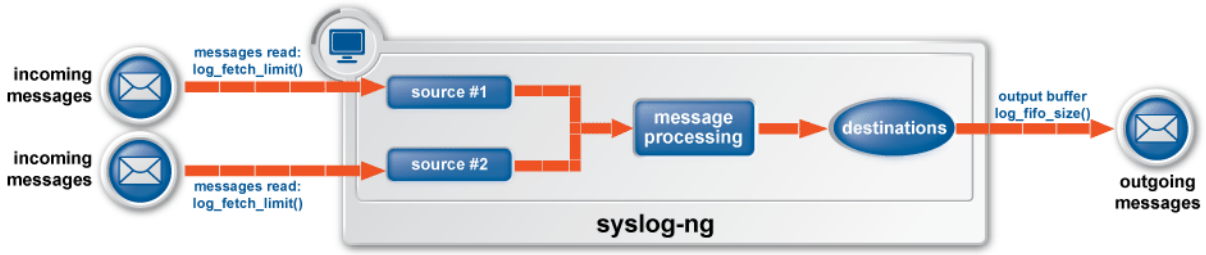

Figure 2.10. Managing log messages in syslog-ng

Note
The `log_fetch_limit()` parameter can be set as a global option, or for every source individually.

Every destination has its own output buffer. The output buffer is needed because the destination might not be able to accept all messages immediately. The `log_fifo_size()` parameter sets the size of the output buffer. The output buffer must be larger than the `log_fetch_limit()` of the sources, to ensure that every message read during the poll loop fits into the output buffer. If the log path sends messages to a destination from multiple sources, the output buffer must be large enough to store the incoming messages of every source.

TCP and unix-stream sources can receive the logs from several incoming connections (for example many different clients or applications). For such sources, syslog-ng reads messages from every connection, thus the `log_fetch_limit()` parameter applies individually to every connection of the source.

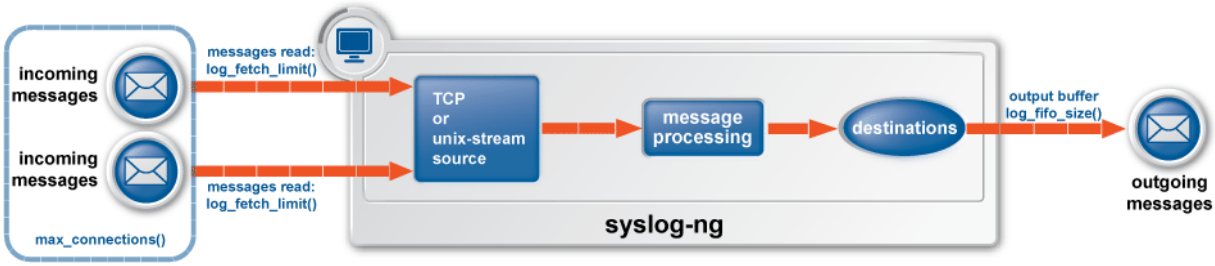

Figure 2.11. Managing log messages of TCP sources in syslog-ng

The flow-control of syslog-ng introduces a control window to the source that tracks how many messages can syslog-ng accept from the source. Every message that syslog-ng reads from the source lowers the window size by one; every message that syslog-ng successfully sends from the output buffer increases the window size by one. If the window is full (i.e., its size decreases to zero), syslog-ng stops reading messages from the source. The initial size of the control window is by default `100`: the `log_fifo_size()` must be larger than this value in order for flow-control to have any effect. If a source accepts messages from multiple connections, all messages use the same control window.

When flow-control is used, every source has its own control window. As a worst-case situation, the output buffer of the destination must be set to accommodate all messages of every control window, that is, the `log_fifo_size()` of the destination must be greater than `number_of_sources*log_iw_size()`. This applies to every source that sends logs to the particular destination. Thus if two sources having several connections and heavy traffic send logs to the same destination, the control window of both sources must fit into the output buffer of the destination. Otherwise, syslog-ng does not activate the flow-control, and messages may be lost.

The syslog-ng application handles outgoing messages the following way:

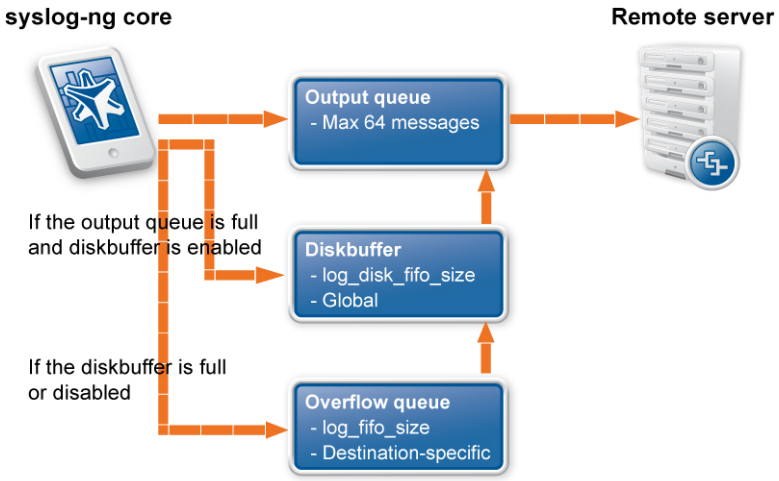

Figure 2.12. Handling outgoing messages in syslog-ng PE

- *Output queue*: Messages from the output queue are sent to the target syslog-ng server. The syslog-ng application puts the outgoing messages directly into the output queue, unless the output queue is full. The output queue can hold 64 messages, this is a fixed value and cannot be modified.

- *Disk buffer*: If the output queue is full and disk-buffering is enabled, syslog-ng Premium Edition puts the outgoing messages into the disk buffer of the destination.

- *Overflow queue*: If the output queue is full and the disk buffer is disabled or full, syslog-ng puts the outgoing messages into the overflow queue of the destination. (The overflow queue is identical to the output buffer used by other destinations.) The `log_fifo_size()` parameter specifies the number of messages stored in the overflow queue. See also *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16)* for details on sizing the `log_fifo_size()` parameter.

2.12.1. Flow-control and multiple destinations

Using flow-control on a source has an important side-effect if the messages of the source are sent to multiple destinations. If flow-control is in use and one of the destinations cannot accept the messages, the other destinations do not receive any messages either, because syslog-ng stops reading the source. For example, if messages from a source are sent to a remote server and also stored locally in a file, and the network connection to the server becomes unavailable, neither the remote server nor the local file will receive any messages.

Note
Creating separate log paths for the destinations that use the same flow-controlled source does not avoid the problem.

2.13. Stable and feature releases of syslog-ng OSE

As of October 2009, the following release policy applies to syslog-ng Open Source Edition:

- *Stable versions*, denoted by a two-digit version number ending with .0 (for example 2.0 or 3.0): Stable branches are supported for at least 1 year, but no more than 2 stable versions of a product are supported at a time. Maintenance releases to the stable branch contain only bugfixes.
- *Feature versions*, denoted by two-digit version number ending with a non-zero version number (for example 3.1, 3.2 and onwards): Feature branches contain enhancements and new features, presumably 1-3 new feature per release. Only the last of the feature releases is supported (for example when a new feature release comes out, the last one becomes unsupported), and the last feature release becomes the new stable release.

Note
Releases of the feature branch are tested just like the stable releases; they are not "unstable" development snapshots. The difference between earlier major releases and current feature releases is the smaller number of features contained in a release, and the shorter support periods. If an unstable snapshot or alpha/beta/rc release is released for public testing, it is always marked explicitly as such.

Warning
Downgrading from a feature release to an earlier (and thus unsupported) feature release, or to the stable release is officially not supported, but usually works as long as your syslog-ng OSE configuration file is appropriate for the old syslog-ng OSE version. However, persistent data like the position of the last processed message in a file source will be probably lost.

2.14. High availability support

Multiple syslog-ng servers can be run in fail-over mode. The syslog-ng application does not include any internal support for this, as clustering support must be implemented on the operating system level. A tool that can be used to create UNIX clusters is Heartbeat (see *http://www.linux-ha.org/* for details).

2.15. Possible causes of losing log messages

During the course of a message from the sending application to the final destination of the message, there are a number of locations where a message may be lost, even though syslog-ng does its best to avoid message loss. Usually losing messages can be avoided with careful planning and proper configuration of syslog-ng and the hosts running syslog-ng. The following list shows the possible locations where messages may be lost, and provides methods to minimize the risk of losing messages.

Note
The following list covers the main possibilities of losing messages, but does not take into account the possible use of flow-control (see *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16)*). This topic will be addressed in more detail in the future releases of this guide.

- *Between the application and the syslog-ng client*: Make sure to use an appropriate source to receive the logs from the application (for example from `/dev/log`). For example, use `unix-stream` instead of `unix-dgram` whenever possible.
- *When syslog-ng is sending messages*: If syslog-ng cannot send messages to the destination and the output buffer gets full, syslog-ng will drop messages. The number of dropped messages is displayed per destination in the log message statistics of syslog-ng (see *Section 4.3.1.1, Log statistics (p. 47)* for details).

- *On the network*: When transferring messages using the UDP protocol, messages may be lost without any notice or feedback — such is the nature of the UDP protocol. Always use the TCP protocol to transfer messages over the network whenever possible.
- *In the socket receive buffer*: When transferring messages using the UDP protocol, the UDP datagram (i.e., the message) that reaches the receiving host placed in a memory area called the `socket receive buffer`. If the host receives more messages than it can process, this area overflows, and the kernel drops messages without letting syslog-ng know about it. Using TCP instead of UDP prevents this issue. If you must use the UDP protocol, increase the size of the receive buffer using the `so_rcvbuf()` option.
- *When syslog-ng is receiving messages*: The receiving syslog-ng (for example the syslog-ng server or relay) may drop messages if the fifo of the destination file gets full. The number of dropped messages is displayed per destination in the log message statistics of syslog-ng (see *Section 4.3.1.1, Log statistics (p. 47)* for details).
- *When the destination cannot handle large load*: When syslog-ng is sending messages at a high rate into an SQL database, a file, or another destination, it is possible that the destination cannot handle the load, and processes the messages slowly. As a result, the buffers of syslog-ng fill up, syslog-ng cannot process the incoming messages, and starts to loose messages. See the previous entry for details. Use the `throttle` parameter to avoid this problem.
- *As a result of an unclean shutdown of the syslog-ng server*: If the host running the syslog-ng server experiences an unclean shutdown, it takes time until the clients realize that the connection to the syslog-ng server is down. Messages that are put into the output TCP buffer of the clients during this period are not sent to the server.

2.16. The structure of a log message

The following sections describe the structure of log messages. Currently there are two standard syslog message formats:

- The old standard described in RFC 3164 (also called the BSD-syslog or the legacy-syslog protocol): see *Section 2.16.1, BSD-syslog or legacy-syslog messages (p. 20)*
- The new standard described in RFC 5424 (also called the IETF-syslog protocol): see *Section 2.16.2, IETF-syslog messages (p. 22)*

2.16.1. BSD-syslog or legacy-syslog messages

This section describes the format of a syslog message, according to the legacy-syslog or BSD-syslog protocol (see RFC 3164 *http://www.ietf.org/rfc/rfc3164.txt*). A syslog message consists of the following parts:

- `PRI`
- `HEADER`
- `MSG`

The total message cannot be longer than 1024 bytes.

The following is a sample syslog message: `<133>Feb 25 14:09:07 webserver syslogd: restart`. The message corresponds to the following format: `<priority>timestamp hostname application: message`. The different parts of the message are explained in the following sections.

Note
The syslog-ng application supports longer messages as well. For details, see the `log_msg_size()` option in *Section 6.9, Global options (p. 170)*. However, it is not recommended to enable messages larger than the packet size when using UDP destinations.

2.16.1.1. The PRI message part

The PRI part of the syslog message (known as Priority value) represents the Facility and Severity of the message. Facility represents the part of the system sending the message, while severity marks its importance. The Priority value is calculated by first multiplying the Facility number by 8 and then adding the numerical value of the Severity. The possible facility and severity values are presented below.

Note
Facility codes may slightly vary between different platforms. The syslog-ng application accepts facility codes as numerical values as well.

Numerical Code	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	security/authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon
16-23	locally used facilities (local0-local7)

Table 2.1. syslog Message Facilities

The following table lists the severity values.

Numerical Code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Table 2.2. syslog Message Severities

2.16.1.2. The HEADER message part

The HEADER part contains a timestamp and the hostname (without the domain name) or the IP address of the device. The timestamp field is the local time in the `Mmm dd hh:mm:ss` format, where:

- *Mmm* is the English abbreviation of the month: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec.
- *dd* is the day of the month on two digits. If the day of the month is less than 10, the first digit is replaced with a space. (For example `Aug 7`.)
- *hh:mm:ss* is the local time. The hour (hh) is represented in a 24-hour format. Valid entries are between 00 and 23, inclusive. The minute (mm) and second (ss) entries are between 00 and 59 inclusive.

Note
The syslog-ng application supports other timestamp formats as well, like ISO, or the PIX extended format. For details, see the `ts_format()` option in *Section 6.9, Global options (p. 170)*.

2.16.1.3. The MSG message part

The MSG part contains the name of the program or process that generated the message, and the text of the message itself. The MSG part is usually in the following format: `program[pid]: message text`.

2.16.2. IETF-syslog messages

This section describes the format of a syslog message, according to the IETF-syslog protocol (see RFC 5424-5428 *http://tools.ietf.org/html/rfc5424*).A syslog message consists of the following parts:

- `HEADER (includes the PRI as well)`
- `STRUCTURED-DATA`
- `MSG`

The following is a sample syslog message:[1]

```
<34>1 2003-10-11T22:14:15.003Z mymachine.example.com su - ID47 - BOM'su root' failed
 for lonvick on /dev/pts/8
```

The message corresponds to the following format:

```
<priority>VERSION ISOTIMESTAMP HOSTNAME APPLICATION PID MESSAGEID STRUCTURED-DATA
 MSG
```

In this example, the Facility has the value of 4, severity is 2, so PRI is 34. The VERSION is 1. The message was created on 11 October 2003 at 10:14:15pm UTC, 3 milliseconds into the next second. The message originated from a host that identifies itself as "mymachine.example.com". The APP-NAME is "su" and the PROCID is unknown. The MSGID is "ID47". The MSG is "'su root' failed for lonvick...", encoded in UTF-8. The encoding is defined by the BOM. There is no STRUCTURED-DATA present in the message, this is indicated by "-" in the STRUCTURED-DATA field. The MSG is "'su root' failed for lonvick...".

The HEADER part of the message must be in plain ASCII format, the parameter values of the STRUCTURED-DATA part must be in UTF-8, while the MSG part should be in UTF-8. The different parts of the message are explained in the following sections.

2.16.2.1. The PRI message part

The PRI part of the syslog message (known as Priority value) represents the Facility and Severity of the message. Facility represents the part of the system sending the message, while severity marks its importance. The Priority value is calculated by first multiplying the Facility number by 8 and then adding the numerical value of the Severity. The possible facility and severity values are presented below.

Note
Facility codes may slightly vary between different platforms. The syslog-ng application accepts facility codes as numerical values as well.

Numerical Code	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem

[1]Source: http://tools.ietf.org/html/rfc5424

Numerical Code	Facility
9	clock daemon
10	security/authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon
16-23	locally used facilities (local0-local7)

Table 2.3. syslog Message Facilities

The following table lists the severity values.

Numerical Code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Table 2.4. syslog Message Severities

2.16.2.2. The HEADER message part

The HEADER part contains the following elements:

- *VERSION*: Version number of the syslog protocol standard. Currently this can only be `1`.
- *ISOTIMESTAMP*: The time when the message was generated in the ISO 8601 compatible standard timestamp format (yyyy-mm-ddThh:mm:ss+-ZONE), for example: `2006-06-13T15:58:00.123+01:00`.
- *HOSTNAME*: The machine that originally sent the message.
- *APPLICATION*: The device or application that generated the message
- *PID*: The process name or process ID of the syslog application that sent the message. It is not necessarily the process ID of the application that generated the message.
- *MESSAGEID*: The ID number of the message.

Note
The syslog-ng application supports other timestamp formats as well, like ISO, or the PIX extended format. The timestamp used in the IETF-syslog protocol is derived from RFC3339, which is based on ISO8601. For details, see the `ts_format()` option in *Section 6.9, Global options (p. 170)*.

2.16.2.3. The STRUCTURED-DATA message part

The STRUCTURED-DATA message part may contain meta- information about the syslog message, or application-specific information such as traffic counters or IP addresses. STRUCTURED-DATA consists of data blocks enclosed in brackets (`[]`). Every block include the ID of the block, and one or more `name=value` pairs. The syslog-ng application automatically parses the STRUCTURED-DATA part of syslog messages, which can be referenced in macros (see *Section 6.5, Macros (p. 154)* for details). An example STRUCTURED-DATA block looks like:

```
[exampleSDID@0 iut="3" eventSource="Application" eventID="1011"][examplePriority@0
 class="high"]
```

2.16.2.4. The MSG message part

The MSG part contains the text of the message itself. The encoding of the text must be UTF-8 if the BOM character is present in the message. If the message does not contain the BOM character, the encoding is treated as unknown. Usually messages arriving from legacy sources do not include the BOM character.

Chapter 3. Installing syslog-ng

This chapter explains how to install syslog-ng Open Source Edition on various platforms using the precompiled binary files.

Version 3.0 of syslog-ng features a unified installer package with identical look on every supported Linux platform.

Note
For instructions on compiling syslog-ng Open Source Edition from the source code, see *Section 3.4, Compiling syslog-ng from source (p. 33)*.

As of syslog-ng Open Source Edition 3.0.2, binary installation packages of syslog-ng OSE are available for free for the supported Linux and BSD platforms.

Third-party packages available for various other platforms are listed in *Section 1.6, Supported platforms (p. 3)*.

The syslog-ng binaries include all required libraries and dependencies of syslog-ng. The components are installed into the `/opt/syslog-ng` directory. It can automatically re-use existing configuration files, and also generate a simple configuration automatically into the `/opt/syslog-ng/etc/syslog-ng.conf` file.

Note
There are two versions of every binary release. The one with the `client` suffix does not include the libraries required to log into SQL databases. If you are installing syslog-ng in client or relay mode, or you do not use the `sql()` destination, use these binaries. That way no unnecessary components are installed to your system.

The syslog-ng application can be installed interactively following the on-screen instructions as described in *Section 3.1, Installing syslog-ng using the .run installer (p. 26)*, and also without user interaction using the silent installation option — see *Section 3.1.3, Installing syslog-ng without user-interaction (p. 31)*.

3.1. Installing syslog-ng using the .run installer

This section describes how to install the syslog-ng application interactively using the binary installer. The installer has a simple interface: use the TAB or the arrow keys of your keyboard to navigate between the options, and Enter to select an option.

- To install syslog-ng on clients or relays, complete *Section 3.1.1, Installing syslog-ng in client or relay mode (p. 27)*.
- To install syslog-ng on your central logserver, complete *Section 3.1.2, Installing syslog-ng in server mode (p. 29)*.
- To install syslog-ng without any user-interaction, complete *Section 3.1.3, Installing syslog-ng without user-interaction (p. 31)*.

Note
The installer stops the running syslogd application if it is running, but its components are not removed. The `/etc/init.d/sysklogd` init script is automatically renamed to `/etc/init.d/sysklogd.backup`. Rename this file to its original name if you want to remove syslog-ng or restart the syslogd package.

3.1.1. Installing syslog-ng in client or relay mode

Complete the following steps to install syslog-ng Open Source Edition on clients or relays. See *Section 2.3, Modes of operation (p. 7)* for details on the different operation modes of syslog-ng.

3.1.1.1. Procedure – Installing syslog-ng in client or relay mode

Step 1. Enable the executable attribute for the installer using the `chmod +x syslog-ng-<edition>-<version>-<OS>-<platform>.run`, then start the installer as root using the `./syslog-ng-<edition>-<version>-<OS>-<platform>.run` command. (Note that the exact name of the file depends on the operating system and platform.) Wait until the package is uncompressed and the welcome screen appears, then select **Continue**.

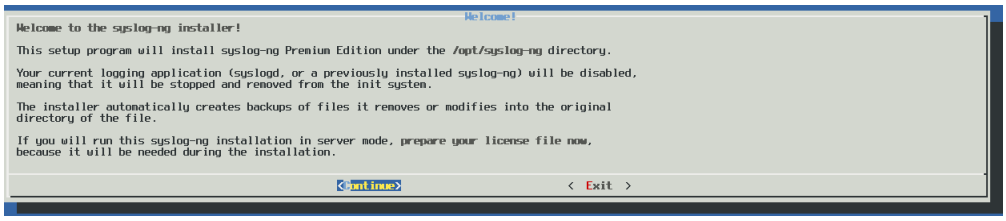

Figure 3.1. The welcome screen

Step 2. *Accepting the EULA*: You can install syslog-ng only if you understand and accept the terms of the End-User License Agreement (EULA). The full text of the EULA can be displayed during installation by selecting the **Show EULA** option, and is also available in this guide for convenience at *Appendix 2, GNU General Public License (p. 195)*. Select **Accept** to accept the EULA and continue the installation.
If you do not accept the terms of the EULA for some reason, select **Reject** to cancel installing syslog-ng.

Step 3. *Detecting platform and operating system*: The installer attempts to automatically detect your oprating system and platform. If the displayed information is correct, select **Yes**. Otherwise select **Exit** to abort the installation, and verify that your platform is supported. See *Section 1.6, Supported platforms (p. 3)* for a list of supported platforms. If your platform is supported but not detected correctly, contact your local distributor, reseller, or the BalaBit Support Team. See *Section 5, Contact and support information (p. xi)* for contact details.

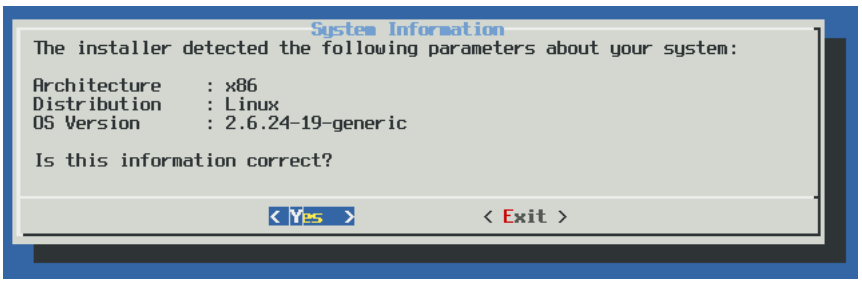

Figure 3.2. Platform detection

Step 4. *Upgrading*: The syslog-ng installer can automatically detect if you have previously installed a version of syslog-ng on your system. To use the configuration file of this previous installation, select **Yes**. To ignore the old configuration file and create a new one, select **No**.

Note that if you decide to use your existing configuration file, the installer automatically checks it for syntax error and displays a list of warnings and errors if it finds any problems.

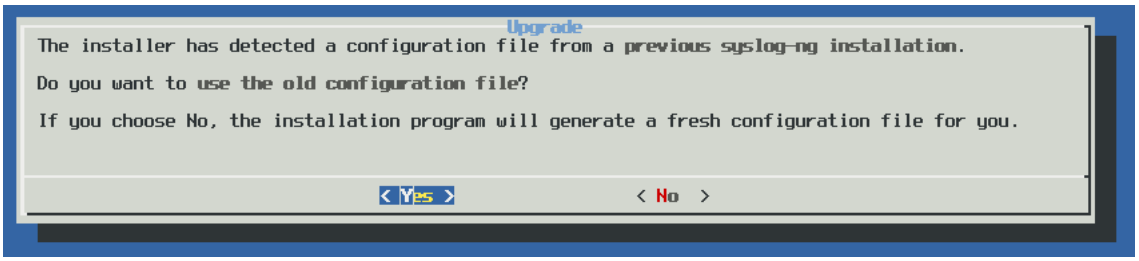

Figure 3.3. Upgrading syslog-ng

Step 5. *Generating a new configuration file*: The installer displays some questions to generate a new configuration file.

Step a. *Remote sources*: Select **Yes** to accept log messages from the network. TCP, UDP, and SYSLOG messages on every interface will be automatically accepted.

Figure 3.4. Accepting remote messages

Step b. *Remote destinations*: Enter the IP address or hostname of your logserver or relay and select **OK**.

Figure 3.5. Forwarding messages to the logserver

Note
Accepting remote messages and forwarding them to a logserver means that syslog-ng will start in relay mode.

Step 6. After the installation is finished, add the `/opt/syslog-ng/bin` and `/opt/syslog-ng/sbin` directories to your search PATH environment variable. That way you can use syslog-ng and its related tools without having to specify the full pathname. Add the following line to your shell profile:

```
PATH=/opt/syslog-ng/bin:$PATH
```

Note
The native logrotation tools do not send a SIGHUP to syslog-ng after rotating the log files, causing syslog-ng to write into files already rotated. To solve this problem, the syslog-ng init script links the `/var/run/syslog.pid` file to syslog-ng's pid. Also, on Linux, the *`install.sh`* script symlinks the initscript of the original syslog daemon to syslog-ng's initscript.

3.1.2. Installing syslog-ng in server mode

Complete the following steps to install syslog-ng on logservers. See *Section 2.3, Modes of operation (p. 7)* for details on the different operation modes of syslog-ng.

3.1.2.1. Procedure – Installing syslog-ng in server mode

Step 1. Enable the executable attribute for the installer using the `chmod +x syslog-ng-<edition>-<version>-<OS>-<platform>.run`, then start the installer as root using the `./syslog-ng-<edition>-<version>-<OS>-<platform>.run` command. (Note that the exact name of the file depends on the operating system and platform.) Wait until the package is uncompressed and the welcome screen appears, then select **Continue**.

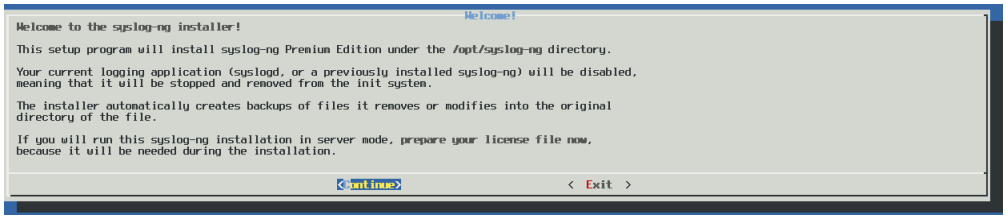

Figure 3.6. The welcome screen

Step 2. *Accepting the EULA*: You can install syslog-ng only if you understand and accept the terms of the End-User License Agreement (EULA). The full text of the EULA can be displayed during installation by selecting the **Show EULA** option, and is also available in this guide for convenience at *Appendix 2, GNU General Public License (p. 195)*. Select **Accept** to accept the EULA and continue the installation.
If you do not accept the terms of the EULA for some reason, select **Reject** to cancel installing syslog-ng.

Step 3. *Detecting platform and operating system*: The installer attempts to automatically detect your oprating system and platform. If the displayed information is correct, select **Yes**. Otherwise select **Exit** to abort the installation, and verify that your platform is supported. See *Section 1.6, Supported platforms (p. 3)* for a list of supported platforms. If your platform is supported but not detected correctly, contact your local distributor, reseller, or the BalaBit Support Team. See *Section 5, Contact and support information (p. xi)* for contact details.

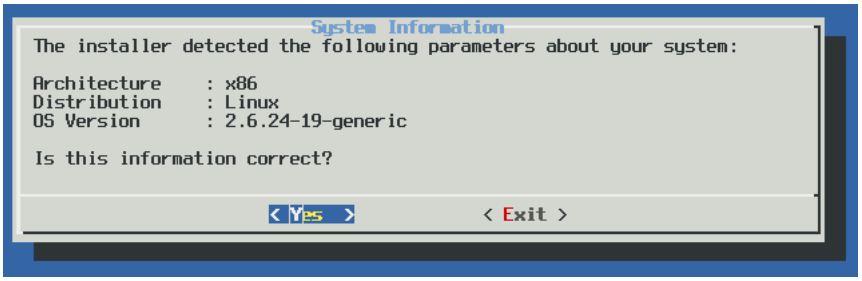

Figure 3.7. Platform detection

Step 4. *Upgrading*: The syslog-ng installer can automatically detect if you have previously installed a version of syslog-ng on your system. To use the configuration file of this previous installation, select **Yes**. To ignore the old configuration file and create a new one, select **No**.
Note that if you decide to use your existing configuration file, the installer automatically checks it for syntax error and displays a list of warnings and errors if it finds any problems.

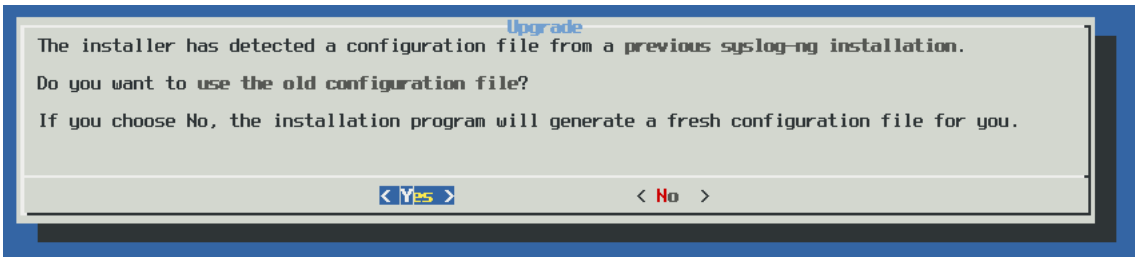

Figure 3.8. Upgrading syslog-ng

Step 5. *Generating a new configuration file*: The installer displays some questions to generate a new configuration file.

Step a. *Remote sources*: Select **Yes** to accept log messages from the network. TCP, UDP, and SYSLOG messages on every interface will be automatically accepted.

Figure 3.9. Accepting remote messages

Step b. *Remote destinations*: Enter the IP address or hostname of your logserver or relay and select **OK**.

Figure 3.10. Forwarding messages to the logserver

Note
Accepting remote messages and forwarding them to a logserver means that syslog-ng will start in relay mode.

Step 6. After the installation is finished, add the `/opt/syslog-ng/bin` and `/opt/syslog-ng/sbin` directories to your search PATH environment variable. That way you can use syslog-ng and its related tools without having to specify the full pathname. Add the following line to your shell profile:

```
PATH=/opt/syslog-ng/bin:$PATH
```

Note
The native logrotation tools do not send a SIGHUP to syslog-ng after rotating the log files, causing syslog-ng to write into files already rotated. To solve this problem, the syslog-ng init script links the `/var/run/syslog.pid` file to syslog-ng's pid. Also, on Linux, the *`install.sh`* script symlinks the initscript of the original syslog daemon to syslog-ng's initscript.

3.1.3. Installing syslog-ng without user-interaction

The syslog-ng application can be installed in silent mode without any user-interaction by specifying the required parameters from the command line. Answers to every question of the installer can be set in advance using command-line parameters.

```
./syslog-ng-<version>.run -- [options]
```

Warning
The `--` characters between the executable and the parameters are mandatory, like in the following example: *`./syslog-ng-3.0.1b-solaris-10-sparc-client.run -- --accept-eula`*

To display the list of parameters, execute the `./syslog-ng-<version>.run -- --h` command. Currently the following options are available:

- *--accept-eula or -a*: Accept the EULA.
- *--upgrade | -u*: Perform automatic upgrade — use the configuration file from an existing installation.

- *--remote <destination host>*: Send logs to the specified remote server. Not available when performing an upgrade.
- *--network*: Accept messages from the network. Not available when performing an upgrade.
- *--configuration <file>*: Use the specified configuration file.

3.2. Installing syslog-ng on RPM-based platforms (Red Hat, SUSE, AIX)

To install syslog-ng on operating systems that use the Red Hat Package Manager (RPM), complete the following steps. Installing syslog-ng automatically replaces the original syslog service. The following supported operating systems use RPM:

- AIX 5.2 and 5.3
- CentOS 4 and 5
- openSUSE Linux Enterprise Server 10.0 and 10.1
- Red Hat Enterprise Server 4 and 5
- SUSE Linux Enterprise Server 10 and 10 SP1

3.2.1. Procedure – Installing syslog-ng on RPM-based systems

Step 1. Login to your MyBalabit account (*http://www.balabit.com/mybalabit*) and download the syslog-ng RPM package for your system from *http://www.balabit.com/network-security/syslog-ng/central-syslog-server/upgrades/*.

Step 2.

- If the host already uses syslog-ng for logging, execute the following command as root. Otherwise, skip this step.

```
rpm -U syslog-ng-<version>-<OS>-<arch>.rpm
```

The syslog-ng application and all its dependencies will be installed, and the configuration of the existing syslog-ng installation will be used.

Note
If you are upgrading from syslog-ng version 2.1, note that the location of the configuration file has been moved to `/opt/syslog-ng/etc/syslog-ng.conf`

- Execute the following command as root:

```
rpm -i syslog-ng-<version>-<OS>-<arch>.rpm
```

The syslog-ng application and all its dependencies will be installed.

Step 3. Answer the configuration questions of syslog-ng. These are described in detail in *Section 3.1, Installing syslog-ng using the .run installer (p. 26)*.

Step 4.

Warning
When performing an upgrade, the package manager might automatically execute the post-uninstall script of the upgraded package, stopping syslog-ng and starting syslogd. If this happens, stop syslogd and start syslog-ng by issuing the following commands:

```
/etc/init.d/syslogd stop
/etc/init.d/syslog-ng start
```

This behavior has been detected on CentOS 4 systems, but may occur on other rpm-based platforms as well.

Step 5. *Optional step for AIX systems*: To redirect the messages of the AIX Error log into syslog, create a file (for example `/tmp/syslog-ng.add`) with the following contents:

```
errnotify:
en_name = "syslog1"
en_persistenceflg = 1
en_method = "logger Msg from Error Log: `errpt -l $1 | grep -v 'ERROR_ID
TIMESTAMP'`"
```

Then execute the following command as root: `odmadd /tmp/syslog-ng.add`.

3.3. Installing syslog-ng on Debian-based platforms

To install syslog-ng on operating systems that use the Debian Software Package (deb) format, complete the following steps. The following supported operating systems use this format:

- Debian etch

3.3.1. Procedure – Installing syslog-ng on Debian-based systems

Step 1. Login to your MyBalabit account (*http://www.balabit.com/mybalabit*) and download the syslog-ng DEB package for your system from *http://www.balabit.com/network-security/syslog-ng/central-syslog-server/upgrades/*.

Step 2. Issue the following command as root:
`dpkg -i syslog-ng-<version>-<OS>-<arch>.deb`

Step 3. Answer the configuration questions of syslog-ng. These are described in detail in *Section 3.1, Installing syslog-ng using the .run installer (p. 26)*.

3.4. Compiling syslog-ng from source

To compile syslog-ng Open Source Edition (OSE) from the source code, complete the following steps. Alternatively, you can use the precompiled binary packages. Precompiled binary packages are available for free for the supported Linux and BSD platforms at *http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/upgrades/*.

3.4.1. Procedure – Compiling syslog-ng from source

Step 1. Download the latest version of syslog-ng OSE from *https://www.balabit.com/downloads/files/syslog-ng/sources/stable/*. The source code is available as a tar.gz archive file.

Step 2. Download the latest version of the EventLog library available at *https://www.balabit.com/downloads/files/eventlog/0.2/*.

Step 3. Install the following packages that are required to compile syslog-ng. These packages are available for most UNIX/Linux systems. Alternatively, you can also download the sources and compile them.

- the *gcc* C compiler (at least version 2.7.2),
- the *GNU flex* lexical analyser generator, available at *http://flex.sourceforge.net/*;
- the *bison* parser generator, available at *http://ftp.gnu.org/gnu/bison/*;
- and the development files of the *glib* library, available at *http://freshmeat.net/projects/glib/*.

Step 4. If you want to use the spoof-source function of syslog-ng, install the development files of the *libnet* library, available at *http://libnet.sourceforge.net*.

Step 5. If you want to use the */etc/hosts.deny* and */etc/hosts.allow* for TCP access, install the development files of the *libwrap* (also called TCP-wrappers) library, available at *ftp://ftp.porcupine.org/pub/security/index.html*.

Step 6. Uncompress the eventlog archive using the

```
$ tar xvfz eventlog-x.x.x.x.tar.gz
```

or the

```
$ gunzip -c eventlog-x.x.x.x.tar.gz | tar xvf -
```

command. A new directory containing the source code of eventlog will be created.

Step 7. By default, eventlog creates a file used by the syslog-ng configure script in the */usr/local/lib/pkgconfig* directory. Issue the following command to add this directory to your PKG_CONFIG_PATH:

```
PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
```

Step 8. Enter the new directory and issue the following commands:

```
$ ./configure
$ make
$ make install
```

Step 9. Uncompress the syslog-ng archive using the

```
tar xvfz syslog-ng-x.xx.tar.gz
```

or the

```
unzip -c syslog-ng-x.xx.tar.gz | tar xvf -
```

command. A new directory containing the source code of syslog-ng will be created.

Step 10. Enter the new directory and issue the following commands:

```
$ ./configure
$ make
$ make install
```

These commands will build syslog-ng using its default options.

Step 11. If needed, use the following options to change how syslog-ng is compiled using the following command syntax:

```
$ ./configure --compile-time-option-name
```

Note
You can also use *--disable options*, to explicitly disable a feature and override autodetection. For example, to disable the TCP-wrapper support, use the *--disable-tcp-wrapper* option.

Warning
Starting with syslog-ng Open Source Edition 3.0.2, default linking mode of syslog-ng is `dynamic`. This means that syslog-ng might not be able to start up if the `/usr` directory is on NFS. On platforms where syslog-ng is used as a system logger, the `--enable-mixed-linking` is preferred.

- *--enable-debug* Include debug information.
- *--enable-dynamic-linking* Compile syslog-ng as a completely dynamic binary. If not specified syslog-ng uses mixed linking (`--enable-mixed-linking`): it links dynamically to system libraries and statically to everything else.
- *--enable-ipv6* Enable IPv6 support.
- *--enable-linux-caps* Enable support for capabilities on Linux.
- *--enable-pcre* Enable using PCRE-type regular expressions. Requires the `libpcre` library package.
- *--enable-spoof-source* Enable spoof_source feature (disabled by default).
- *--enable-static-linking* Compile syslog-ng as a static binary.
- *--enable-sun-door* Enable Sun door support even if not detected (autodetected by default).
- *--enable-sun-streams* Enable Sun STREAMS support even if not detected (autodetected by default).
- *--enable-tcp-wrapper* Enable using */etc/hosts.deny* and */etc/hosts.allow* for TCP access (enabled automatically if the `libwrap` libraries are detected).
- *--with-timezone-dir* Specifies the directory where syslog-ng looks for the timezone files to resolve the `time_zone()` and `local_time_zone()` options. If not specified, the `/opt/syslog-ng/share/zoneinfo/` and `/usr/share/zoneinfo/` directories are checked, respectively. Note that HP-UX uses a unique file format (`tztab`) to describe the timezone information; that format is currently not supported in syslog-ng. As a workaround, copy the zoneinfo files from another, non-HP-UX system to the `/opt/syslog-ng/share/zoneinfo/` directory of your HP-UX system.

For information on configuring syslog-ng, see the *Chapter 4, Configuring syslog-ng (p. 42)*.

3.5. Uninstalling syslog-ng

If you need to uninstall syslog-ng for some reason, you have the following options:

- *If you have installed syslog-ng using the .run installer*: Execute the `uninstall.sh` script located at `/opt/syslog-ng/bin/uninstall.sh`. The uninstall script will automatically restore the syslog daemon used before installing syslog-ng. To completely remove syslog-ng, including the configuration files, use the `uninstall.sh --purge` command.

- *If you have installed syslog-ng from a .deb package*: Execute the `dpkg -r syslog-ng` command to remove syslog-ng; or the `dpkg -P syslog-ng` command to remove syslog-ng and the configuration files as well. Note that removeing syslog-ng does not restore the syslog daemon used before syslog-ng.
- *If you have installed syslog-ng from an .rpm package*: Execute the `rpm -e syslog-ng` command to remove syslog-ng. Note that removing syslog-ng does not restore the syslog daemon used before syslog-ng.

3.6. Configuring Microsoft SQL Server to accept logs from syslog-ng

Complete the following steps to configure your Microsoft SQL Server to enable remote logins and accept log messages from syslog-ng.

3.6.1. Procedure – Configuring Microsoft SQL Server to accept logs from syslog-ng

Step 1. Start the SQL Server Management Studio application. Select **Start > Programs > Microsoft SQL Server 2005 > SQL Server Management Studio**.

Step 2. Create a new database.

Step a.

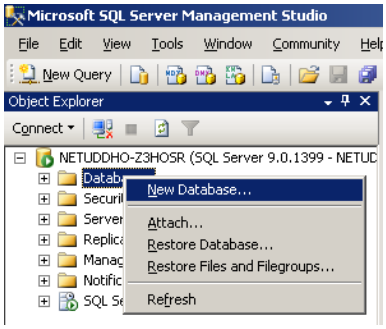

Figure 3.11. Creating a new MSSQL database 1.

In the Object Explorer, right-click on the **Databases** entry and select **New Database**.

Step b.

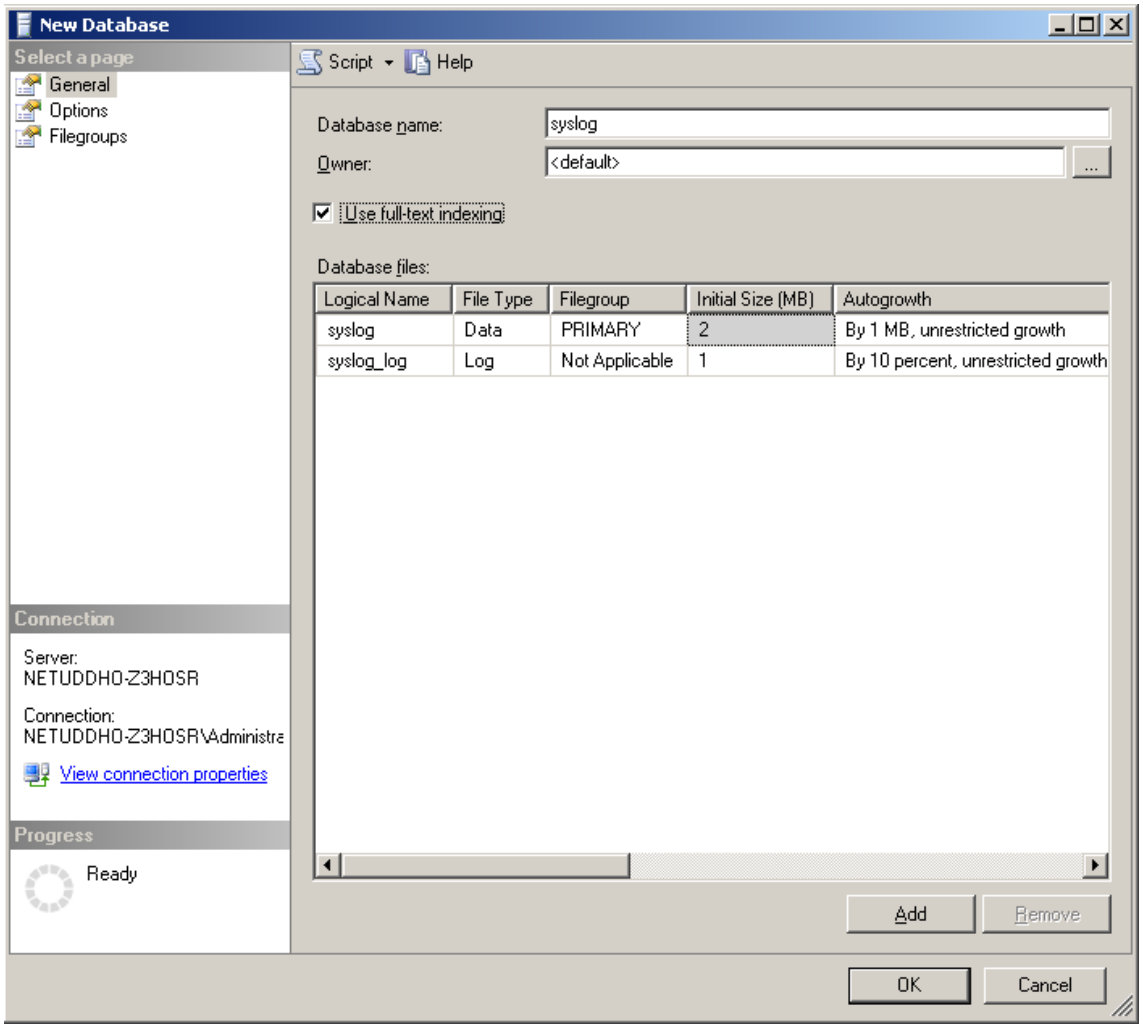

Figure 3.12. Creating a new MSSQL database 2.

Enter the name of the new database (for example `syslogng`) into the **Database name** field and click **OK**.

Step 3. Create a new database user and associate it with the new database.

Step a.

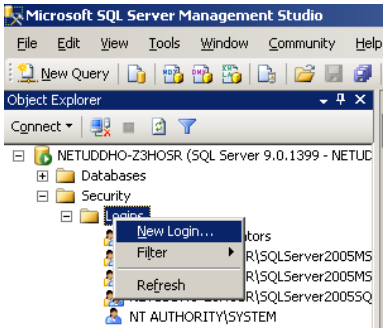

Figure 3.13. Creating a new MSSQL user 1.

In the Object Explorer, select **Security**, right-click on the **Logins** entry, then select **New Login**.

Step b.

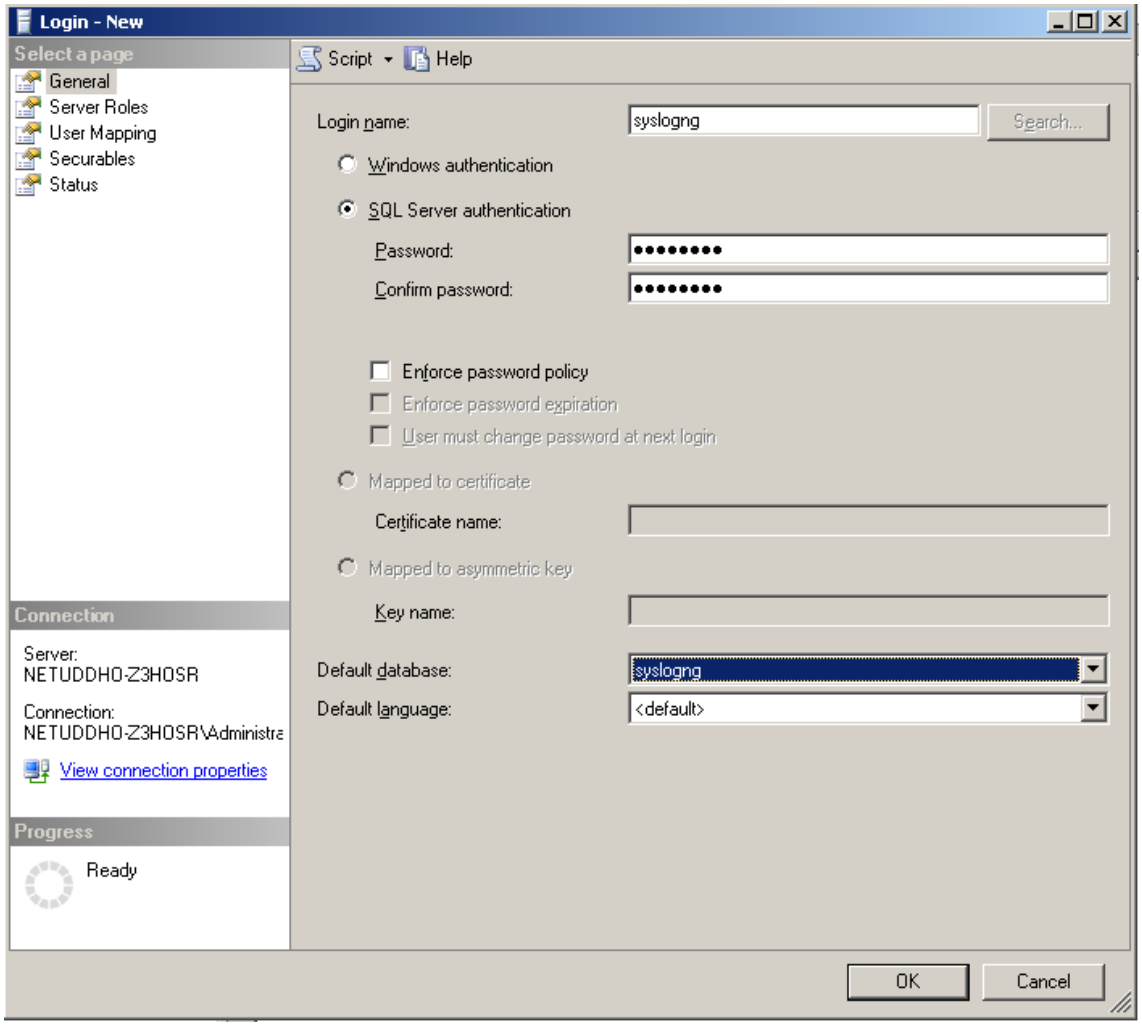

Figure 3.14. Creating a new MSSQL user 2.

Enter a name (for example `syslog-ng`) for the user into the **Login name** field.

Step c. Select the **SQL Server Authentication** option and enter a password for the user.

Step d. In the **Default database** field, select the database created in Step 2 (for example *syslogng*).

Step e. In the **Default language** field, select the language of log messages that you want to store in the database, then click **OK**.

Warning
Incorrect language settings may result in the database converting the messages to a different character-encoding format. That way the log messages may become unreadable, causing information loss.

Step f. In the Object Explorer, select **Security > Logins**, then right-click on the new login created in the previous step, and select **Properties**.

Step g.

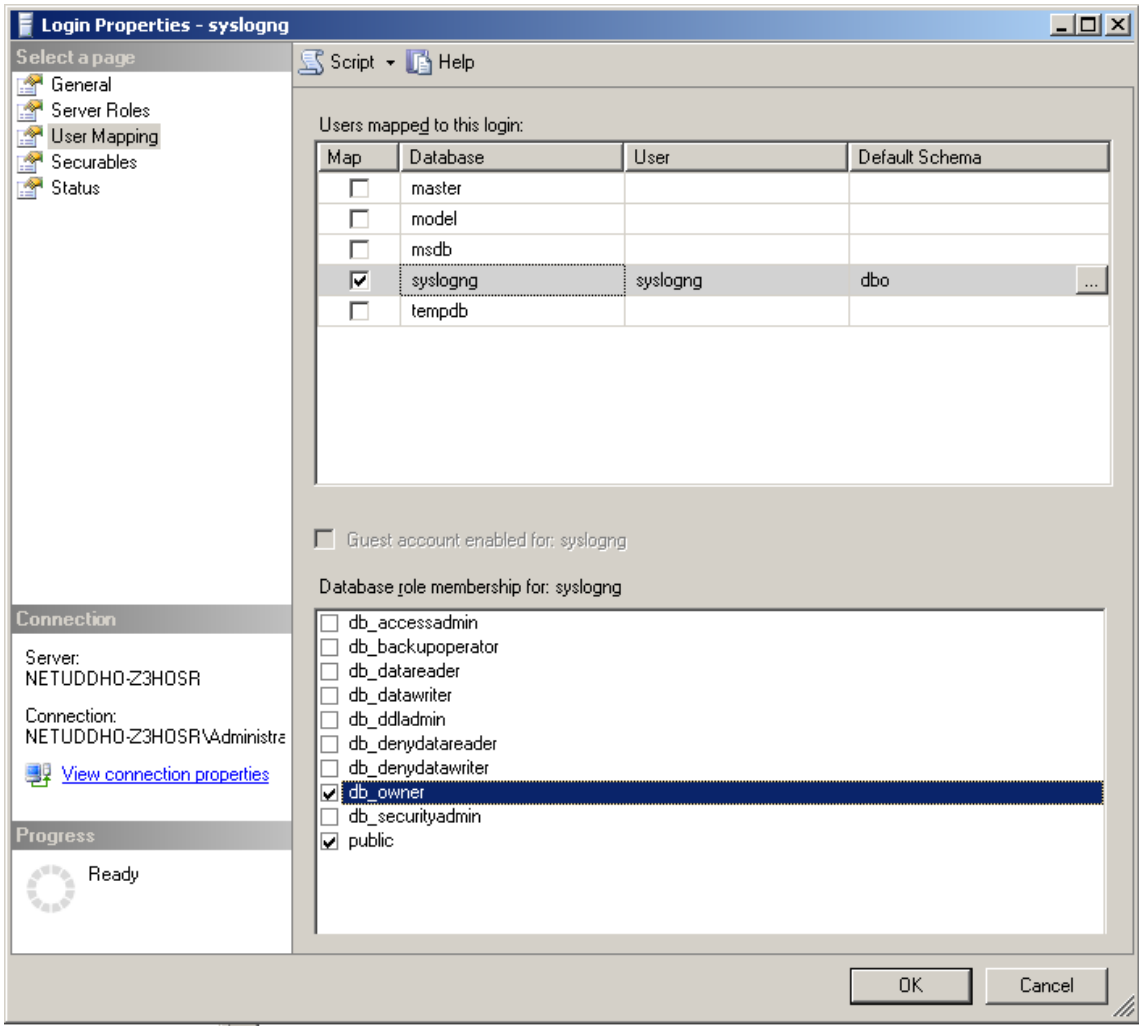

Figure 3.15. Associating database with the new user

Select **User Mapping**. In the **Users mapped to this login** option, check the line corresponding to the new login (for example `syslogng`). In the **Database role membership** field, check the **db_owner** and **public** options.

Step 4.

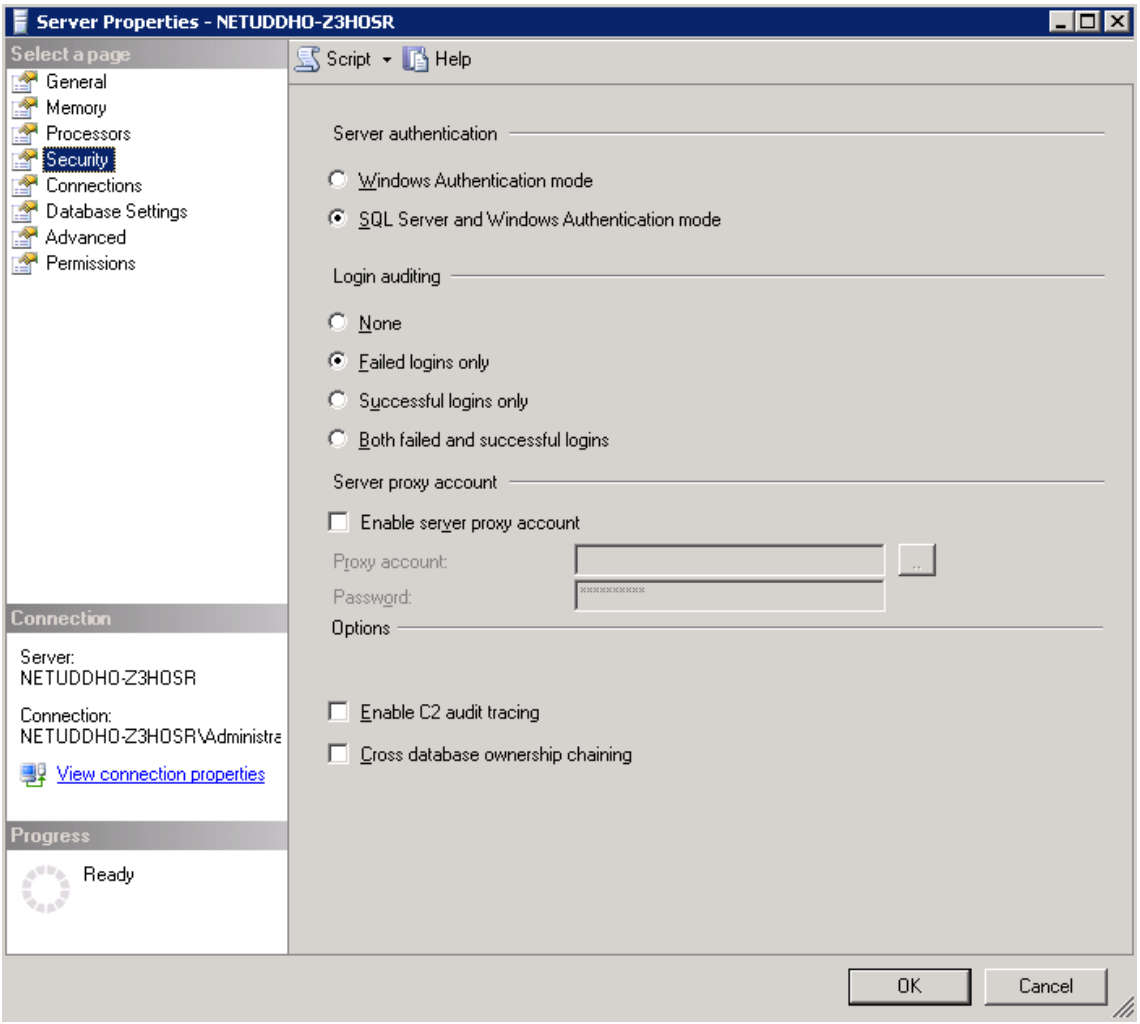

Figure 3.16. Associating database with the new user

Enable remote logins for SQL users.

In the Object Explorer right-click on your database server, and select **Properties > Security**, and set the **Server Authentication** option to **SQL Server and Windows Authentication mode**.

Chapter 4. Configuring syslog-ng

This chapter describes how to configure syslog-ng.

4.1. The syslog-ng configuration file

The syslog-ng application is configured by editing the `syslog-ng.conf` file. Use any regular text editor application to modify the file. The precompiled syslog-ng packages include sample configuration files as well.

Every syslog-ng configuration file must begin with a line containing the version information of syslog-ng. For syslog-ng version 3.1, this line looks like:

```
@version:3.1
```

Versioning the configuration file was introduced in syslog-ng 3.0. If the configuration file does not contain the version information, syslog-ng assumes that the file is for syslog-ng version 2.x. In this case it interprets the configuration and sends warnings about the parts of the configuration that should be updated. Version 3.0 and later will correctly operate with configuration files of version 2.x, but the default values of certain parameters have changed since 3.0.

All identifiers, option names and attributes, and any other strings used in the syslog-ng configuration file are case sensitive. Objects must be defined before they are referenced in another statement.

Example 4.1. A simple configuration file
The following is a very simple configuration file for syslog-ng: it collects the internal messages of syslog-ng and the messages from */dev/log* into the `/var/log/messages_syslog-ng.log` file.

```
@version:3.0

source s_local { unix-stream("/dev/log"); internal(); };

destination d_file {file("/var/log/messages_syslog-ng.log"); };

log { source(s_local); destination(d_file); };
```

Tip
Before activating a new configuration, check that your configuration file is syntactically correct using the `syslog-ng --syntax-only` command.

To activate the configuration, reload the configuration of syslog-ng using the `/etc/init.d/syslog-ng reload` command.

The `syslog-ng.conf` file is located under the `/opt/syslog-ng/etc/` directory.

Note
Earlier versions of syslog-ng OSE stored the configuration file in different directories, depending on the platform; typically under `/etc/syslog-ng/`.

4.1.1. Including configuration files

The syslog-ng application supports including external files in its configuration file, so parts of its configuration can be managed separately. To include the contents of a file in the syslog-ng configuration, use the following syntax

```
include "filename";
```

This imports the entire file into the configuration of syslog-ng, at the location of the include statement. If you specify a directory, syslog-ng will try to include every file in alphabetic order. When including configuration files, consider the following points:

- If an object is defined twice (for example the original syslog-ng configuration file and the file imported into this configuration file both define the same option, source, or other object), then the object that is defined later in the configuration file will be effective. For example, if you set a global option at the beginning of the configuration file, and later include a file that defines the same option with a different value, then the option defined in the imported file will be used.
- Files can be embedded into each other: the included files can contain include statements as well, up to a maximum depth of 15 levels.
- Include statements can only be used at top level of the configuration file. For example, the following is correct:

  ```
  @version:3.0
  include "example.conf";
  ```

  But the following is not:

  ```
  source s_example {
      include "example.conf"
       };
  ```

Warning
The syslog-ng application will not start if it cannot find a file that is to be included in its configuration. Always double-check the filenames, paths, and access rights when including configuration files, and use the `--syntax-only` command-line option to check your configuration.

4.2. Defining global objects

Global objects (for example sources, destinations, log paths, or filters) are defined in the syslog-ng configuration file. Object definitions consist of the following elements:

- *Type of the object*: One of `source`, `destination`, `log`, `filter`, `parser`, `rewrite` rule, or `template`.
- *Identifier of the object*: A unique name identifying the object. When using a reserved word as an identifier, enclose the identifier in quotation marks.

Tip
Use identifiers that refer to the type of the object they identify. For example, prefix source objects with `s_`, destinations with `d_`, and so on.

- *Parameters*: The parameters of the object, enclosed in braces `{parameters}`.
- *Semicolon*: Object definitions end with a semicolon (`;`).

The syntax is summarized as follows:

```
type identifier { parameters };
```

Objects have parameters; some of them are required, others are optional. Required parameters are positional, meaning that they must be specified in a defined order. Optional arguments can be specified in any order using the `option(value)` format. If a parameter (optional or required) is not specified, its default value is used. The parameters and their default values are listed in the reference section of the particular object. See *Chapter 6, Reference (p. 91)* for details.

Example 4.2. Using required and optional parameters
The `unix-stream()` source driver has a single required argument: the name of the socket to listen on. Optional parameters follow the socket name in any order, so the following source definitions have the same effect:

```
source s_demo_stream1 {
          unix-stream("/dev/log" max-connections(10) group(log)); };
source s_demo_stream2 {
          unix-stream("/dev/log" group(log) max-connections(10)); };
```

To add comments to the configuration file, start a line with `#` and write your comments. These lines are ignored by syslog-ng.

```
# Comment: This is a stream source
source s_demo_stream {
          unix-stream("/dev/log" max-connections(10) group(log)); };
```

4.2.1. Notes about the configuration syntax

When you are editing the syslog-ng configuration file, note the following points:

- When writing the names of options and parameters (or other reserved words), the hyphen (`-`) and underscore (`_`) characters are equivalent, for example `max-connections(10)` and `max_connections(10)` are both correct.
- Number can be prefixed with `+` or `-` to indicate positive or negative values. Numbers beginning with zero (`0`) or `0x` are treated as octal or hexadecimal numbers, respectively.
- You can use commas (`,`) to separate options or other parameters for readability; syslog-ng completely ignores them. The following declarations are equivalent:

```
source s_demo_stream {
                          unix-stream("/dev/log" max-connections(10)
group(log)); };
```

```
source s_demo_stream {
                        unix-stream("/dev/log", max-connections(10),
group(log)); };
```

- Strings between single quotes (*'string'*) are treated literally, you do not have to escape special characters. This makes writing and reading regular expressions much more simple: it is recommended to use single quotes when writing regular expressions.
- When enclosing strings between double-quotes (*"string"*), you have to escape special characters: for example when enclosing a regular expression that uses the \ character to escape a special character, you have to add an extra \ (for example *"\\n"*). It is recommended to use single quotes instead.
- Enclosing normal strings between double-quotes (*"string"*) is not necessary, you can just omit the double-quotes. For example when writing filters, *match("sometext")* and *match(sometext)* will both match for the *sometext* string.
- When enclosing object IDs (for example the name of a destination) between double-quotes (*"mydestination"*), the ID can include whitespace as well, for example:

```
source "s demo stream" {
                        unix-stream("/dev/log" max-connections(10)
group(log)); };
```

4.3. Sources and source drivers

A source is where syslog-ng receives log messages. Sources consist of one or more drivers, each defining where and how messages are received.

To define a source, add a source statement to the syslog-ng configuration file using the following syntax:

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

Example 4.3. A simple source statement
The following source statement receives messages on the TCP port *1999* of the interface having the *10.1.2.3* IP address.

```
source s_demo_tcp { tcp(ip(10.1.2.3) port(1999)); };
```

Example 4.4. A source statement using two source drivers
The following source statement receives messages on the *1999* TCP port and the *1999* UDP port of the interface having the *10.1.2.3* IP address.

```
source s_demo_two_drivers {
          tcp(ip(10.1.2.3) port(1999));
          udp(ip(10.1.2.3) port(1999)); };
```

Example 4.5. Setting default priority and facility
If the message received by the source does not have a proper syslog header, you can use the *default-facility()* and *default-priority()* options to set the facility and priority of the messages. Note that these values are applied only to messages that do not set these parameters in their header.

```
source headerless_messages { udp(default-facility(syslog) default-priority(emerg)); };
```

Define a source only once. The same source can be used in several log paths. Duplicating sources causes syslog-ng to open the source (TCP/IP port, file, etc.) more than once, which might cause problems. For example, include the `/dev/log` file source only in one source statement, and use this statement in more than one log path if needed.

To collect log messages on a specific platform, it is important to know how the native `syslogd` communicates on that platform. The following table summarizes the operation methods of `syslogd` on some of the tested platforms:

Platform	Method
Linux	A `SOCK_STREAM` unix socket named `/dev/log`; some of the distributions switched over to using `SOCK_DGRAM`, though applications still work with either 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 earlier versions, 2.6 uses a new multithreaded IPC method called door. By default the door used by `syslogd` is `/etc/.syslog_door`.
HP-UX 11 or later	HP-UX uses a named pipe called `/dev/log` that is padded to 2048 bytes, for example `source s_hp-ux {pipe ("/dev/log" pad_size(2048)};`.
AIX 5.2 and 5.3	A `SOCK_STREAM` or `SOCK_DGRAM` unix socket called `/dev/log`.

Table 4.1. Communication methods used between the applications and syslogd

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

Example 4.6. Source statement on a Linux based operating system
The following source statement collects the following log messages:

- *internal()*: Messages generated by syslog-ng.
- *udp(ip(0.0.0.0) port(514))*: Messages arriving to the `514/UDP` port of any interface of the host.
- *unix-stream("/dev/log");*: Messages arriving to the `/dev/log` socket.

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

The following table lists the source drivers available in syslog-ng.

Name	Description
internal()	Messages generated internally in syslog-ng.
file()	Opens the specified file and reads messages.
pipe(), fifo	Opens the specified named pipe and reads messages.
program()	Opens the specified application and reads messages from its standard output.
sun-stream(), sun-streams()	Opens the specified `STREAMS` device on Solaris systems and reads incoming messages.
syslog()	Listens for incoming messages using the new *IETF-standard syslog protocol*.

Name	Description
tcp(), tcp6()	Listens on the specified TCP port for incoming messages using the *BSD-syslog protocol* over IPv4 and IPv6 networks, respectively.
udp(), udp6()	Listens on the specified UDP port for incoming messages using the *BSD-syslog protocol* over IPv4 and IPv6 networks, respectively.
unix-dgram()	Opens the specified unix socket in `SOCK_DGRAM` mode and listens for incoming messages.
unix-stream()	Opens the specified unix socket in `SOCK_STREAM` mode and listens for incoming messages.

Table 4.2. Source drivers available in syslog-ng

For a complete description of the parameters of the above drivers, see *Section 6.1, Source drivers (p. 91)*.

4.3.1. Collecting internal messages

All messages generated internally by syslog-ng use this special source. To collect warnings, errors and notices from syslog-ng itself, include this source in one of your source statements.

```
internal()
```

The syslog-ng application will issue a warning upon startup if none of the defined log paths reference this driver.

Example 4.7. Using the internal() driver

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

4.3.1.1. Log statistics

Periodically, syslog-ng sends a message containing statistics about the received messages, and about any lost messages since the last such message. It includes a `processed` entry for every source and destination, listing the number of messages received or sent, and a `dropped` entry including the IP address of the server for every destination where syslog-ng has lost messages. The `center(received)` entry shows the total number of messages received from every configured sources.

The following is a sample log statistics message for a configuration that has a single source (`s_local`) and a network and a local file destination (`d_network` and `d_local`, respectively). All incoming messages are sent to both destinations.

```
Log statistics;
                    dropped='tcp(AF_INET(192.168.10.1:514))=6439',
                    processed='center(received)=234413',
                    processed='destination(d_tcp)=234413',
                    processed='destination(d_local)=234413',
                    processed='source(s_local)=234413'
```

Log statistics can be also retrieved on-demand using one of the following options:

- Use the `socat` application: `echo STATS | socat -vv UNIX-CONNECT:/opt/syslog-ng/var/run/syslog-ng.ctl -`
- If you have an OpenBSD-style `netcat` application installed, use the `echo STATS | nc -U var/run/syslog-ng.ctl` command. Note that the `netcat` included in most Linux distributions is a GNU-style version that is not suitable to query the statistics of syslog-ng.
- Starting from syslog-ng Open Source Edition version 3.1, syslog-ng Open Source Edition includes the syslog-ng-ctl utility. Use the `syslog-ng-ctl stats` command.

The statistics include a list of source groups and destinations, as well as the number of processed messages for each. The verbosity of the statistics can be set using the *`stats_level()`* option. See *Section 6.9, Global options (p. 170)* for details. An example output is shown below.

```
src.internal;s_all#0;;a;processed;6445
src.internal;s_all#0;;a;stamp;1268989330
destination;df_auth;;a;processed;404
destination;df_news_dot_notice;;a;processed;0
destination;df_news_dot_err;;a;processed;0
destination;d_ssb;;a;processed;7128
destination;df_uucp;;a;processed;0
source;s_all;;a;processed;7128
destination;df_mail;;a;processed;0
destination;df_user;;a;processed;1
destination;df_daemon;;a;processed;1
destination;df_debug;;a;processed;15
destination;df_messages;;a;processed;54
destination;dp_xconsole;;a;processed;671
dst.tcp;d_network#0;10.50.0.111:514;a;dropped;5080
dst.tcp;d_network#0;10.50.0.111:514;a;processed;7128
dst.tcp;d_network#0;10.50.0.111:514;a;stored;2048
destination;df_syslog;;a;processed;6724
destination;df_facility_dot_warn;;a;processed;0
destination;df_news_dot_crit;;a;processed;0
destination;df_lpr;;a;processed;0
destination;du_all;;a;processed;0
destination;df_facility_dot_info;;a;processed;0
center;;received;a;processed;0
destination;df_kern;;a;processed;70
center;;queued;a;processed;0
destination;df_facility_dot_err;;a;processed;0
```

The statistics are semicolon separated; every line contains statistics for a particular object (for example source, destination, tag, etc.). The statistics have the following fields:

1. The type of the object (for example *`dst.file`*, *`tag`*, *`src.facility`*)
2. The ID of the object used in the syslog-ng configuration file, for example *`d_internal`* or *`source.src_tcp`*. The *`#0`* part means that this is the first destination in the destination group.
3. The instance ID (destination) of the object, for example the filename of a file destination, or the name of the application for a program source or destination.
4. The status of the object. One of the following:

- `a` - active. At the time of quering the statistics, the source or the destination was still alive (it continuously received statistical data).
- `d` - dynamic. Such objects may not be continuously available, for example, like statistics based on the sender's hostname.
- `o` - This object was once active, but stopped receiving messages. (For example a dynamic object may disappear and become orphan.)

5. The type of the statistics:
   - `processed`: The number of messages that successfully reached their destination.
   - `dropped`: The number of dropped messages — syslog-ng OSE could not send the messages to the destination and the output buffer got full, so messages were lost.
   - `stored`: The number of messages stored in the message queue, waiting to be sent to the destination.
   - `suppressed`: The number of suppressed messages (if the `suppress()` feature is enabled).
   - `stamp`: The UNIX timestamp of the last message sent to the destination.
6. The number of such messages.

Note
Note that certain statistics are available only if the `stats-level()` option is set to a higher value.

4.3.2. Collecting messages from text files

Collects log messages from plain-text files, for example from the logfiles of an Apache webserver.

The syslog-ng application notices if a file is renamed or replaced with a new file, so it can correctly follow the file even if logrotation is used. When syslog-ng is restarted, it records the position of the last sent log message, and continues to send messages from this position after the restart.

The file driver has a single required parameter specifying the file to open. For the list of available optional parameters, see *Section 6.1.2, file() (p. 91)*.

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

Example 4.8. Using the file() driver

```
source s_file { file("/var/log/messages"); };
```

The kernel usually sends log messages to a special file (`/dev/kmsg` on BSDs, `/proc/kmsg` on Linux). The `file()` driver reads log messages from such files. The syslog-ng application can periodically check the file for new log messages if the `follow_freq()` option is set.

Note
On Linux, the *klogd* daemon can be used in addition to syslog-ng to read kernel messages and forward them to syslog-ng. *klogd* used to preprocess kernel messages to resolve symbols etc., but as this is deprecated by *ksymoops* there is really no point in running both *klogd* and syslog-ng in parallel. Also note that running two processes reading `/proc/kmsg` at the same time might result in dead-locks.

When using syslog-ng to read messages from the `/proc/kmsg` file, syslog-ng automatically disables the *follow_freq()* parameter to avoid blocking the file.

4.3.3. Collecting messages from named pipes

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

The pipe driver has a single required parameter, specifying the filename of the pipe to open. For the list of available optional parameters, see *Section 6.1.3, pipe() (p. 95)*.

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

Note
As of syslog-ng Open Source Edition 3.0.2, pipes are created automatically. In earlier versions, you had to create the pipe using the `mkfifo(1)` command.

Pipe is very similar to the *file()* driver, but there are a few differences, for example *pipe()* opens its argument in read-write mode, therefore it is not recommended to be used on special files like `/proc/kmsg`.

Warning
It is not recommended to use *pipe()* on anything else than real pipes.

Example 4.9. Using the pipe() driver

```
source s_pipe { pipe("/dev/pipe" pad_size(2048)); };
```

4.3.4. Collecting messages on Sun Solaris

Solaris uses its *STREAMS* framework to send messages to the *syslogd* process. Solaris 2.5.1 and above uses an IPC called *door* in addition to *STREAMS*, to confirm the delivery of a message. The syslog-ng application supports the IPC mechanism via the *door()* option (see below).

Note
The *sun-streams()* driver must be enabled when the syslog-ng application is compiled (see `./configure --help`).

The *sun-streams()* driver has a single required argument specifying the *STREAMS* device to open, and the *door()* option. For the list of available optional parameters, see *Section 6.1.5, sun-streams() driver (p. 102)*.

```
Declaration:
        sun-streams(name_of_the_streams_device door(filename_of_the_door));
```

Example 4.10. Using the sun-streams() driver

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

4.3.5. Collecting messages using the IETF syslog protocol

The *syslog()* driver enables to receive messages from the network using the new standard syslog protocol and message format (also called IETF-syslog protocol; described in RFC 5424-28, see *Section 2.16.2, IETF-syslog messages (p. 22)*). UDP, TCP, and TLS-encrypted TCP can all be used to transport the messages.

For the list of available optional parameters, see *Section 6.1.6, syslog() (p. 105)*.

```
Declaration:
            syslog(ip() port() transport() options());
```

Example 4.11. Using the syslog() driver

TCP source listening on the localhost on port 1999.

```
source s_syslog { syslog(ip(127.0.0.1) port(1999) transport("tcp")); };
```

UDP source with defaults.

```
source s_udp { syslog( transport("udp")); };
```

Encrypted source where the client is also authenticated. See *Section 6.10, TLS options (p. 177)* for details on the encryption settings.

```
source s_syslog_tls{ syslog(
                    ip(10.100.20.40)
                    transport("tls")
                    tls(
                    peer-verify(required-trusted)
                    ca_dir('/opt/syslog-ng/etc/syslog-ng/keys/ca.d/')
                    key_file('/opt/syslog-ng/etc/syslog-ng/keys/server_privatekey.pem')
                   cert_file('/opt/syslog-ng/etc/syslog-ng/keys/server_certificate.pem')

                    )
                    );};
```

4.3.6. Collecting messages from remote hosts using the BSD syslog protocol

The *tcp()*, *tcp6()*, *udp()*, *udp6()* drivers can receive messages from the network using the TCP and UDP networking protocols. The *tcp6()* and *udp6()* drivers use the IPv6 network protocol, while *tcp()* and *udp()* use IPv4.

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 at the protocol level to retransmit such lost messages. The *BSD-syslog* protocol traditionally uses UDP.

TCP provides connection-oriented service, which basically means that the path of the messages is flow-controlled. Along this path, each message is acknowledged, and retransmission is done for lost packets. Generally it is safer to use TCP, because lost connections can be detected, and no messages get lost, assuming that the TCP connection does not break. When a TCP connection is broken the 'in-transit' messages that were sent by syslog-ng but not yet received on the other side are lost. (Basically these messages are still sitting in the socket buffer of the sending host and syslog-ng has no information about the fate of these messages).

The `tcp()` and `udp()` drivers do not have any required parameters. By default they bind to the `0.0.0.0:514` address, which means that syslog-ng will listen on all available interfaces, port 514. To limit accepted connections to only one interface, use the `localip()` parameter as described below. For the list of available optional parameters, see *Section 6.1.7, tcp(), tcp6(), udp() and udp6() (p. 111)*.

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

Note
The tcp port 514 is reserved for use with `rshell`, so select a different port if syslog-ng and `rshell` is used at the same time.

If you specify a multicast bind address to `udp()` and `udp6()`, syslog-ng will automatically join the necessary multicast group. TCP does not support multicasting.

The syslog-ng application supports TLS (Transport Layer Security, also known as SSL) for the tcp() and tcp6() drivers. See the TLS-specific options below and *Section 4.12, Encrypting log messages with TLS (p. 77)* for details. For the list of available optional parameters, see *Section 6.1.7, tcp(), tcp6(), udp() and udp6() (p. 111)*.

Tip
The `syslog()` driver also supports TLS-encrypted connections.

Example 4.12. Using the udp() and tcp() drivers
A simple udp() source with default settings.

```
source s_udp { udp(); };# An UDP source with default settings.
```

A TCP source listening on the localhost interface, with a limited number of connections allowed.

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

A TCP source listening on a TLS-encrypted channel.

```
source s_tcp { tcp(ip(127.0.0.1) port(1999)
                   tls(peer-verify('required-trusted')
                       key_file('/opt/syslog-ng/etc/syslog-ng/syslog-ng.key')
                   cert_file('/opt/syslog-ng/etc/syslog-ng/syslog-ng.crt')));
};
```

A TCP source listening for messages using the IETF-syslog message format. Note that for transferring IETF-syslog messages, generally you are recommended to use the `syslog()` driver on both the client and the server, as it uses both the IETF-syslog message format and the protocol. See *Section 4.3.5, Collecting messages using the IETF syslog protocol (p. 51)* for details.

```
source s_tcp_syslog { tcp(ip(127.0.0.1) port(1999) flags(syslog-protocol) ); };
```

4.3.7. Collecting messages from UNIX domain sockets

The *unix-stream()* and *unix-dgram()* drivers open an *AF_UNIX* socket and start listening on it for messages. The *unix-stream()* driver is primarily used on Linux and uses *SOCK_STREAM* semantics (connection oriented, no messages are lost); while *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 have a single required argument that specifies the filename of the socket to create. For the list of available optional parameters, see *Section 6.1.8, unix-stream() and unix-dgram() (p. 117)*

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

Note
syslogd on Linux originally used *SOCK_STREAM* sockets, but some distributions switched to *SOCK_DGRAM* around 1999 to fix a possible DoS problem. On Linux you can choose to use whichever driver you like as syslog clients automatically detect the socket type being used.

The difference between the unix-stream and unix-dgram drivers is similar to the difference between the TCP and UDP network protocols. Use the following guidelines to select which driver to use in a particular situation:

Choose unix-stream if you would choose TCP (stream) instead of UDP (datagram). The unix-stream driver offers the following features:

- Increased reliability
- Ordered delivery of messages
- Client-side notification of failures

Choose unix-dgram if you would choose TCP (stream) over UDP (datagram). The unix-dgram driver offers the following features:

- Decreased possibility of Dos by opening too many connections (a local vulnerability)
- Less overhead

However, the client does not notice if a message is lost when using the unix-dgram driver.

Example 4.13. 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"); };
```

4.4. Destinations and destination drivers

A destination is where a log message is sent if the filtering rules match. Similarly to sources, destinations consist of one or more drivers, each defining where and how messages are sent.

Tip
If no drivers are defined for a destination, all messages sent to the destination are discarded. This is equivalent to omitting the destination from the log statement.

To define a destination, add a destination statement to the syslog-ng configuration file using the following syntax:

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

Example 4.14. A simple destination statement
The following destination statement sends messages to the TCP port *1999* of the *10.1.2.3* host.

```
destination d_demo_tcp { tcp("10.1.2.3" port(1999)); };
```

If name resolution is configured, the hostname of the target server can be used as well.

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

The following table lists the destination drivers available in syslog-ng.

Name	Description
file()	Writes messages to the specified file.
fifo(), pipe()	Writes messages to the specified named pipe.
program()	Forks and launches the specified program, and sends messages to its standard input.
sql()	Sends messages into an SQL database. In addition to the standard syslog-ng packages, the `sql()` destination requires database-specific packages to be installed. Refer to the section appropriate for your platform in *Chapter 3, Installing syslog-ng (p. 26)*.
syslog()	Sends messages to the specified remote host using the *IETF-syslog protocol*. The IETF standard supports message transport using the UDP, TCP, and TLS networking protocols.
tcp() and tcp6()	Sends messages to the specified TCP port of a remote host using the *BSD-syslog protocol* over IPv4 and IPv6, respectively.
udp() and udp6()	Sends messages to the specified UDP port of a remote host using the *BSD-syslog protocol* over IPv4 and IPv6, respectively.
unix-dgram()	Sends messages to the specified unix socket in `SOCK_DGRAM` style (BSD).
unix-stream()	Sends messages to the specified unix socket in `SOCK_STREAM` style (Linux).

Name	Description
usertty()	Sends messages to the terminal of the specified user, if the user is logged in.

Table 4.3. Destination drivers available in syslog-ng

For detailed list of driver parameters, see *Section 6.2, Destination drivers (p. 122)*.

4.4.1. Storing messages in plain-text files

The file driver is one of the most important destination drivers in syslog-ng. It allows to output messages to the specified text file, or to a set of files.

The destination filename may include macros which get expanded when the message is written, thus a simple `file()` driver may create several files. For more information on available macros see *Section 6.5, Macros (p. 154)*.

If the expanded filename refers to a directory which does not exist, it will be created depending on the `create_dirs()` setting (both global and a per destination option).

The `file()` has a single required parameter that specifies the filename that stores the log messages. For the list of available optional parameters, see *Section 6.2.1, file() (p. 122)*.

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

Example 4.15. Using the file() driver

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

Example 4.16. 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));
};
```

Note
When using the `file()` destination, update the configuration of your log rotation program to rotate these files. Otherwise, the log files can become very large.

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 is closed, and its state is freed.

Exploiting this, a DoS attack can be mounted against the system. If the number of possible destination files and its needed memory is more than the amount available on the syslog-ng server.

The most suspicious macro is `$PROGRAM`, where the number of possible variations is rather high. Do not use the `$PROGRAM` macro in insecure environments.

4.4.2. Sending messages to named pipes

The *pipe()* 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. The filename can include macros. For the list of available optional parameters, see *Section 6.2.2, pipe() (p. 126)*.

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

Warning
As of syslog-ng Open Source Edition 3.0.2, pipes are created automatically. In earlier versions, you had to create the pipe using the mkfifo(1) command.

Example 4.17. Using the pipe() driver

```
destination d_pipe { pipe("/dev/xconsole"); };
```

4.4.3. Sending messages to external applications

The *program()* driver starts an external application or script and sends the log messages to its standard input (*stdin*).

The *program()* driver has a single required parameter, specifying a program name to start. The program is executed with the help of the current shell, so the command may include both file patterns and I/O redirections. For the list of available optional parameters, see *Section 6.2.3, program() (p. 129)*.

```
Declaration:
                program(command_to_run);
```

Note
The syslog-ng application automatically restarts the external program if it exits for reliability reasons. However it is not recommended to launch programs for single messages, because if the message rate is high, launching several instances of an application might overload the system, resulting in Denial of Service.

Note that the message format does not include the priority and facility values by default. To add these values, specify a template for the program destination, as shown in the following example.

Example 4.18. Using the program() destination driver

```
destination d_prog { program("/bin/script" template("<$PRI>$DATE $HOST $MSG\n"); ); };
```

4.4.4. Storing messages in an SQL database

The `sql()` driver sends messages into an SQL database. Currently the Microsoft SQL (MSSQL), MySQL, Oracle, PostgreSQL, and SQLite databases are supported.

The `sql()` driver has the following required parameters:

type

Type:	mssql, mysql, oracle, pgsql, or sqlite3
Default:	n/a

Description: Specifies the type of the database, i.e., the DBI database driver to use. Use the `mssql` option to send logs to an MSSQL database. See the examples of the databases on the following sections for details.

database

Type:	string
Default:	n/a

Description: Name of the database that stores the logs.

table

Type:	string
Default:	n/a

Description: Name of the database table to use (can include macros). When using macros, note that some databases limit the length of table names.

columns

Type:	string list
Default:	"date", "facility", "level", "host", "program", "pid", "message"

Description: Name of the columns storing the data in `fieldname [dbtype]` format. The `[dbtype]` parameter is optional, and specifies the type of the field. By default, syslog-ng creates `text` columns. Note that not every database engine can index text fields.

values

Type:	string list
Default:	"${R_YEAR}-${R_MONTH}-${R_DAY} ${R_HOUR}:${R_MIN}:${R_SEC}", "$FACILITY", "$LEVEL", "$HOST", "$PROGRAM", "$PID", "$MSGONLY"

Description: The parts of the message to store in the fields specified in the `columns` parameter.

For the list of available optional parameters, see *Section 6.2.4, sql() (p. 132).*

```
Declaration:
    sql(database_type host_parameters database_parameters [options]);
```

Warning
The syslog-ng application requires read and write access to the SQL table, otherwise it cannot verify that the destination table exists.

Currently the syslog-ng application has default schemas for the different databases and uses these defaults if the database schema (for example columns and column types) is not defined in the configuration file. However, these schemas will be deprecated and specifying the exact database schema will be required in later versions of syslog-ng.

Note
In addition to the standard syslog-ng packages, the `sql()` destination requires database-specific packages to be installed. These packages are automatically installed by the binary syslog-ng installer.

The `sql()` driver is currently not available for every platform that is supported by syslog-ng. For a list of platforms that support the `sql()` driver, visit *http://www.balabit.com/network-security/syslog-ng/central-syslog-server/*.

The `table` and `value` parameters can include macros to create tables and columns dynamically (see *Section 6.5, Macros (p. 154)* for details).

Warning
When using macros in table names, note that some databases limit the maximum allowed length of table names. Consult the documentation of the database for details.

Inserting the records into the database is performed by a separate thread. The syslog-ng application automatically performs the escaping required to insert the messages into the database.

Example 4.19. Using the sql() driver
The following example sends the log messages into a PostgreSQL database running on the `logserver` host. The messages are inserted into the `logs` database, the name of the table includes the exact date and the name of the host sending the messages. The syslog-ng application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

```
destination d_sql {
  sql(type(pgsql)
  host("logserver") username("syslog-ng") password("password")
  database("logs")
  table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
  columns("datetime", "host", "program", "pid", "message")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid", "message"));
  };
```

The following example specifies the type of the database columns as well:

```
destination d_sql {
  sql(type(pgsql)
  host("logserver") username("syslog-ng") password("password")
  database("logs")
  table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
  columns("datetime varchar(16)", "host varchar(32)", "program  varchar(20)", "pid
varchar(8)", "message  varchar(200)")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid", "message"));
};
```

4.4.4.1. Using the sql() driver with an Oracle database

The Oracle sql destination has some special aspects that are important to note.

- The hostname of the database server is set in the `tnsnames.ora` file, not in the *host* parameter of the *sql()* destination.
  Make sure to set the Oracle-related environment variables properly, so syslog-ng and the Oracle client will find the file. The following variables must be set: *ORACLE_BASE*, *ORACLE_HOME*, and *ORACLE_SID*. See the documentation of the Oracle Instant Client for details.

- As certain database versions limit the maximum length of table names, macros in the table names should be used with care.
- In the current version of syslog-ng PE, the types of database columns must be explicitly set for the Oracle destination. The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng, therefore it is usually recommended to use the *varchar2* or *clob* column type. (The maximum length of the messages can be set using the *log_msg_size()* option.) See the following example for details.

Example 4.20. Using the sql() driver with an Oracle database
The following example sends the log messages into an Oracle database running on the *logserver* host, which must be set in the `/etc/tnsnames.ora` file. The messages are inserted into the *LOGS* database, the name of the table includes the exact date when the messages were sent. The syslog-ng application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

```
destination d_sql {
  sql(type(oracle)
  username("syslog-ng") password("password")
  database("LOGS")
  table("msgs_${R_YEAR}${R_MONTH}${R_DAY}")
  columns("datetime varchar(16)", "host varchar(32)", "program varchar(32)", "pid
varchar(8)", "message varchar2")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid", "message"));
  };
```

The Oracle Instant Client retrieves the address of the database server from the `/etc/tnsnames.ora` file. Edit or create this file as needed for your configuration. A sample is provided below.

```
LOGS =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)
(HOST = logserver)
(PORT = 1521))
)
(CONNECT_DATA =
(SERVICE_NAME = EXAMPLE.SERVICE)
)
)
```

4.4.4.2. Using the sql() driver with a Microsoft SQL database

The *mssql* database driver can access Microsoft SQL (MSSQL) destinations. This driver has some special aspects that are important to note.

- The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the syslog-ng server. See the following example for details.

- As certain database versions limit the maximum length of table names, macros in the table names should be used with care.
- In the current version of syslog-ng PE, the types of database columns must be explicitly set for the MSSQL destination. The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng. The `varchar` column type can store maximum 4096 bytes-long messages. The maximum length of the messages can be set using the `log_msg_size()` option. See the following example for details.
- Remote access for SQL users must be explicitly enabled on the Microsoft Windows host running the Microsoft SQL Server. See *Section 3.6, Configuring Microsoft SQL Server to accept logs from syslog-ng (p. 36)* for details.

Example 4.21. Using the sql() driver with an MSSQL database
The following example sends the log messages into an MSSQL database running on the `logserver` host. The messages are inserted into the `syslogng` database, the name of the table includes the exact date when the messages were sent. The syslog-ng application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

```
destination d_mssql {
sql(type(mssql) host("logserver") port("1433")
  username("syslogng") password("syslogng") database("syslogng")
  table("msgs_${R_YEAR}${R_MONTH}${R_DAY}")columns("datetime varchar(16)", "host
varchar(32)",
  "program varchar(32)", "pid varchar(8)", "message varchar(4096)")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid"));
  };
```

The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the syslog-ng server. Edit or create this file as needed for your configuration. A sample is provided below.

```
[default]
date = "%Y-%m-%d %H:%M:%S"
```

4.4.5. Sending messages to a remote logserver using the IETF-syslog protocol

The `syslog()` driver sends messages to a remote host (for example a syslog-ng server or relay) on the local intranet or internet using the new standard syslog protocol developed by IETF (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details about the new protocol). The protocol supports sending messages using the UDP, TCP, or the encrypted TLS networking protocols.

The required arguments of the driver are the address of the destination host (where messages should be sent). The transport method (networking protocol) is optional, syslog-ng uses the TCP protocol by default. For the list of available optional parameters, see *Section 6.2.5, syslog() (p. 136)*.

```
Declaration:
                syslog(host transport [options]);
```

Note
Note that the `syslog` destination driver has required parameters, while the source driver defaults to the local bind address, and every parameter is optional.

The *udp* transport method automatically sends multicast packets if a multicast destination address is specified. The *tcp* and *tls* methods do not support multicasting.

Note
The default ports for the different transport protocols are as follows: UDP — 514; TLS — 6514.

Example 4.22. Using the syslog() driver

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

If name resolution is configured, the hostname of the target server can be used as well.

```
destination d_tcp { syslog(ip("target_host") transport("tcp") port(1999) localport(999));
 };
```

Send the log messages using TLS encryption and use mutual authentication. See *Section 6.10, TLS options (p. 177)* for details on the encryption and authentication options.

```
destination d_syslog_tls{
                syslog("10.100.20.40"
                transport("tls")
                port(6514)
                tls(peer-verify(required-trusted)
                ca_dir('/opt/syslog-ng/etc/syslog-ng/keys/ca.d/')
                key_file('/opt/syslog-ng/etc/syslog-ng/keys/client_key.pem')
                cert_file('/opt/syslog-ng/etc/syslog-ng/keys/client_certificate.pem'))
                );};
```

4.4.6. Sending messages to a remote logserver using the legacy BSD-syslog protocol

The *tcp()*, *tcp6()*, *udp()*, and *udp6()* drivers send messages to another host (for example a syslog-ng server or relay) on the local intranet or internet using the UDP or TCP protocol. The *tcp6()* and *udp6()* drivers use the IPv6 network protocol.

All four drivers have a single required parameter specifying the destination host address, where messages should be sent. For the list of available optional parameters, see *Section 6.2.6, tcp(), tcp6(), udp(), and udp6() (p. 141)*.

The *udp()* and *udp6()* drivers automatically send multicast packets if a multicast destination address is specified. The *tcp()* and *tcp6()* drivers do not support multicasting.

```
Declaration:
                tcp(host [options]);
                udp(host [options]);
                tcp6(host [options]);
                udp6(host [options]);
```

Example 4.23. Using the tcp() driver

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

If name resolution is configured, the hostname of the target server can be used as well.

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

To send messages using the IETF-syslog message format without using the IETF-syslog protocol, enable the *syslog-protocol* flag:

```
destination d_tcp { tcp("10.1.2.3" port(1999) flags(syslog-protocol) ); };
```

(To use the IETF-syslog protocol, see *Section 6.2.5, syslog() (p. 136)*.)

4.4.7. Sending messages to UNIX domain sockets

The `unix-stream()` and `unix-dgram()` drivers send messages to a UNIX domain 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. For the list of available optional parameters, see *Section 6.2.7, unix-stream() & unix-dgram() (p. 146)*.

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

Example 4.24. Using the unix-stream() driver

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

4.4.8. 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. Use the asterisk `*` to specify every user currently logged in to the system.

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

The `usertty()` does not have any further options nor does it support templates.

Example 4.25. Using the usertty() driver

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

4.5. Log paths

Log paths determine what happens with the incoming log messages. Messages coming from the sources listed in the log statement and matching all the filters are sent to the listed destinations.

To define a log path, add a log statement to the syslog-ng configuration file using the following syntax:

```
log {
    source(s1); source(s2); ...
    optional_element(filter1|parser1|rewrite1);
```

```
optional_element(filter2|parser2|rewrite2);...
    destination(d1); destination(d2); ...
    flags(flag1[, flag2...]);
    };
```

Warning
Log statements are processed in the order they appear in the configuration file, thus the order of log paths may influence what happens to a message, especially when using filters and log flags.

Example 4.26. A simple log statement
The following log statement sends all messages arriving to the localhost to a remote server.

```
source s_localhost { tcp(ip(127.0.0.1) port(1999) ); };
destination d_tcp { tcp("10.1.2.3" port(1999); localport(999)); };
log { source(s_localhost); destination(d_tcp); };
```

All matching log statements are processed by default, and the messages are sent to *every* matching destination by default. So a single log message might be sent to the same destination several times, provided the destination is listed in several log statements, and it can be also sent to several different destinations.

This default behavior can be changed using the `flags()` parameter. Flags apply to individual log paths; they are not global options. The following flags available in syslog-ng:

- *final*: Do not send the messages processed by this log path to any further destination.
- *fallback*: Process messages that were not processed by other log paths.
- *catchall*: Process every message, regardless of its source or if it was already processed by other log paths.
- *flow-control*: Stop reading messages from the source if the destination cannot accept them. See *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16)*.

Warning
The `final`, `fallback`, and `catchall` flags apply only for the top-level log paths, they have no effect on embedded log paths.

Example 4.27. Using log path flags
Let's suppose that you have two hosts (`myhost_A` and `myhost_B`) that run two applications each (`application_A` and `application_B`), and you collect the log messages to a central syslog-ng server. On the server, you create two log paths:

- one that processes only the messages sent by `myhost_A`; and
- one that processes only the messages sent by `application_A`.

This means that messages sent by `application_A` running on `myhost_A` will be processed by both log paths, and the messages of `application_B` running on `myhost_B` will not be processed at all.

- If you add the `final` flag to the first log path, then only this log path will process the messages of `myhost_A`, so the second log path will receive only the messages of `application_A` running on `myhost_B`.
- If you create a third log path that includes the `fallback` flag, it will process the messages not processed by the first two log paths, in this case, the messages of `application_B` running on `myhost_B`.
- Adding a fourth log path with the `catchall` flag would process every message received by the syslog-ng server.

```
log { source(s_localhost); destination(d_file); flags(catchall); };
```

For details on the individual flags, see *Section 6.3, Log path flags (p. 150)*. The effect and use of the `flow-control` flag is detailed in *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16)*.

4.5.1. Using embedded log statements

Embedded log statements (see *Section 2.2.2, Embedded log statements (p. 6)*) re-use the results of processing messages (for example the results of filtering or rewriting) to create complex log paths. Embedded log statements use the same syntax as regular log statements, but they cannot contain additional sources. To define embedded log statements, use the following syntax:

```
log {
    source(s1); source(s2); ...

    optional_element(filter1|parser1|rewrite1);
    optional_element(filter2|parser2|rewrite2);...

    destination(d1); destination(d2); ...

    #embedded log statement
    log
        {
        optional_element(filter1|parser1|rewrite1);
        optional_element(filter2|parser2|rewrite2);...
        destination(d1); destination(d2); ...
        #another embedded log statement
        log
        {
            optional_element(filter1|parser1|rewrite1);
            optional_element(filter2|parser2|rewrite2);...
            destination(d1); destination(d2); ...};
        };
    #set flags after the embedded log statements
    flags(flag1[, flag2...]);
        };
```

Warning
The `final`, `fallback`, and `catchall` flags apply only for the top-level log paths, they have no effect on embedded log paths.

Example 4.28. Using embedded log paths
The following log path sends every message to the `d_file1` and the `d_file2` destinations.

```
log { source(s_localhost); destination(d_file1); destination(d_file2); };
```

The next example is equivalent with the one above, but uses an embedded log statement.

```
log { source(s_localhost); destination(d_file1);
                    log {destination(d_file2); };
};
```

The following example sends every message coming from the host `192.168.1.1` into the `d_file1` destination, and sends every message coming from the host `192.168.1.1` and containing the string `example` into the `d_file2` destination.

```
log { source(s_localhost); host(192.168.1.); destination(d_file1);
                    log {message("example"); destination(d_file2); };
};
```

The following example collects logs from multiple source groups and uses the `source()` filter in the embedded log statement to select messages of the `s_network` source group.

```
log { source(s_localhost); source(s_network); destination(d_file1);
                    log {source(s_network); destination(d_file2); };
                    };
```

4.5.2. Configuring flow-control

For details on how flow-control works, see *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16)*. The summary of the main points is as follows:

- The syslog-ng application normally reads a maximum of `log_fetch_limit()` number of messages from a source.
- From TCP and unix-stream sources, syslog-ng reads a maximum of `log_fetch_limit()` from every connection of the source. The number of connections to the source is set using the `max_connections()` parameter.
- Every destination has an output buffer (`log_fifo_size()`).
- Flow-control uses a control window to determine if there is free space in the output buffer for new messages. Every source has its own control window; `log_iw_size()` parameter sets the size of the control window.
- When a source accepts multiple connections, the messages of every connection use the same control window.
- The output buffer must be larger than the control window of every source that logs to the destination.
- If the control window is full, syslog-ng stops reading messages from the source until some messages are successfully sent to the destination.
- If the output buffer becomes full, and neither disk-buffering nor flow-control is used, messages may be lost.

Note
If you modify the `max_connections()` or the `log_fetch_limit()` parameter, do not forget to adjust the `log_iw_size()` and `log_fifo_size()` parameters accordingly.

Example 4.29. Sizing parameters for flow-control
Suppose that syslog-ng has a source that must accept up to 300 parallel connections. Such situation can arise when a network source receives connections from many clients, or if many applications log to the same socket. Therefore, set the `max_connections()` parameter of the source to `300`. However, the `log_fetch_limit()` (default value: 10) parameter applies to every connection of the source individually, while the `log_iw_size()` (default value: 100) parameter applies to the source. In a worst-case scenario, the destination does not accept any messages, while all 300 connections send at least `log_fetch_limit()` number of messages to the source during every poll loop. Therefore, the control window must accommodate at least `max_connections()*log_fetch_limit()` messages to be able to read every incoming message of a poll loop. In the current example this means that (`log_iw_size()` should be greater than `300*10=3000`. If the control

window is smaller than this value, the control window might fill up with messages from the first connections — causing syslog-ng to read only one message of the last connections in every poll loop.

The output buffer of the destination must accommodate at least `log_iw_size()` messages, but use a greater value: in the current example `3000*10=30000` messages. That way all incoming messages of ten poll loops fit in the output buffer. If the output buffer is full, syslog-ng does not read any messages from the source until some messages are successfully sent to the destination.

```
source s_localhost {
            tcp(ip(127.0.0.1) port(1999) max-connections(300)); };
            destination d_tcp {
            tcp("10.1.2.3" port(1999); localport(999)); log_fifo_size(30000); };
            log { source(s_localhost); destination(d_tcp); flags(flow-control); };
```

If other sources send messages to this destination, than the output buffer must be further increased. For example, if a network host with maximum `100` connections also logs into the destination, than increase the `log_fifo_size()` by `10000`.

```
source s_localhost {
            tcp(ip(127.0.0.1) port(1999) max-connections(300)); };
            source s_tcp {
            tcp(ip(192.168.1.5) port(1999) max-connections(100)); };
            destination d_tcp {
            tcp("10.1.2.3" port(1999); localport(999)); log_fifo_size(40000); };
            log { source(s_localhost); destination(d_tcp); flags(flow-control); };
```

See also *Section 5.2, Handling lots of parallel connections (p. 86)*.

4.6. Filters

The following sections describe how to select and filter log messages.

- *Section 4.6.1, Using filters (p. 66)* describes how to configure and use filters.
- *Section 4.6.2, Optimizing regular expressions in filters (p. 68)* provides tips on using regular expressions.
- *Section 4.6.3, Tagging messages (p. 69)* explains how to tag messages and how to filter on the tags.

4.6.1. Using filters

Filters perform log routing within syslog-ng: a message passes the filter if the filter expression is true for the particular message. If a log statement includes filters, the messages are sent to the destinations only if they pass all filters of the log path. For example, a filter can select only the messages originating from a particular host. Complex filters can be created using filter functions and logical boolean expressions.

To define a filter, add a filter statement to the syslog-ng configuration file using the following syntax:

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

The expression may contain the following elements:

- The functions listed in *Section facility() (p. 151)*. Some of the functions accept extended regular expressions as parameters.
- The boolean operators `and`, `or`, `not`.
- Parentheses to mark the precedence of the operators when using complex filters.

Example 4.30. A simple filter statement
The following filter statement selects the messages that contain the word *deny* and come from the host *example*.

```
filter demo_filter { host("example") and match("deny" value("MESSAGE")); };
```

For the filter to have effect, include it in a log statement:

```
log demo_filteredlog {
        source(s1); source(s2);
        filter(demo_filter);
        destination(d1); destination(d2); };
```

The *host()*, *match()*, and *program()* filter functions accept regular expressions as parameters.

```
filter demo_regexp_filter { host("system.*1") and match("deny" value("MESSAGE")); };
```

The *value()* parameter limits the scope of a filter function to the scope of a macro. For example, to limit the scope of the *match()* filter to the text part of the message, use:

```
match("keyword" value("MESSAGE"))
```

The *value()* parameter accepts both built-in macros and user-defined ones created with a parser. Do not prefix the macros with the $ sign. For details on macros and parsers, see *Section 4.7, Templates and macros (p. 70)* and *Section 4.8, Parsing messages (p. 71)*.

Note
When a log statement includes multiple filter statements, syslog-ng sends a message to the destination only if all filters are true for the message. In other words, the filters are connected with the logical *AND* operator. In the following example, no message arrives to the destination, because the filters are exclusive (the hostname of a client cannot be *example1* and *example2* at the same time):

```
filter demo_filter1 { host("example1"); };
filter demo_filter2 { host("example2"); };

log demo_filteredlog {
 source(s1); source(s2);
 filter(demo_filter1); filter(demo_filter2);
 destination(d1); destination(d2); };
```

To select the messages that come from either host *example1* or *example2*, use a single filter expression:

```
filter demo_filter { host("example1") or host("example2"); };

log demo_filteredlog {
 source(s1); source(s2);
 filter(demo_filter);
 destination(d1); destination(d2); };
```

Use the *not* operator to invert filters, for example, to select the messages that were not sent by host *example1*:

```
filter demo_filter { not host("example1"); };
```

However, to select the messages that were not sent by host *example1* or *example2*, you have to use the *and* operator (that's how boolean logic works):

```
filter demo_filter { not host("example1") and not host("example2"); };
```

Alternatively, you can use parentheses to avoid this confusion:

```
filter demo_filter { not (host("example1") or host("example2")); };
```

In the extended regular expressions, the characters *()[].*?+^$|* are used as special symbols. Therefore, these characters have to be preceded with a backslash (\) if they are meant literally. For example, the *\$40* expression

matches the `$40` string. Backslashes have to be escaped as well if they are meant literally. For example, the `\\d` expression matches the `\d` string.

Tip
If you use single quotes in, you do not need to escape the backslash, for example `match("\\.")` is equivalent to `match('\.')`.

By default, all regular expressions are case sensitive. To disable the case sensitivity of the expression, add the `flags(ignore-case)` option to the regular expression.

```
filter demo_regexp_insensitive { host("system" flags(ignore-case)); };
```

For details on regular expressions, see *Section 6.8, Regular expressions (p. 168)*.

Note
In regular expressions, the asterisk (`*`) character means 0, 1 or any number of the previous expression. For example, in the `f*ilter` expression the asterisk means 0 or more f letters. This expression matches for the following strings: `ilter`, `filter`, `ffilter`, etc. To achieve the wildcard functionality commonly represented by the asterisk character in other applications, use `.*` in your expressions, for example `f.*ilter`.

The `level()` filter can select messages corresponding to a single importance level, or a level-range. To select messages of a specific level, use the name of the level as a filter parameter, for example use the following to select warning messages:

```
level(warning)
```

To select a range of levels, include the beginning and the ending level in the filter, separated with two dots (`..`). For example, to select every message of error or higher level, use the following filter:

```
level(err..emerg)
```

Similarly, messages sent by a range of facilities can also be selected. Note that this is only possible when using the name of the facilities. It is not possible to select ranges the numerical codes of the facilities.

```
facility(local0..local5)
```

For a complete list of the available levels and facilities, see *Section 6.4, Filter functions (p. 151)*.

For a complete description on the above functions, see *Section 6.4, Filter functions (p. 151)*.

4.6.2. Optimizing regular expressions in filters

Some filter functions accept regular expressions as parameters. But evaluating general regular expressions puts a high load on the CPU, which can cause problems when the message traffic is very high. Often the regular expression can be replaced with simple filter functions and logical operators. Using simple filters and logical operators, the same effect can be achieved at a much lower CPU load.

Example 4.31. Optimizing regular expressions in filters
Suppose you need a filter that matches the following error message logged by the *xntpd* NTP daemon:

```
xntpd[1567]: time error -1159.777379 is too large (set clock manually);
```

The following filter uses regular expressions and matches every instance and variant of this message.

```
filter f_demo_regexp {
    program("demo_program") and
    match("time error .* is too large .* set clock manually"); };
```

Segmenting the *match()* part of this filter into separate *match()* functions greatly improves the performance of the filter.

```
filter f_demo_optimized_regexp {
    program("demo_program") and
    match("time error") and
    match("is too large") and
    match("set clock manually"); };
```

4.6.3. Tagging messages

Starting with syslog-ng 3.1, it is also possible to label the messages with custom tags. Tags are simple labels, identified by their names, which must be unique. Currently syslog-ng can tag a message at two different places:

- at the source when the message is received; and
- when the message matches a pattern in the pattern database. For details on using the pattern database, see *Section 4.9, Classifying messages (p. 72)*, for details on creating tags in the pattern database, see *Section 6.6.2.3, Creating pattern databases (p. 163)*.

When syslog-ng receives a message, it automatically adds the *.source.<id_of_the_source_statement>* tag to the message. Use the *tags()* option of the source to add custom tags, and the *tags()* option of the filters to select only specific messages.

Note

- Tagging messages and also filtering on the tags is very fast, much faster then other types of filters.
- Tags are available locally, that is, if you add tags to a message on the client, these tags will not be available on the server.
- To include the tags in the message, use the *$TAGS* macro in a template. Alternatively, if you are using the IETF-syslog message format, you can include the *$TAGS* macro in the *.SDATA.meta* part of the message. Note that the *$TAGS* macro is available only in syslog-ng OSE 3.1.1 and later.

Example 4.32. Adding tags and filtering messages with tags

```
source s_tcp {
tcp(ip(192.168.1.1) port(1514) tags("tcp", "router"));
};
```

Use the *tags()* option of the filters to select only specific messages:

```
filter f_tcp {
   tags(".source.s_tcp");
};

filter f_router {
   tags("router");
};
```

4.7. Templates and macros

The syslog-ng application allows you to define message templates, and reference them from every object that can use a template. Templates can be used to create standard message formats or filenames. Templates can reference one or more macros (for example date, the hostname, etc.). See *Section 6.5, Macros (p. 154)* for a list of macros available in syslog-ng Open Source Edition. Fields from the structured data (SD) part of messages using the new IETF-syslog standard can also be used as macros.

Template objects have a single option called `template_escape`, which is disabled by default (`template_escape(no)`). This behavior is useful when the messages are passed to an application that cannot handle escaped characters properly. Enabling template escaping (`template_escape(yes)`) causes syslog-ng to escape the `'` and `"` characters from the messages.

Note
In versions 2.1 and earlier, the `template_escape()` option was enabled by default.

Macros can be included by prefixing the macro name with a `$` sign, just like in Bourne compatible shells. Regarding braces around macro names, the following two formats are equivalent `"$MSG"` and `"${MSG}"`.

Default values for macros can also be specified by appending the `:-` characters and the default value to the macro, for example

```
${HOST:-default_hostname}
```

The macros related to the date of the message (for example: `ISODATE`, `HOUR`, etc.) have two further versions each: one with the `S_` and one with the `R_` prefix (for example: `S_DATE` and `R_DATE`). The `S_DATE` macro represents the date found in the log message, i.e. when the message was sent by the original application. `R_DATE` is the date when syslog has received the message.

`DATE` equals either `S_DATE` or `R_DATE`, depending on the global option set in the now deprecated `use_time_recvd()` parameter (see *Section 6.9, Global options (p. 170)*).

Warning
The hostname-related macros (`FULLHOST`, `FULLHOST_FROM`, `HOST`, and `HOST_FROM`) do not have any effect if the `keep_hostname()` option is disabled.

By default, syslog-ng sends messages using the following template: `$ISODATE $HOST $MSGHDR$MSG\n`. (The `$MSGHDR$MSG` part is written together because the `$MSGHDR` macro includes a trailing whitespace.)

Note
Earlier versions of syslog-ng used templates and scripts to send log messages into SQL databases. Starting from version 2.1, syslog-ng natively supports direct database access using the `sql()` destination. See *Section 6.2.4, sql() (p. 132)* for details.

Example 4.33. Using templates
The following template (*t_demo_filetemplate*) adds the date of the message and the name of the host sending the message to the beginning of the message text. The template is then used in a file destination: messages sent to this destination (*d_file*) will use the message format defined in the template.

```
template t_demo_filetemplate {
            template("$ISODATE $HOST $MSG\n"); template_escape(no); };
destination d_file {
            file("/var/log/messages" template(t_demo_filetemplate)); };
```

Templates can also be used inline, if they are used only at a single location. The following destination is equivalent with the previous example:

```
destination d_file {
               file ("/var/log/messages"
                       template("$ISODATE $HOST $MSG\n") template_escape(no) );
               };
```

4.8. Parsing messages

The syslog-ng application can separate parts of log messages (i.e., the contents of the $MSG macro) to named fields (columns). These fields act as user-defined macros that can be referenced in message templates, file- and tablenames, etc.

Parsers are similar to filters: they must be defined in the syslog-ng configuration file and used in the log statement.

Note
The order of filters, rewriting rules, and parsers in the log statement is important, as they are processed sequentially.

To create a parser, define the columns of the message, the delimiter or separator characters, and optionally the characters that are used to escape the delimiter characters (quote-pairs). For the list of parser parameters, see *Section 6.6, Message parsers (p. 158)*.

```
Declaration:
            parser parser_name {
            csv-parser(column1, column2, ...)
            delimiters()
            quote-pairs()
            };
```

Column names work like macros. Always use a prefix to identify the columns of the parsers, for example *MYPARSER1.COLUMN1, MYPARSER2.COLUMN2*, etc. Column names starting with a dot (for example *.HOST*) are reserved for use by syslog-ng.

Example 4.34. Segmenting hostnames separated with a dash
The following example separates hostnames like *example-1* and *example-2* into two parts.

```
parser p_hostname_segmentation {
    csv-parser(columns("HOSTNAME.NAME", "HOSTNAME.ID")
    delimiters("-")
    flags(escape-none)
    template("${HOST}"));
};
```

```
destination d_file { file("/var/log/messages-${HOSTNAME.NAME:-examplehost}"); };
log { source(s_local); parser(p_hostname_segmentation); destination(d_file);};
```

Example 4.35. Parsing Apache log files
The following parser processes the log of Apache web servers and separates them into different fields. Apache log messages can be formatted like:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %T %v"
```

Here is a sample message:

```
192.168.1.1 - - [31/Dec/2007:00:17:10 +0100] "GET /cgi-bin/example.cgi HTTP/1.1" 200 2708
 "-" "curl/7.15.5 (i4 86-pc-linux-gnu) libcurl/7.15.5 OpenSSL/0.9.8c zlib/1.2.3 libidn/0.6.5"
 2 example.balabit
```

To parse such logs, the delimiter character is set to a single whitespace (*delimiters(" ")*). Whitespaces between quotes and brackets are ignored (*quote-pairs('""[]')*).

```
parser p_apache {
    csv-parser(columns("APACHE.CLIENT_IP", "APACHE.IDENT_NAME", "APACHE.USER_NAME",
        "APACHE.TIMESTAMP", "APACHE.REQUEST_URL", "APACHE.REQUEST_STATUS",
        "APACHE.CONTENT_LENGTH", "APACHE.REFERER", "APACHE.USER_AGENT",
        "APACHE.PROCESS_TIME", "APACHE.SERVER_NAME")
         flags(escape-double-char,strip-whitespace)
         delimiters(" ")
         quote-pairs('""[]')
         );
};
```

The results can be used for example to separate log messages into different files based on the APACHE.USER_NAME field. If the field is empty, the *nouser* name is assigned.

```
log { source(s_local);
    parser(p_apache); destination(d_file);};
};
destination d_file { file("/var/log/messages-${APACHE.USER_NAME:-nouser}"); };
```

Multiple parsers can be used to split a part of an already parsed message into further segments.

Example 4.36. Segmenting a part of a message
The following example splits the timestamp of a parsed Apache log message into separate fields.

```
parser p_apache_timestamp {
    csv-parser(columns("APACHE.TIMESTAMP.DAY", "APACHE.TIMESTAMP.MONTH",
"APACHE.TIMESTAMP.YEAR", "APACHE.TIMESTAMP.HOUR", "APACHE.TIMESTAMP.MIN",
"APACHE.TIMESTAMP.MIN", "APACHE.TIMESTAMP.ZONE")
    delimiters("/: ")
    flags(escape-none)
    template("${APACHE.TIMESTAMP}"));
    };
log { source(s_local);
    log { parser(p_apache); parser(p_apache_timestamp); destination(d_file);};
};
```

4.9. Classifying messages

To classify messages using a pattern database, include a *db_parser()* statement in your syslog-ng configuration file using the following syntax:

```
Declaration:
        parser <identifier> {db_parser(file("<database_filename>"));};
```

Note that using the parser in a log statement only performs the classification, but does not automatically do anything with the results of the classification.

Example 4.37. Defining pattern databases
The following statement uses the database located at `/opt/syslog-ng/var/db/patterndb.xml`.

```
parser pattern_db {
            db_parser(
                file("/opt/syslog-ng/var/db/patterndb.xml")
            );
            };
```

To apply the patterns on the incoming messages, include the parser in a log statement:

```
log {
            source(s_all);
            parser(pattern_db);
            destination( di_messages_class);
            };
```

Note
The default location of the pattern database file is `/opt/syslog-ng/var/run/patterndb.xml`. The *`file`* option of the *`db-parser`* statement can be used to specify a different file, thus different db-parser statements can use different pattern databases. Later versions of syslog-ng will be able to dynamically generate a main database from separate pattern database files.

Example 4.38. Using classification results
The following destination separates the log messages into different files based on the class assigned to the pattern that matches the message (for example Violation and Security type messages are stored in a separate file), and also adds the ID of the matching rule to the message:

```
destination di_messages_class {
        file("/var/log/messages-${.classifier.class}"

template("${.classifier.rule_id};${S_UNIXTIME};${SOURCEIP};${HOST};${PROGRAM};${PID};${MSG}\n")

        template_escape(no)
    );
};
```

To create your own pattern databases see *Section 6.6.2.3, Creating pattern databases (p. 163)*.

4.9.1. Downloading sample pattern databases

Sample pattern databases are available at the BalaBit Download page *http://www.balabit.com/downloads/files/patterndb-snapshot/*. Note that even though these pattern databases contain over 8000 rules for more than 200 applications and devices, they are only samples and experimental databases that are not officially supported and may or may not work in your environment.

The syslog-ng pattern databases are available under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 (CC by-NC-SA) license. This includes every pattern database written by community contributors or the BalaBit staff. It means that:

- you are free to use and modify the patterns for noncommercial purposes;
- when redistributing the pattern databases you must distribute your modifications under the same license;

- and when redistributing the pattern databases, you must make it obvious that the original syslog-ng pattern databases are available at *http://www.balabit.com/network-security/syslog-ng/*.

For legal details, the full text of the license is available at *http://creativecommons.org/licenses/by-nc-sa/3.0/us/legalcode*.

4.9.2. Using parser results in filters and templates

4.9.2.1. Filtering messages based on classification

The results of message classification and parsing can be used in custom filters and file and database templates as well. There are two built-in macros in syslog-ng OSE that allow you to use the results of the classification: the `.classifier.class` macro contains the class assigned to the message (for example violation, security, or unknown), while the `.classifier.rule_id` macro contains the identifier of the message pattern that matched the message.

Example 4.39. Using classification results for filtering messages
To filter on a specific message class, create a filter that checks the **.classifier_class** macro, and use this filter in a log statement.

```
filter fi_class_violation {
                        match("violation"
                        value(".classifier.class")
                        type("string")
                        );
                        };
```

```
log {
                        source(s_all);
                        parser(pattern_db);
                        filter(fi_class_violation);
                        destination(di_class_violation);
                        };
```

Filtering on the *unknown* class selects messages that did not match any rule of the pattern database. Routing these messages into a separate file allows you to periodically review new or unknown messages.

To filter on messages matching a specific classification rule, create a filter that checks the **.classifier_rule_id** macro. The unique identifier of the rule (for example *e1e9c0d8-13bb-11de-8293-000c2922ed0a*) is the *id* attribute of the rule in the XML database.

```
filter fi_class_rule {
                        match("e1e9c0d8-13bb-11de-8293-000c2922ed0a"
                        value(".classifier_rule_id")
                        type("string")
                        );
                        };
```

The message-segments parsed by the pattern parsers can also be used as macros as well. To accomplish this, you have to add a name to the parser, and then you can use this name as a macro that refers to the parsed value of the message.

Example 4.40. Using pattern parsers as macros
For example, you want to parse messages of an application that look like *"Transaction: <type>."*, where <type> is a string that has different values (for example refused, accepted, incomplete, etc.). To parse these messages, you can use the following pattern:

```
'Transaction: @ESTRING::.@'
```

Here the @ESTRING@ parser parses the message until the next full stop character. To use the results in a filter or a filename template, include a name in the parser of the pattern, for example:

```
'Transaction: @ESTRING:TRANSACTIONTYPE:.@'
```

After that, add a custom template to the logpath that uses this template. For example, to select every *accepted* transaction, use the following custom filter in the log path:

```
match("accepted" value("TRANSACTIONTYPE"));
```

Note
The above macros can be used in database columns and filename templates as well, if you create custom templates for the destination or logspace.

Use a consistent naming scheme for your macros, for example, *APPLICATIONNAME_MACRONAME*.

4.10. Rewriting messages

The syslog-ng application can rewrite parts of log messages: it can search and replace text, and also set a specific field to a specified value. Rewriting messages is often used in conjunction with message parsing *Section 4.8, Parsing messages (p. 71)*.

Rewrite rules are similar to filters: they must be defined in the syslog-ng configuration file and used in the log statement.

Note
The order of filters, rewriting rules, and parsers in the log statement is important, as they are processed sequentially.

To create replace a part of the log message, define the string or regular expression to replace, the string to replace the original text (macros can be used as well), and the field of the message that the rewrite rule should process. Substitution rules can operate on any value available via macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (see *Section 6.6, Message parsers (p. 158)* for details). The only exceptions are the FACILITY, SEVERITY, TAGS, and the date-related fields, which cannot be rewritten. Substitution rules use the following syntax:

```
Declaration:
            rewrite <name_of_the_rule>
                {subst("<string or regular expression to find>", "<replacement
string>", value(<field name>), flags());};
```

A single substitution rule can include multiple substitutions that are applied sequentially to the message. Note that rewriting rules must be included in the log statement to have any effect.

Tip
For case-insensitive searches, add the *flags(ignore-case)* option; to replace every occurrence of the string, add *flags(global)* option.

Example 4.41. Using substitution rules
The following example replaces the first occurrence of the string *IP* in the text of the message with the string *IP-Address*.

```
rewrite r_rewrite_subst{subst("IP", "IP-Address", value("MESSAGE"));};
```

To replace every occurrence, use:

```
rewrite r_rewrite_subst{subst("IP", "IP-Address", value("MESSAGE"), flags("global"));};
```

Multiple substitution rules are applied sequentially; the following rules replace the first occurrence of the string *IP* with the string *IP-Addresses*.

```
rewrite r_rewrite_subst{subst("IP", "IP-Address", value("MESSAGE")); subst("Address",
"Addresses", value("MESSAGE"));};
```

To set a field of the message to a specific value, define the string to include in the message, and the field where it should be included. Setting a field can operate on any value available via macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (see *Section 6.6, Message parsers (p. 158)* for details). The only exceptions are the FACILITY, SEVERITY, TAGS, and the date-related fields, which cannot be rewritten. Note that the rewrite operation completely replaces any previous value of that field. Use the following syntax:

```
Declaration:
            rewrite <name_of_the_rule>
                {set("<string to include>", value(<field name>));};
```

Example 4.42. Setting message fields to a particular value
The following example sets the HOST field of the message to *myhost*.

```
rewrite r_rewrite_set{set("myhost", value("HOST"));};
```

The following example sets the sequence ID field of the RFC5424-formatted (IETF-syslog) messages to a fixed value.

```
rewrite r_sd { set("55555" value(".SDATA.meta.sequenceId")); };
```

It is also possible to set the value of a field that does not exist yet, and create a new name-value pair that is associated with the message. The following example created the *MODIFIED* field and sets its value to *yes*. If you use the *$MODIFIED* macro in a template or SQL table, its value will be *yes* for every message that was processed with this rewrite rule, and empty for every other message.

```
rewrite r_rewrite_set{set("yes", value("MODIFIED"));};
```

4.11. Configuring global syslog-ng options

The syslog-ng application has a number of global options governing DNS usage, the timestamp format used, and other general points. Each option may have parameters, similarly to driver specifications. To set global options, add an option statement to the syslog-ng configuration file using the following syntax:

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

Example 4.43. Using global options
To disable domain name resolving, add the following line to the syslog-ng configuration file:

```
options { use_dns(no); };
```

For a detailed list of the available options, see *Section 6.9, Global options (p. 170)*. See *Chapter 5, Best practices and examples (p. 86)* for important global options and recommendations on their use.

4.12. Encrypting log messages with TLS

This section describes how to configure TLS encryption in syslog-ng. For the concepts of using TLS in syslog-ng, see *Section 2.7, Secure logging using TLS (p. 11)*.

Create an X.509 certificate for the syslog-ng server.

Note
The `subject_alt_name` parameter (or the `Common Name` parameter if the `subject_alt_name` parameter is empty) of the server's certificate must contain the hostname or the IP address (as resolved from the syslog-ng clients and relays) of the server (for example `syslog-ng.example.com`).

Alternatively, the `Common Name` or the `subject_alt_name` parameter can contain a generic hostname, for example `*.example.com`.

Note that if the `Common Name` of the certificate contains a generic hostname, do not specify a specific hostname or an IP address in the `subject_alt_name` parameter.

Complete the following steps on every syslog-ng client host. Examples are provided using both the legacy BSD-syslog protocol (using the `tcp()` driver) and the new IETF-syslog protocol standard (using the `syslog()` driver):

4.12.1. Procedure – Configuring TLS on the syslog-ng clients

Step 1. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng server to the syslog-ng client hosts, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.
Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem`
The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

Issue the following command to create a symbolic link to the certificate that uses the hash returned by the previous command and the `.0` suffix.

```
ln -s cacert.pem 6d2962a8.0
```

Step 2. Add a destination statement to the syslog-ng configuration file that uses the `tls( ca_dir(path_to_ca_directory) )` option and specify the directory using the CA certificate. The destination must use the `tcp()` or `tcpv6()` destination driver, and the IP address and port parameters of the driver must point to the syslog-ng server.

Example 4.44. A destination statement using TLS
The following destination encrypts the log messages using TLS and sends them to the `6514/TCP` port of the syslog-ng server having the `10.1.2.3` IP address.

```
destination demo_tls_destination {
                tcp("10.1.2.3" port(6514)
                    tls( ca_dir("/opt/syslog-ng/etc/syslog-ng/ca.d")) ); };
```

A similar statement using the IETF-syslog protocol and thus the `syslog()` driver:

```
destination demo_tls_syslog_destination { syslog("10.1.2.3" port(6514)
                        transport("tls")
                        port(3214)
                        tls(ca_dir("/opt/syslog-ng/etc/syslog-ng/ca.d")) );
};
```

Step 3. Include the destination created in Step 2 in a log statement.

Warning
The encrypted connection between the server and the client fails if the *Common Name* or the *subject_alt_name* parameter of the server certificate does not contain the hostname or the IP address (as resolved from the syslog-ng clients and relays) of the server.

Do not forget to update the certificate files when they expire.

Complete the following steps on the syslog-ng server:

4.12.2. Procedure – Configuring TLS on the syslog-ng server

Step 1. Copy the certificate (for example `syslog-ng.cert`) of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/cert.d` directory. The certificate must be a valid X.509 certificate in PEM format.

Step 2. Copy the private key (for example `syslog-ng.key`) matching the certificate of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/key.d` directory. The key must be in PEM format, and must not be password-protected.

Step 3. Add a source statement to the syslog-ng configuration file that uses the *tls(key_file(key_file_fullpathname) cert_file(cert_file_fullpathname))* option and specify the key and certificate files. The source must use the source driver (*tcp()* or *tcpv6()*) matching the destination driver used by the syslog-ng client.

Example 4.45. A source statement using TLS
The following source receives log messages encrypted using TLS, arriving to the *1999/TCP* port of any interface of the syslog-ng server.

```
source demo_tls_source {
          tcp(ip(0.0.0.0) port(1999)
              tls( key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")

                cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert"))
 ); };
```

A similar source for receiving messages using the IETF-syslog protocol:

```
source demo_tls_syslog_source {
                    syslog(ip(0.0.0.0) port(1999)
                    transport("tls")
                    tls(
key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")
                cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert"))
 ); };
```

Step 4. Disable mutual authentication for the source by setting the following TLS option in the source statement: `tls( peer_verify(optional-untrusted);`
To configure mutual authentication, see *Section 4.13, Mutual authentication using TLS (p. 79)*.

Example 4.46. Disabling mutual authentication
The following source receives log messages encrypted using TLS, arriving to the `1999/TCP` port of any interface of the syslog-ng server. The identity of the syslog-ng client is not verified.

```
source demo_tls_source {
           tcp(ip(0.0.0.0) port(1999)
               tls( key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")

                 cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert")

                    peer_verify(optional-untrusted))  ); };
```

A similar source for receiving messages using the IETF-syslog protocol:

```
source demo_tls_syslog_source {
                        syslog(ip(0.0.0.0) port(1999)
                        transport("tls")
                        tls(
key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")

cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert")
                        peer_verify(optional-untrusted))  ); };
```

Warning
Do not forget to update the certificate and key files when they expire.

For the details of the available `tls()` options, see *Section 6.10, TLS options (p. 177)*.

4.13. Mutual authentication using TLS

This section describes how to configure mutual authentication between the syslog-ng server and the client. Configuring mutual authentication is similar to configuring TLS (see *Section 4.12, Encrypting log messages with TLS (p. 77)*), but the server verifies the identity of the client as well. Therefore, each client must have a certificate, and the server must have the certificate of the CA that issued the certificate of the clients. For the concepts of using TLS in syslog-ng, see *Section 2.7, Secure logging using TLS (p. 11)*.

Complete the following steps on every syslog-ng client host. Examples are provided using both the legacy BSD-syslog protocol (using the `tcp()` driver) and the new IETF-syslog protocol standard (using the `syslog()` driver):

4.13.1. Procedure – Configuring TLS on the syslog-ng clients

Step 1. Create an X.509 certificate for the syslog-ng client.

Step 2. Copy the certificate (for example `client_cert.pem`) and the matching private key (for example `client.key`) to the syslog-ng client host, for example into the `/opt/syslog-ng/etc/syslog-ng/cert.d` directory. The certificate must be a valid X.509 certificate in PEM format and must not be password-protected.

Step 3. Copy the CA certificate of the Certificate Authority (for example `cacert.pem`) that issued the certificate of the syslog-ng server to the syslog-ng client hosts, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.
Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem`
The result is a hash (for example *6d2962a8*), a series of alphanumeric characters based on the Distinguished Name of the certificate.

Issue the following command to create a symbolic link to the certificate that uses the hash returned by the previous command and the *.0* suffix.

```
ln -s cacert.pem 6d2962a8.0
```

Step 4. Add a destination statement to the syslog-ng configuration file that uses the *tls(ca_dir(path_to_ca_directory))* option and specify the directory using the CA certificate. The destination must use the *tcp()* or *tcpv6()* destination driver, and the IP address and port parameters of the driver must point to the syslog-ng server. Include the client's certificate and private key in the *tls()* options.

Example 4.47. A destination statement using mutual authentication
The following destination encrypts the log messages using TLS and sends them to the *1999/TCP* port of the syslog-ng server having the *10.1.2.3* IP address. The private key and the certificate file authenticating the client is also specified.

```
destination demo_tls_destination {
    tcp("10.1.2.3" port(1999)
        tls( ca_dir("/opt/syslog-ng/etc/syslog-ng/ca.d")
             key_file("/opt/syslog-ng/etc/syslog-ng/key.d/client.key")
            cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/client_cert.pem")) );
 };
```

```
destination demo_tls_syslog_destination {
    syslog("10.1.2.3" port(1999)
        transport("tls")
            tls( ca_dir("/opt/syslog-ng/etc/syslog-ng/ca.d")
                 key_file("/opt/syslog-ng/etc/syslog-ng/key.d/client.key")
                cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/client_cert.pem"))
 ); };
```

Step 5. Include the destination created in Step 2 in a log statement.

Warning
The encrypted connection between the server and the client fails if the *Common Name* or the *subject_alt_name* parameter of the server certificate does not the hostname or the IP address (as resolved from the syslog-ng clients and relays) of the server.

Do not forget to update the certificate files when they expire.

Complete the following steps on the syslog-ng server:

4.13.2. Procedure – Configuring TLS on the syslog-ng server

Step 1. Copy the certificate (for example `syslog-ng.cert`) of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/cert.d` directory. The certificate must be a valid X.509 certificate in PEM format.

Step 2. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng clients to the syslog-ng server, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.
Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem`
The result is a hash (for example *6d2962a8*), a series of alphanumeric characters based on the Distinguished Name of the certificate.

Issue the following command to create a symbolic link to the certificate that uses the hash returned by the previous command and the *.0* suffix.

```
ln -s cacert.pem 6d2962a8.0
```

Step 3. Copy the private key (for example `syslog-ng.key`) matching the certificate of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/key.d` directory. The key must be in PEM format, and must not be password-protected.

Step 4. Add a source statement to the syslog-ng configuration file that uses the *tls(key_file(key_file_fullpathname) cert_file(cert_file_fullpathname))* option and specify the key and certificate files. The source must use the source driver (*tcp()* or *tcpv6()*) matching the destination driver used by the syslog-ng client. Also specify the directory storing the certificate of the CA that issued the client's certificate.

Example 4.48. A source statement using TLS
The following source receives log messages encrypted using TLS, arriving to the *1999/TCP* port of any interface of the syslog-ng server.

```
source demo_tls_source {
          tcp(ip(0.0.0.0) port(1999)
              tls( key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")

                cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert")

                   ca_dir("/opt/syslog-ng/etc/syslog-ng/ca.d")) ); };
```

A similar source for receiving messages using the IETF-syslog protocol:

```
source demo_tls_syslog_source {
                        syslog(ip(0.0.0.0) port(1999)
                        transport("tls")
                        tls(
key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")

cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert")
                        ca_dir("/opt/syslog-ng/etc/syslog-ng/ca.d")) ); };
```

Warning
Do not forget to update the certificate and key files when they expire.

For the details of the available `tls()` options, see *Section 6.10, TLS options (p. 177)*.

4.14. Configuring syslog-ng clients

To configure syslog-ng on a client host, complete the following steps:

4.14.1. Procedure – Configuring syslog-ng on client hosts

Step 1. Install the syslog-ng application on the host. See *Chapter 3, Installing syslog-ng (p. 26)* for details installing syslog-ng on specific operating systems.

Step 2. Configure the local sources that collect the log messages of the host.

Step 3. Create a network destination that points directly to the syslog-ng server, or to a local relay.

Step 4. Create a log statement connecting the local sources to the syslog-ng server or relay.

Step 5. If the logs will also be stored locally on the host, create local file destinations.

Step 6. Create a log statement connecting the local sources to the file destination.

Step 7. Set filters and options (for example TLS encryption) as necessary.

Example 4.49. A simple configuration for clients

The following is a simple configuration file that collects local log messages and forwards them to a logserver using the IETF-syslog protocol.

```
@version:3.0

options {
            mark_freq(30);
        };

source s_local { unix-stream("/dev/log"); internal(); };

destination d_syslog_tcp {
             syslog("192.168.1.1" transport("tcp") port(2010));
        };

log { source(s_local);destination(d_syslog_tcp); };
```

4.15. Configuring syslog-ng relays

To configure syslog-ng on a relay host, complete the following steps:

4.15.1. Procedure – Configuring syslog-ng on relay hosts

Step 1. Install the syslog-ng application on the host. See *Chapter 3, Installing syslog-ng (p. 26)* for details installing syslog-ng on specific operating systems.

Step 2. Configure the network sources that collect the log messages sent by the clients.

Step 3. Create a network destination that points to the syslog-ng server.

Step 4. Create a log statement connecting the network sources to the syslog-ng server.

Step 5. Configure the local sources that collect the log messages of the relay host.

Step 6. Create a log statement connecting the local sources to the syslog-ng server.

Step 7. Set filters and options (for example TLS encryption) as necessary.

Note
By default, the syslog-ng server will treat the relayed messages as if they were created by the relay host, not the host that originally sent them to the relay. In order to use the original hostname on the syslog-ng server, use the *keep_hostname(yes)* option both on the syslog-ng relay and the syslog-ng relay. This option can be set individually for every source if needed.

In relay mode, syslog-ng cannot write messages received from network sources into files; the *file()* destination is disabled. The following sources are network sources: *syslog()*, *tcp()*, *tcp6()*, *udp()*, *udp6()*.

Example 4.50. A simple configuration for relays
The following is a simple configuration file that collects local and incoming log messages and forwards them to a logserver using the IETF-syslog protocol.

```
@version:3.0

                options {
                mark_freq(30);
                keep_hostname(yes);
                };

                source s_local { unix-stream("/dev/log"); internal(); };
                source s_network { syslog(transport(tcp))};

                destination d_syslog_tcp {
                syslog("192.168.1.5" transport("tcp") port(2010)
                );
                };

                log { source(s_local); source(s_network);
                      destination(d_syslog_tcp); };
```

4.16. Configuring syslog-ng servers

To configure syslog-ng on a server host, complete the following steps:

4.16.1. Procedure – Configuring syslog-ng on server hosts

Step 1. Install the syslog-ng application on the host. See *Chapter 3, Installing syslog-ng (p. 26)* for details installing syslog-ng on specific operating systems.

Step 2. Configure the network sources that collect the log messages sent by the clients and relays.

Step 3. Create local destinations that will store the log messages, for example files or programs.

Step 4. Create a log statement connecting the network sources to the local destinations.

Step 5. Configure the local sources that collect the log messages of the syslog-ng server.

Step 6. Create a log statement connecting the local sources to the local destinations.

Step 7. Set filters, options (for example TLS encryption) and other advanced features as necessary.

Note
By default, the syslog-ng server will treat the relayed messages as if they were created by the relay host, not the host that originally sent them to the relay. In order to use the original hostname on the syslog-ng server, use the *keep_hostname(yes)* option both on the syslog-ng relay and the syslog-ng relay. This option can be set individually for every source if needed.

Example 4.51. A simple configuration for servers
The following is a simple configuration file for syslog-ng Open Source Edition that collects incoming log messages and stores them in a text file.

```
@version:3.0
    options {
        time_reap(30);
        mark_freq(10);
        keep_hostname(yes);
        };

    source s_local { unix-stream("/dev/log"); internal();};
    source s_network { syslog(transport(tcp))};

    destination d_logs {
        file(
            "/var/log/syslog-ng/logs.txt"
            owner("root")
            group("root")
            perm(0777)
            ); };

    log { source(s_local); source(s_network); destination(d_logs); };
```

4.17. Troubleshooting syslog-ng

This section provides tips and guidelines about troubleshooting problems related to syslog-ng.

Tip
As a general rule, first try to get logging the messages to a local file. Once this is working, you know that syslog-ng is running correctly and receiving messages, and you can proceed to forwarding the messages to the server.

If the syslog-ng server does not receive the messages, use tcpdump or a similar packet sniffer tool on the client to verify that the messages are sent correctly, and on the server to verify that it receives the messages.

If syslog-ng is closing the connections for no apparent reason, be sure to check the log messages of syslog-ng. You might also want to run syslog-ng with the `--verbose` or `--debug` command-line options for more-detailed log messages. Starting from syslog-ng OSE version 3.1, you can enable these messages without restarting syslog-ng using the `syslog-ng-ctl verbose --set=on` command. See the syslog-ng-ctl man page for details at *syslog-ng-ctl(1) (p. 192)*.

Similarly, build up encrypted connections step-by-step: first create a working unencrypted (for example TCP) connection, then add TLS encryption, and finally client authentication if needed.

4.17.1. Creating syslog-ng core files

When syslog-ng crashes for some reason, it can create a core file that contains important troubleshooting information. To enable core files, complete the following procedure:

4.17.1.1. Procedure – Creating syslog-ng core files

Step 1. Core files are produced only if the *`maximum core file size`* ulimit is set to a high value in the init script of syslog-ng. Add the following line to the init script of syslog-ng:

```
ulimit -c unlimited
```

Step 2. Verify that syslog-ng has permissions to write the directory it is started from, for example `/opt/syslog-ng/sbin/`.

Step 3. If syslog-ng crashes, it will create a core file in the directory syslog-ng was started from.

Step 4. To test that syslog-ng can create a core file, you can create a crash manually. For this, determine the PID of syslog-ng (for example using the `ps -All|grep syslog-ng` command), then issue the following command: `kill -ABRT <syslog-ng pid>`
This should create a core file in the current working directory.

4.17.2. Running a failure script

When syslog-ng is abnormally terminated, it can execute a user-created failure script. This can be used for example to send an automatic e-mail notification. The script must be located at `/opt/syslog-ng/sbin/syslog-ng-failure`.

4.17.3. Stopping syslog-ng

To avoid problems, always use the init scripts to stop syslog-ng (*`/etc/init.d/syslog-ng stop`*), instead of using the `kill` command. This is especially true on Solaris and HP-UX systems, here use *`/etc/init.d/syslog stop`*.

Chapter 5. Best practices and examples

This chapter discusses some special examples and recommendations.

5.1. General recommendations

This section provides general tips and recommendations on using syslog-ng. Some of the recommendations are detailed in the subsequent sections.

- Do not base the separation of log messages into different files on the `facility` parameter. As several applications and processes can use the same facility, the facility does not identify the application that sent the message. By default, the `facility` parameter is not even included in the log message itself. In general, sorting the log messages into several different files can make finding specific log messages difficult. If you must create separate log files, use the application name.
- Standard log messages include the local time of the sending host, without any time zone information. It is recommended to replace this timestamp with an ISODATE timestamp, because the ISODATE format includes the year and timezone as well. To convert all timestamps to the ISODATE format, include the following line in the syslog-ng configuration file:

  ```
  options {ts_format(iso) ; };
  ```

- Resolving the IP addresses of the clients to domain names can decrease the performance of syslog-ng. See *Section 5.4, Using name resolution in syslog-ng (p. 87)* for details.

5.2. Handling lots of parallel connections

When syslog-ng is receiving messages from a large number of TCP or unix-stream connections, the CPU usage of syslog-ng might increase even if the number of messages is low. By default, syslog-ng processes every message when it is received. To reduce the CPU usage, process the incoming messages in batches. To accomplish this, instruct syslog-ng to wait for a short time before processing a message. During this period additional messages might arrive that can be processed together with the original message. To process log messages in batches, set the `time_sleep()` option (measured in milliseconds) to a non-zero value. Include the following line in your syslog-ng configuration:

```
options { time_sleep(20); };
```

Note
It is not recommended to increase the `time_sleep()` parameter above 100ms, as that might distort timestamps, slow down syslog-ng, and cause messages to be dropped.

When modifying the `time_sleep()` option, also adjust the `log_fetch_limit()` and `log_fifo_size()` options accordingly.

The `max_connections()` parameter limits the number of parallel connections for the source.

If adjusting the `time_sleep()` option is not desired for some reason, an alternative solution is to use `unix-stream()`, `udp()` and `unix-dgram()` sources instead of `tcp()` connections.

5.3. Handling large message load

This section provides tips on optimizing the performance of syslog-ng. Optimizing the performance is important for syslog-ng hosts that handle large traffic.

- Disable DNS resolution, or resolve hostnames locally. See *Section 5.4, Using name resolution in syslog-ng (p. 87)* for details.
- Enable flow-control for the TCP sources. See *Section 2.12, Managing incoming and outgoing messages with flow-control (p. 16)* for details.
- Do not use the `usertty()` destination driver. Under heavy load, the users are not be able to read the messages from the console, and it slows down syslog-ng.
- Do not use regular expressions in our filters. Evaluating general regular expressions puts a high load on the CPU. Use simple filter functions and logical operators instead. See *Section 4.6.2, Optimizing regular expressions in filters (p. 68)* for details.
- When receiving lots of messages using the UDP protocol, increase the size of the UDP receive buffer on the syslog-ng hosts. For information about sizing and modifying the UDP buffer, see *http://www.29west.com/docs/THPM/udp-buffer-sizing.html*.

5.4. Using name resolution in syslog-ng

The syslog-ng application can resolve the hostnames of the clients and include them in the log messages. However, the performance of syslog-ng is severely degraded if the domain name server is unaccessible or slow. Therefore, it is not recommended to resolve hostnames in syslog-ng. If you must use name resolution from syslog-ng, consider the following:

- Use DNS caching. Verify that the DNS cache is large enough to store all important hostnames. (By default, the syslog-ng DNS cache stores `1007` entries.)

```
options { dns_cache(2000); };
```

- If the IP addresses of the clients change only rarely, set the expiry of the DNS cache large.

```
options { dns_cache_expire(87600); };
```

- If possible, resolve the hostnames locally. See *Section 5.4.1, Resolving hostnames locally (p. 88)* for details.

Note
Domain name resolution is important mainly in relay and server mode.

5.4.1. Resolving hostnames locally

Resolving hostnames locally enables you to display hostnames in the log files for frequently used hosts, without having to rely on a DNS server. The known IP address – hostname pairs are stored locally in a file. In the log messages, syslog-ng will replace the IP addresses of known hosts with their hostnames. To configure local name resolution, complete the following steps:

5.4.1.1. Procedure – Resolving hostnames locally

Step 1. Add the hostnames and the respective IP addresses to the file used for local name resolution. On Linux and UNIX systems, this is the `/etc/hosts` file. Consult the documentation of your operating system for details.

Step 2. Instruct syslog-ng to resolve hostnames locally. Set the *`use_dns()`* option of syslog-ng to *`persist_only`*.

Step 3. Set the *`dns_cache_hosts()`* option to point to the file storing the hostnames.

```
options {
        use_dns(persist_only);
        dns_cache_hosts(/etc/hosts); };
```

5.5. Collecting logs from chroot

To collect logs from a chroot using a syslog-ng client running on the host, complete the following steps:

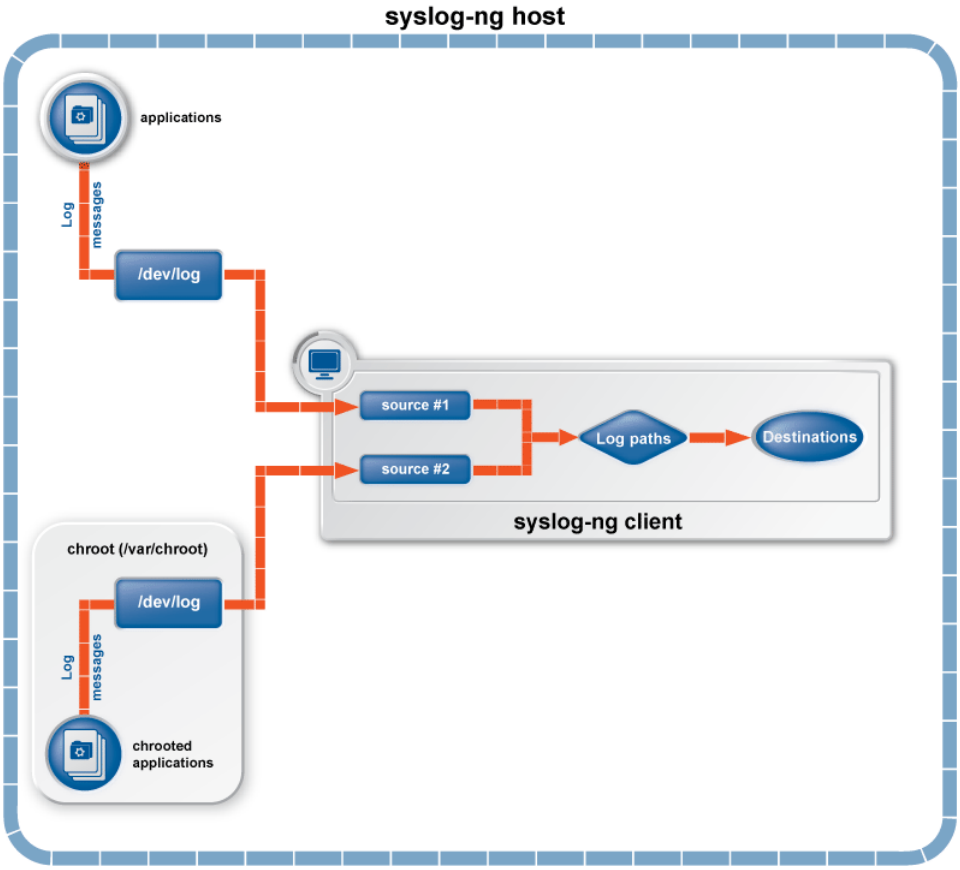

Figure 5.1. Collecting logs from chroot

5.5.1. Procedure – Collecting logs from chroot

Step 1. Create a `/dev` directory within the chroot. The applications running in the chroot send their log messages here.

Step 2. Create a local source in the configuration file of the syslog-ng application running outside the chroot. This source should point to the `/dev/log` file within the chroot (for example to the `/chroot/dev/log` directory).

Step 3. Include the source in a log statement.

Note
You need to set up timezone information within your chroot as well. This usually means creating a symlink to `/etc/localtime`.

5.6. Replacing klogd on Linux

The syslog-ng application can replace both the syslogd and klogd daemons on Linux hosts. To replace klogd, complete the following steps:

5.6.1. Procedure – Replacing klogd on Linux

Step 1. Add a file source pointing to `/proc/kmsg` to the syslog-ng configuration file.

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

Warning
Do not use a pipe source to read `/proc/kmsg`; pipe opens the source in read-write mode and this may cause problems when using SELinux or similar security measures.

Step 2. Include the source defined in Step 1 in a log path.

Step 3. Stop klogd.

Warning
Do not run klogd and syslog-ng simultaneously when using syslog-ng to read `/proc/kmsg`, as it might block syslog-ng.

5.7. A note on timezones and timestamps

If the clients run syslog-ng, then use the ISO timestamp, because it includes timezone information. That way you do not need to adjust the *`recv_time_zone()`* parameter of syslog-ng.

If you want syslog-ng to output timestamps in Unix (POSIX) time format, use the *`S_UNIXTIME`* and *`R_UNIXTIME`* macros. You do not need to change any of the timezone related parameters, because the timestamp information of incoming messages is converted to Unix time internally, and Unix time is a timezone-independent time representation. (Actually, Unix time measures the number of seconds elapsed since midnight of Coordinated Universal Time (UTC) January 1, 1970, but does not count leap seconds.)

5.8. Dropping messages

To skip the processing of a message without sending it to a destination, create a log statement with the appropriate filters, but do not include any destination in the statement, and use the *`final`* flag.

Example 5.1. Skipping messages
The following log statement drops all *debug* level messages without any further processing.

```
filter demo_debugfilter { level(debug); };
log { source(s_all); filter(demo_debugfilter); flags(final); };
```

Chapter 6. Reference

This chapter documents the drivers and options that can be used in the configuration file. For details on how to use syslog-ng, see *Chapter 4, Configuring syslog-ng (p. 42)*.

6.1. Source drivers

6.1.1. internal()

All messages generated internally by syslog-ng use this special source. To collect warnings, errors and notices from syslog-ng itself, include this source in one of your source statements.

Note
Internal messages always use the local timezone of the host.

```
internal()
```

This driver does not have any parameters.

Example 6.1. Using the internal() driver

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

6.1.2. file()

Collects log messages from plain-text files. The file driver has a single required parameter specifying the file to open.

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

Note
If the message does not have a proper syslog header, syslog-ng treats messages received from files as sent by the *kern* facility. Use the *default-facility* and *default-priority* options in the source definition to assign a different facility if needed.

The *file()* driver has the following options:

default-facility()

Type:	facility string
Default:	kern

Description: This parameter assigns a facility value to the messages received from the file source, if the message does not specify one.

default-priority()

Type:	priority string
Default:	

Description: This parameter assigns an emergency level to the messages received from the file source, if the message does not specify one.

file

Type:	filename with path
Default:	

Description: The file to read messages from. Note that only syslog-ng PE supports wildcards in the filename (but not in the pathname). To monitor the subdirectories as well, use the `recursive` option.

encoding()

Type:	string
Default:	

Description: Specifies the characterset (encoding, for example `UTF-8`) of messages using the legacy BSD-syslog protocol. To list the available character sets on a host, execute the `iconv -l` command.

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The `kernel` flag makes the source default to the `LOG_KERN | LOG_CRIT` priority if not specified otherwise.

The `no-multi-line` flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the `no-parse` flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the `store-legacy-msghdr` flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). Note that `store-legacy-msghdr` should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the `syslog` driver.

The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero, syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global `log_fetch_limit()` option, which defaults to `10`.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log_fetch_limit()` is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global *log_msg_size()* option, which defaults to *8192*.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding *kernel:* to the kernel messages on Linux. NOTE: This option is deprecated. Use *program_override()* instead.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the *pipe()*, *unix-dgram*, and *unix-stream* drivers.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

recursive

Type:	yes or no
Default:	no

Description: When enabled, syslog-ng PE monitors every subdirectory of the directory set in the path of the *file* parameter, and reads log messages from files with the set filename. The *recursive* option can be used together with wildcards in the filename.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example *tags("dmz", "router")*. This option is available only in syslog-ng 3.1 and later.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

Example 6.2. Using the file() driver

```
source s_file { file("/var/log/messages"); };
```

Example 6.3. Tailing files
The following source checks the `access.log` file every second for new messages.

```
source s_tail { file("/var/log/apache/access.log"
                     follow_freq(1) flags(no-parse)); };
```

6.1.3. pipe()

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

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

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

Note
As of syslog-ng Open Source Edition 3.0.2, pipes are created automatically. In earlier versions, you had to create the pipe using the `mkfifo(1)` command.

The *pipe* driver has the following options:

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The `kernel` flag makes the source default to the `LOG_KERN | LOG_CRIT` priority if not specified otherwise.

The `no-multi-line` flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the `no-parse` flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the `store-legacy-msghdr` flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). Note that `store-legacy-msghdr` should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the `syslog` driver.

The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero, syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global *log_fetch_limit()* option, which defaults to *10*.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if *log_fetch_limit()* is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global *log_msg_size()* option, which defaults to *8192*.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding *kernel:* to the kernel messages on Linux. NOTE: This option is deprecated. Use *program_override()* instead.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the *pipe()*, *unix-dgram*, and *unix-stream* drivers.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

pipe

Type:	filename with path
Default:	

Description: The filename of the pipe to read messages from.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example *tags("dmz", "router")*. This option is available only in syslog-ng 3.1 and later.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

Example 6.4. Using the pipe() driver

```
source s_pipe { pipe("/dev/pipe" pad_size(2048)); };
```

6.1.4. program()

The program driver starts an external application and reads messages from the standard output (stdout) of the application. It is mainly useful to receive log messages from daemons that accept incoming messages and convert them to log messages.

The program driver has a single required parameter, specifying the name of the application to start.

```
Declaration:
    program(filename);
```

Note
The program is restarted automatically if it exits.

The *program* driver has the following options:

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The *kernel* flag makes the source default to the *LOG_KERN | LOG_CRIT* priority if not specified otherwise.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.

The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero, syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global _`log_fetch_limit()`_ option, which defaults to `10`.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log_fetch_limit()` is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global `log_msg_size()` option, which defaults to `8192`.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program_override()` instead.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad_size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

program

Type:	filename with path
Default:	

Description: The name of the application to start and read messages from.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the `program_override("kernel")` option in the source containing `/proc/kmsg`. NOTE: This option replaces the deprecated `log_prefix()` option.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example `tags("dmz", "router")`. This option is available only in syslog-ng 3.1 and later.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

Example 6.5. Using the program() driver

```
source s_program { program("/etc/init.d/mydaemon"); };
```

6.1.5. sun-streams() driver

Solaris uses its *STREAMS* framework to send messages to the *syslogd* process.

Newer versions of Solaris (2.5.1 and above), use a new IPC in addition to *STREAMS*, called door to confirm the delivery of a message. The syslog-ng application supports this new IPC mechanism via the *door()* option (see below).

Note
The *sun-streams()* driver must be enabled when the syslog-ng application is compiled (see `./configure --help`).

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

```
Declaration:
    sun-streams(name_of_the_streams_device door(filename_of_the_door));
```

door()

Type:	string
Default:	none

Description: Specifies the filename of a door to open, needed on Solaris above 2.5.1.

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The `kernel` flag makes the source default to the `LOG_KERN | LOG_CRIT` priority if not specified otherwise.

The `no-multi-line` flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the `no-parse` flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the `store-legacy-msghdr` flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). Note that `store-legacy-msghdr` should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the `syslog` driver.

The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero, syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global _`log_fetch_limit()`_ option, which defaults to `10`.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log_fetch_limit()` is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global `log_msg_size()` option, which defaults to `8192`.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program_override()` instead.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad_size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the `program_override("kernel")` option in the source containing `/proc/kmsg`. NOTE: This option replaces the deprecated `log_prefix()` option.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example `tags("dmz", "router")`. This option is available only in syslog-ng 3.1 and later.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

Example 6.6. Using the sun-streams() driver

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

6.1.6. syslog()

This driver enables to receive messages from the network using the new standard syslog protocol and message format (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details about the protocol). UDP, TCP, and TLS-encrypted TCP can all be used to transport the messages.

```
Declaration:
    syslog(ip() port() transport() options());
```

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The `kernel` flag makes the source default to the `LOG_KERN | LOG_CRIT` priority if not specified otherwise.

The `no-multi-line` flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the `no-parse` flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the `store-legacy-msghdr` flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). Note that `store-legacy-msghdr` should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the `syslog` driver.

The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero, syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

host_override()

Type:	string
Default:	

Description: Replaces the $HOST part of the message with the parameter string.

ip() or localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to. Note that this is not the address where messages are accepted from.

ip_tos()

Type:	number
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip_ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to sources should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the syslog-ng connections, client-side (destination) connections are always reopened after receiving a HUP signal unless the *keep-alive* option is enabled for the destination.

keep_hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting. Enable this option to use hostname-related macros. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available. When relaying messages, enable this option on the syslog-ng server and also on every relay, otherwise syslog-ng will treat incoming messages as if they were sent by the last relay.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global *log_fetch_limit()* option, which defaults to *10*.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log_fetch_limit()` is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global `log_msg_size()` option, which defaults to `8192`.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program_override()` instead.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

port() or localport()

Type:	number
Default:	514

Description: The port number to bind to.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

so_broadcast()

Type:	yes or no
Default:	no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. See the `socket(7)` manual page for details.

so_keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. See the `socket(7)` manual page for details.

so_rcvbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. See the `socket(7)` manual page for details.

so_sndbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket send buffer in bytes. See the `socket(7)` manual page for details.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example *`tags("dmz", "router")`*. This option is available only in syslog-ng 3.1 and later.

tcp-keep-alive()

Type:	yes or no
Default:	no

Description: This is an obsolete alias of the *`so_keepalive()`* option.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

transport

Type:	udp, tcp, or tls
Default:	tcp

Description: Specifies the protocol used to receive messages from the source.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations and can only be used with the *`tcp`* transport protocols. See *Section 6.10, TLS options (p. 177)* for more information.

use_dns()

Type:	yes, no, persist_only
Default:	yes

Description: Enable or disable DNS usage. The *`persist_only`* option attempts to resolve hostnames locally from file (for example from `/etc/hosts`). 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 are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

use_fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

Example 6.7. Using the syslog() driver
TCP source listening on the localhost on port 1999.

```
source s_syslog { syslog(ip(127.0.0.1) port(1999) transport("tcp")); };
```

UDP source with defaults.

```
source s_udp { syslog( transport("udp")); };
```

Encrypted source where the client is also authenticated. See *Section 6.10, TLS options (p. 177)* for details on the encryption settings.

```
source s_syslog_tls{ syslog(
                     ip(10.100.20.40)
                     transport("tls")
                     tls(
                     peer-verify(required-trusted)
                     ca_dir('/opt/syslog-ng/etc/syslog-ng/keys/ca.d/')
                     key_file('/opt/syslog-ng/etc/syslog-ng/keys/server_privatekey.pem')
                     cert_file('/opt/syslog-ng/etc/syslog-ng/keys/server_certificate.pem')

                     )
                     );};
```

6.1.7. tcp(), tcp6(), udp() and udp6()

The `tcp()`, `tcp6()`, `udp()`, `udp6()` drivers can receive messages from the network using the TCP and UDP networking protocols. The `tcp6()` and `udp6()` drivers use the IPv6 network protocol, while `tcp()` and `udp()` use IPv4.

The `tcp()` and `udp()` drivers do not have any required 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 only one interface, use the `localip()` parameter as described below.

Note
The tcp port 514 is reserved for use with `rshell`, so select a different port if syslog-ng and `rshell` is used at the same time.

If you specify a multicast bind address to `udp()` and `udp6()`, syslog-ng will automatically join the necessary multicast group. TCP does not support multicasting.

The syslog-ng application supports TLS (Transport Layer Security, also known as SSL) for the tcp() and tcp6() drivers. See the TLS-specific options below and *Section 4.12, Encrypting log messages with TLS (p. 77)* for details.

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

The following options are valid for *tcp()*, *tcp6()*, *udp()*, and *udp6()* drivers:

encoding()

Type:	string
Default:	

Description: Specifies the characterset (encoding, for example *UTF-8*) of messages using the legacy BSD-syslog protocol. To list the available character sets on a host, execute the `iconv -l` command.

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The *kernel* flag makes the source default to the *LOG_KERN | LOG_CRIT* priority if not specified otherwise.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.

The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero, syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

host_override()

Type:	string
Default:	

Description: Replaces the $HOST part of the message with the parameter string.

ip() or localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to. Note that this is not the address where messages are accepted from.

ip_tos()

Type:	number
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip_ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to sources should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the syslog-ng connections, client-side (destination) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the destination.

keep_hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting. Enable this option to use hostname-related macros. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available. When relaying messages, enable this option on the syslog-ng server and also on every relay, otherwise syslog-ng will treat incoming messages as if they were sent by the last relay.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global *log_fetch_limit()* option, which defaults to *10*.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if *log_fetch_limit()* is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global *log_msg_size()* option, which defaults to *8192*.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding *kernel:* to the kernel messages on Linux. NOTE: This option is deprecated. Use *program_override()* instead.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

port() or localport()

Type:	number
Default:	514

Description: The port number to bind to.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

so_broadcast()

Type:	yes or no
Default:	no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. See the `socket(7)` manual page for details.

so_keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. See the `socket(7)` manual page for details.

so_rcvbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. See the `socket(7)` manual page for details.

so_sndbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket send buffer in bytes. See the `socket(7)` manual page for details.

tcp-keep-alive()

Type:	yes or no
Default:	no

Description: This is an obsolete alias of the *`so_keepalive()`* option.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example *`tags("dmz", "router")`*. This option is available only in syslog-ng 3.1 and later.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations and can only be used with the *`tcp`* transport protocols. See *Section 6.10, TLS options (p. 177)* for more information.

use_dns()

Type:	yes, no, persist_only
Default:	yes

Description: Enable or disable DNS usage. The *persist_only* option attempts to resolve hostnames locally from file (for example from `/etc/hosts`). 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 are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

use_fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

Example 6.8. Using the udp() and tcp() drivers
A simple udp() source with default settings.

```
source s_udp { udp(); };# An UDP source with default settings.
```

A TCP source listening on the localhost interface, with a limited number of connections allowed.

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

A TCP source listening on a TLS-encrypted channel.

```
source s_tcp { tcp(ip(127.0.0.1) port(1999)
                   tls(peer-verify('required-trusted')
                       key_file('/opt/syslog-ng/etc/syslog-ng/syslog-ng.key')
                   cert_file('/opt/syslog-ng/etc/syslog-ng/syslog-ng.crt')));
};
```

A TCP source listening for messages using the IETF-syslog message format. Note that for transferring IETF-syslog messages, generally you are recommended to use the *syslog()* driver on both the client and the server, as it uses both the IETF-syslog message format and the protocol. See *Section 4.3.5, Collecting messages using the IETF syslog protocol (p. 51)* for details.

```
source s_tcp_syslog { tcp(ip(127.0.0.1) port(1999) flags(syslog-protocol) ); };
```

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

These two drivers behave similarly: they open an *AF_UNIX* socket and start listening on it for messages.

Both unix-stream and unix-dgram have a single required argument, specifying the filename of the socket to create.

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

The following options can be specified for these divers:

encoding()

Type:	string
Default:	

Description: Specifies the characterset (encoding, for example *UTF-8*) of messages using the legacy BSD-syslog protocol. To list the available character sets on a host, execute the `iconv -l` command.

flags()

Type:	empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng removes empty lines automatically.

The *kernel* flag makes the source default to the *LOG_KERN | LOG_CRIT* priority if not specified otherwise.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, etc.) is added automatically. This flag is useful for parsing files not complying to the syslog format.

If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful of the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.

The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.

The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically instead of being polled. This is useful for files which always indicate readability, even though no new lines were appended. If this value is higher than zero,

syslog-ng will not attempt to use `poll()` on the file, but checks whether the file changed every time the `follow_freq()` interval (in seconds) has elapsed. Floating-point numbers (for example `1.5`) can be used as well.

group()

Type:	string
Default:	root

Description: Set the gid of the socket.

host_override()

Type:	string
Default:	

Description: Replaces the $HOST part of the message with the parameter string.

keep-alive()

Type:	yes or no
Default:	yes

Description: Selects whether to keep connections open when syslog-ng is restarted; cannot be used with `unix-dgram()`.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global _`log_fetch_limit()`_ option, which defaults to `10`.

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log_fetch_limit()` is too high.

log_iw_size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control.

log_msg_size()

Type:	number
Default:	Use the global *log_msg_size()* option, which defaults to *8192*.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding *kernel:* to the kernel messages on Linux. NOTE: This option is deprecated. Use *program_override()* instead.

max-connections()

Type:	number
Default:	256

Description: Limits the number of simultaneously open connections. Cannot be used with *unix-dgram()*.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the *pipe()*, *unix-dgram*, and *unix-stream* drivers.

owner()

Type:	string
Default:	root

Description: Set the uid of the socket.

pad_size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all 0 messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes.

perm()

Type:	number
Default:	0666

Description: Set the permission mask. For octal numbers prefix the number with '0', for example: use 0755 for rwxr-xr-x.

program_override

Type:	string
Default:	

Description: Replaces the $PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

so_broadcast()

Type:	yes or no
Default:	no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. See the `socket(7)` manual page for details.

so_keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. See the `socket(7)` manual page for details.

so_rcvbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. See the `socket(7)` manual page for details.

so_sndbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket send buffer in bytes. See the `socket(7)` manual page for details.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example `tags("dmz", "router")`. This option is available only in syslog-ng 3.1 and later.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

Example 6.9. 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"); };
```

6.2. Destination drivers

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

6.2.1. file()

The file driver outputs messages to the specified text file, or to a set of files.

The destination filename may include macros which get expanded when the message is written, thus a simple `file()` driver may create several files. For more information on available macros see *Section 6.5, Macros (p. 154)*.

Warning
When creating several thousands separate log files, syslog-ng might not be able to open the required number of files. This might happen for example when using the `$HOST` macro in the filename while receiving messages from a large number of hosts. To overcome this problem, adjust the `--fd-limit` comman-line parameter of syslog-ng or the global ulimit parameter of your host. For setting the `--fd-limit` comman-line parameter of syslog-ng see the *syslog-ng(8) (p. 180)* manual page. For setting the `ulimit` parameter of the host, see the documentation of your operating system.

The `file()` destination has the following options:

create_dirs()

Type:	yes or no
Default:	no

Description: Enable creating non-existing directories.

dir_group()

Type:	string
Default:	root

Description: The group of directories created by syslog-ng.

dir_owner()

Type:	string
Default:	root

Description: The owner of directories created by syslog-ng.

dir_perm()

Type:	number
Default:	0600

Description: 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 directory creation is enabled (see the *create_dirs()* option below). For octal numbers prefix the number with *0*, for example use *0755* for *rwxr-xr-x*.

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the driver.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the `flush_lines` option for more information.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The `frac_digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

fsync()

Type:	yes or no
Default:	no

Description: Forces an `fsync()` call on the destination fd after each write. Note: enabling this option may seriously degrade performance.

group()

Type:	string
Default:	root

Description: Set the group of the created file to the one specified.

local_time_zone()

Type:	name of the timezone or the timezone offset
Default:	The local timezone.

Description: Sets the timezone used when expanding filename and tablename templates. The timezone can be specified as using the name of the (for example `time_zone("Europe/Budapest")`), or as the timezone offset (for example `+01:00`). The valid timezone names are listed under the `/usr/share/zoneinfo` directory.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

overwrite_if_older()

Type:	number
Default:	0

Description: If set to a value higher than 0, syslog-ng checks when the file was last modified before starting to write into the file. If the file is older than the specified amount of time (in seconds), then syslog-ng removes the existing file and opens a new file with the same name. In combination with for example the *$WEEKDAY* macro, this can be used for simple log rotation, in case not all history has to be kept. (Note that in this weekly log rotation example if its Monday 00:01, then the file from last Monday is not seven days old, because it was probably last modified shortly before 23:59 last Monday, so it is actually not even six days old. So in this case, set the *overwrite_if_older()* parameter to a-bit-less-than-six-days, for example, to *518000* seconds.

owner()

Type:	string
Default:	root

Description: Set the owner of the created file to the one specified.

perm()

Type:	number
Default:	0600

Description: The permission mask of the file if it is created by syslog-ng. For octal numbers prefix the number with *0*, for example use *0755* for *rwxr-xr-x*.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 6.5, Macros (p. 154)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *syslogd* or syslog-ng itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping ' and " in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying *0* or a lower value sets the output limit to unlimited.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global *ts_format()* parameter) for the specific destination. See also *Section 5.7, A note on timezones and timestamps (p. 90).*

Example 6.10. Using the file() driver

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

Example 6.11. 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));
};
```

6.2.2. 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. The filename can include macros.

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

Warning
As of syslog-ng Open Source Edition 3.0.2, pipes are created automatically. In earlier versions, you had to create the pipe using the `mkfifo(1)` command.

The *pipe()* destination has the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the driver.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the *flush_lines* option for more information.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored

for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

group()

Type:	string
Default:	root

Description: Set the group of the pipe to the one specified.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

owner()

Type:	string
Default:	root

Description: Set the owner of the pipe to the one specified.

perm()

Type:	number
Default:	0600

Description: The permission mask of the pipe. For octal numbers prefix the number with '0', for example: use 0755 for rwxr-xr-x.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the `Last message repeated n times.` message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 6.5, Macros (p. 154)*. Please note that for network destinations it might not be appropriate to change the template

as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *syslogd* or syslog-ng itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping ' and " in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying *0* or a lower value sets the output limit to unlimited.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global *ts_format()* parameter) for the specific destination. See also *Section 5.7, A note on timezones and timestamps (p. 90).*

Example 6.12. Using the pipe() driver

```
destination d_pipe { pipe("/dev/xconsole"); };
```

6.2.3. program()

This driver starts an external application or script and sends the log messages to its standard input (*stdin*).

The *program()* driver has a single required parameter, specifying a program name to start.

```
Declaration:
    program(command_to_run);
```

The *program()* destination has the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the driver.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the *flush_lines* option for more information.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the `Last message repeated n times.` message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 6.5, Macros (p. 154)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like `syslogd` or syslog-ng itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping `'` and `"` in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. See also *Section 5.7, A note on timezones and timestamps (p. 90)*.

Example 6.13. Using the program() destination driver

```
destination d_prog { program("/bin/script" template("<$PRI>$DATE $HOST $MSG\n"); ); };
```

6.2.4. sql()

This driver sends messages into an SQL database. The `sql()` driver has the following required parameters: `type`, `database`, `table`, `columns`, `values`.

```
Declaration:
    sql(database_type host_parameters database_parameters [options]);
```

The `sql()` destination has the following options:

columns

Type:	string list
Default:	"date", "facility", "level", "host", "program", "pid", "message"

Description: Name of the columns storing the data in `fieldname [dbtype]` format. The `[dbtype]` parameter is optional, and specifies the type of the field. By default, syslog-ng creates `text` columns. Note that not every database engine can index text fields.

database

Type:	string
Default:	n/a

Description: Name of the database that stores the logs.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The `frac_digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

host

Type:	hostname or IP address
Default:	n/a

Description: Hostname of the database server. Note that Oracle destinations do not use this parameter, but retrieve the hostname from the `/etc/tnsnames.ora` file.

indexes

Type:	string list
Default:	"date", "facility", "host", "program"

Description: The list of columns that are indexed by the database to speed up searching. To disable indexing for the destination, include the empty `indexes()` parameter in the destination, simply omitting the `indexes` parameter will cause syslog-ng to request indexing on the default columns.

local_time_zone()

Type:	name of the timezone or the timezone offset
Default:	The local timezone.

Description: Sets the timezone used when expanding filename and tablename templates. The timezone can be specified as using the name of the (for example `time_zone("Europe/Budapest")`), or as the timezone offset (for example `+01:00`). The valid timezone names are listed under the `/usr/share/zoneinfo` directory.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

null

Type:	string
Default:	

Description: If the content of a column matches the string specified in the `null()` parameter, the contents of the column will be replaced with an SQL NULL value. If unset (by default), the option does not match on any string. See the *Example 6.17, Using SQL NULL values (p. 136)* for details.

password

Type:	string
Default:	n/a

Description: Password of the database user.

table

Type:	string
Default:	n/a

Description: Name of the database table to use (can include macros). When using macros, note that some databases limit the length of table names.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

type

Type:	mssql, mysql, oracle, pgsql, or sqlite3
Default:	n/a

Description: Specifies the type of the database, i.e., the DBI database driver to use. Use the `mssql` option to send logs to an MSSQL database. See the examples of the databases on the following sections for details.

username

Type:	string
Default:	n/a

Description: Name of the database user.

values

Type:	string list
Default:	"${R_YEAR}-${R_MONTH}-${R_DAY} ${R_HOUR}:${R_MIN}:${R_SEC}", "$FACILITY", "$LEVEL", "$HOST", "$PROGRAM", "$PID", "$MSGONLY"

Description: The parts of the message to store in the fields specified in the `columns` parameter.

Note
If you specify *host="localhost"*, syslog-ng will use a socket to connect to the local database server. Use *host="127.0.0.1"* to force TCP communication between syslog-ng and the local database server.

To specify the socket to use, set and export the *MYSQL_UNIX_PORT* environment variable, for example `MYSQL_UNIX_PORT=/var/lib/mysql/mysql.sock; export MYSQL_UNIX_PORT`.

Example 6.14. Using the sql() driver
The following example sends the log messages into a PostgreSQL database running on the *logserver* host. The messages are inserted into the *logs* database, the name of the table includes the exact date and the name of the host sending the messages. The syslog-ng application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

```
destination d_sql {
  sql(type(pgsql)
  host("logserver") username("syslog-ng") password("password")
  database("logs")
  table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
  columns("datetime", "host", "program", "pid", "message")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid", "message"));
  };
```

The following example specifies the type of the database columns as well:

```
destination d_sql {
  sql(type(pgsql)
  host("logserver") username("syslog-ng") password("password")
  database("logs")
  table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
  columns("datetime varchar(16)", "host varchar(32)", "program  varchar(20)", "pid
varchar(8)", "message  varchar(200)")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid", "message"));
};
```

Example 6.15. Using the sql() driver with an Oracle database
The following example sends the log messages into an Oracle database running on the *logserver* host, which must be set in the `/etc/tnsnames.ora` file. The messages are inserted into the *LOGS* database, the name of the table includes the exact date when the messages were sent. The syslog-ng application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

```
destination d_sql {
  sql(type(oracle)
  username("syslog-ng") password("password")
  database("LOGS")
  table("msgs_${R_YEAR}${R_MONTH}${R_DAY}")
  columns("datetime varchar(16)", "host varchar(32)", "program varchar(32)", "pid
varchar(8)", "message varchar2")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid", "message"));
  };
```

The Oracle Instant Client retrieves the address of the database server from the `/etc/tnsnames.ora` file. Edit or create this file as needed for your configuration. A sample is provided below.

```
LOGS =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)
(HOST = logserver)
(PORT = 1521))
)
(CONNECT_DATA =
(SERVICE_NAME = EXAMPLE.SERVICE)
)
)
```

Example 6.16. Using the sql() driver with an MSSQL database
The following example sends the log messages into an MSSQL database running on the *logserver* host. The messages are inserted into the *syslogng* database, the name of the table includes the exact date when the messages were sent. The syslog-ng application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

```
destination d_mssql {
sql(type(mssql) host("logserver") port("1433")
  username("syslogng") password("syslogng") database("syslogng")
  table("msgs_${R_YEAR}${R_MONTH}${R_DAY}")columns("datetime varchar(16)", "host
varchar(32)",
  "program varchar(32)", "pid varchar(8)", "message varchar(4096)")
  values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
  indexes("datetime", "host", "program", "pid"));
  };
```

The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the syslog-ng server. Edit or create this file as needed for your configuration. A sample is provided below.

```
[default]
date = "%Y-%m-%d %H:%M:%S"
```

Example 6.17. Using SQL NULL values
The *null()* parameter of the SQL driver can be used to replace the contents of a column with a special SQL NULL value. To replace every column that contains an empty string with NULL, use the *null("")* option, for example

```
destination d_sql {
                    sql(type(pgsql)
                    host("logserver") username("syslog-ng") password("password")
                    database("logs")
                    table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
                    columns("datetime", "host", "program", "pid", "message")
                    values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
                    indexes("datetime", "host", "program", "pid", "message")
                    null(""));
                    };
```

To replace only a specific column (for example *pid*) if it is empty, assign a default value to the column, and use this default value in the *null()* parameter:

```
destination d_sql {
                    sql(type(pgsql)
                    host("logserver") username("syslog-ng") password("password")
                    database("logs")
                    table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
                    columns("datetime", "host", "program", "pid", "message")
                   values("$R_DATE", "$HOST", "$PROGRAM", "${PID:-@@NULL@@}", "$MSGONLY")

                    indexes("datetime", "host", "program", "pid", "message")
                    null("@@NULL@@"));
                    };
```

Ensure that the default value you use does not appear in the actual log messages, because other occurrences of this string will be replaced with NULL as well.

6.2.5. syslog()

The *syslog()* driver sends messages to a remote host (for example a syslog-ng server or relay) on the local intranet or internet using the new standard syslog protocol developed by IETF (see *Section 2.16.2, IETF-syslog messages (p. 22)* for details about the protocol). The protocol supports sending messages using the UDP, TCP, or the encrypted TLS networking protocols.

The required arguments of the driver are the address of the destination host (where messages should be sent) and the transport method (networking protocol).

The *udp* transport method automatically sends multicast packets if a multicast destination address is specified. The *tcp* and *tls* methods do not support multicasting.

```
Declaration:
    syslog(host transport [options]);
```

These destinations have the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the driver.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the *flush_lines* option for more information.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

ip_tos()

Type:	number
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip_ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the source. When the `keep-alive` option is enabled, syslog-ng saves the contents of the output queue of the destination when receiving a HUP signal, reducing the risk of losing messages

localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to before connecting to target.

localport()

Type:	number
Default:	0

Description: The port number to bind to. Messages are sent from this port.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

port() or destport()

Type:	number
Default:	601

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so_broadcast()

Type:	yes or no
Default:	no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. See the `socket(7)` manual page for details.

so_keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. See the `socket(7)` manual page for details.

so_rcvbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. See the `socket(7)` manual page for details.

so_sndbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket send buffer in bytes. See the `socket(7)` manual page for details.

spoof_source()

Type:	yes or no
Default:	no

Description: 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* configuration option.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the `Last message repeated n times.` message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 6.5, Macros (p. 154)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like `syslogd` or syslog-ng itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping `'` and `"` in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations. TLS can be used only with the `tcp` transport protocols. See *Section 6.10, TLS options (p. 177)* for more information.

transport

Type:	udp, tcp, or tls
Default:	tcp

Description: Specifies the protocol used to receive messages from the source.

ts_format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. See also *Section 5.7, A note on timezones and timestamps (p. 90)*.

Example 6.18. Using the syslog() driver

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

If name resolution is configured, the hostname of the target server can be used as well.

```
destination d_tcp { syslog(ip("target_host") transport("tcp") port(1999) localport(999));
 };
```

Send the log messages using TLS encryption and use mutual authentication. See *Section 6.10, TLS options (p. 177)* for details on the encryption and authentication options.

```
destination d_syslog_tls{
                syslog("10.100.20.40"
                transport("tls")
                port(6514)
                tls(peer-verify(required-trusted)
                ca_dir('/opt/syslog-ng/etc/syslog-ng/keys/ca.d/')
                key_file('/opt/syslog-ng/etc/syslog-ng/keys/client_key.pem')
                cert_file('/opt/syslog-ng/etc/syslog-ng/keys/client_certificate.pem'))
                );};
```

6.2.6. tcp(), tcp6(), udp(), and udp6()

This driver sends messages to another host on the local intranet or internet using the UDP or TCP protocol. The `tcp6()` and `udp6()` drivers use the IPv6 network 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.

The *udp()* and *udp6()* drivers automatically send multicast packets if a multicast destination address is specified. The *tcp()* and *tcp6()* drivers do not support multicasting.

```
Declaration:
    tcp(host [options]);
    udp(host [options]);
    tcp6(host [options]);
    udp6(host [options]);
```

Example 6.19. Using the tcp() driver

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

If name resolution is configured, the hostname of the target server can be used as well.

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

To send messages using the IETF-syslog message format without using the IETF-syslog protocol, enable the *syslog-protocol* flag:

```
destination d_tcp { tcp("10.1.2.3" port(1999) flags(syslog-protocol) ); };
```

(To use the IETF-syslog protocol, see *Section 6.2.5, syslog() (p. 136)*.)

These destinations have the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the driver.

The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.

The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the `flush_lines` option for more information.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The `frac_digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

ip_tos()

Type:	number
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip_ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the source. When the `keep-alive` option is enabled, syslog-ng saves the contents of the output queue of the destination when receiving a HUP signal, reducing the risk of losing messages

localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to before connecting to target.

localport()

Type:	number
Default:	0

Description: The port number to bind to. Messages are sent from this port.

port() or destport()

Type:	number
Default:	514

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so_broadcast()

Type:	yes or no
Default:	no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. See the `socket(7)` manual page for details.

so_keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. See the `socket(7)` manual page for details.

so_rcvbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. See the `socket(7)` manual page for details.

so_sndbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket send buffer in bytes. See the `socket(7)` manual page for details.

spoof_source()

Type:	yes or no
Default:	no

Description: 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` configuration option.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the `Last message repeated n times.` message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 6.5, Macros (p. 154)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like `syslogd` or syslog-ng itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping `'` and `"` in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations. TLS can be used only with the `tcp` transport protocols. See *Section 6.10, TLS options (p. 177)* for more information.

ts_format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. See also *Section 5.7, A note on timezones and timestamps (p. 90)*.

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

These drivers send 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]);
```

The `unix-stream()` and `unix-dgram()` destinations have the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the driver.

The `no-multi-line` flag disables line-breaking in the messages; the entire message is converted to a single line.

The `syslog-protocol` flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the `syslog` driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the *flush_lines* option for more information.

frac_digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the *keep-alive* option is enabled for the source. When the *keep-alive* option is enabled, syslog-ng saves the contents of the output queue of the destination when receiving a HUP signal, reducing the risk of losing messages

so_broadcast()

Type:	yes or no
Default:	no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. See the `socket(7)` manual page for details.

so_keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. See the `socket(7)` manual page for details.

so_rcvbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. See the `socket(7)` manual page for details.

so_sndbuf()

Type:	number
Default:	0

Description: Specifies the size of the socket send buffer in bytes. See the `socket(7)` manual page for details.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *`Last message repeated n times.`* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 6.5, Macros (p. 154)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *`syslogd`* or syslog-ng itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping ' and " in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. See also *Section 5.7, A note on timezones and timestamps (p. 90)*.

Example 6.20. Using the unix-stream() driver

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

6.2.8. 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. Use the asterisk `*` to specify every user currently logged in to the system.

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

The `usertty()` does not have any further options nor does it support templates.

Example 6.21. Using the usertty() driver

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

6.3. Log path flags

Flags influence the behavior of syslog-ng, and the way it processes messages. The following flags may be used in the log paths, as described in *Section 4.5, Log paths (p. 62)*.

Flag	Description
catchall	This flag means that the source of the message is ignored, only the filters are taken into account when matching messages. A log statement using the `catchall` flag processes every message that arrives to any of the defined sources.
fallback	This flag makes a log statement 'fallback'. Fallback log statements process messages that were not processed by other, 'non-fallback' log statements.
final	This flag means that the processing of log messages processed by the log statement ends here, other log statements appearing later in the configuration file will not process the messages processed by the log statement labeled as 'final'. Note that this does not necessarily mean that matching messages will be stored only once, as there can be matching log statements processed prior the current one.
f l o w - control	Enables flow-control to the log path, meaning that syslog-ng will stop reading messages from the sources of this log statement if the destinations are not able to process the messages at the required speed. If disabled, syslog-ng will drop messages if the destination queues are full. If enabled, syslog-ng will only drop messages if the destination queues/window sizes are improperly sized.

Table 6.1. Log statement flags

Warning
The `final`, `fallback`, and `catchall` flags apply only for the top-level log paths, they have no effect on embedded log paths.

Example 6.22. Using log path flags
Let's suppose that you have two hosts (`myhost_A` and `myhost_B`) that run two applications each (`application_A` and `application_B`), and you collect the log messages to a central syslog-ng server. On the server, you create two log paths:

- one that processes only the messages sent by `myhost_A`; and
- one that processes only the messages sent by `application_A`.

This means that messages sent by `application_A` running on `myhost_A` will be processed by both log paths, and the messages of `application_B` running on `myhost_B` will not be processed at all.

- If you add the `final` flag to the first log path, then only this log path will process the messages of `myhost_A`, so the second log path will receive only the messages of `application_A` running on `myhost_B`.
- If you create a third log path that includes the `fallback` flag, it will process the messages not processed by the first two log paths, in this case, the messages of `application_B` running on `myhost_B`.
- Adding a fourth log path with the `catchall` flag would process every message received by the syslog-ng server.

```
log { source(s_localhost); destination(d_file); flags(catchall); };
```

6.4. Filter functions

The following functions may be used in the filter statement, as described in *Section 4.6, Filters (p. 66)*.

facility()

Synopsis:	facility(facility[,facility])

Description: Match messages having one of the listed facility code. An alternate syntax permits the use an arbitrary facility codes.

The `facility()` filter accepts both the name and the numerical code of the facility or the importance level. The syslog-ng application recognizes the following facilities: (Note that some of these facilities are available only on specific platforms.)

Numerical Code	Facility name	Facility
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news subsystem
8	uucp	UUCP subsystem
9	cron	clock daemon
10	auth	security/authorization messages
11	ftp	FTP daemon
12		NTP subsystem
13		log audit
14		log alert
15	cron	clock daemon
16-23	local0..local7	locally used facilities (local0-local7)

Table 6.2. syslog Message Facilities recognized by the facility() filter

facility()

Synopsis:	facility(<numeric facility code>)

Description: An alternate syntax for `facility` permitting the use of an arbitrary facility code. Facility codes 0-23 are predefined and can be referenced by their usual name. Facility codes above 24 are not defined but can be used by this alternate syntax.

filter()

Synopsis:	filter(filtername)

Description: Call another filter rule and evaluate its value.

host()

Synopsis:	host(regexp)

Description: Match messages by using a regular expression against the hostname field of log messages.

level() or priority()

Synopsis:	level(pri[,pri1..pri2[,pri3]])

Description: Match messages based on priority.

The `level()` filter accepts the following levels: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`.

match()

Synopsis:	match(regexp)

Description: Match a regular expression to the headers and the message itself (i.e., the values returned by the `MSGHDR` and `MSG` macros). Note that in syslog-ng version 2.1 and earlier, the `match()` filter was applied only to the text of the message, excluding the headers. This functionality has been moved to the `message()` filter. To limit the scope of the match to a specific part of the message (identified with a macro), use the `match(regexp value("MACRO"))` syntax. Do not include the $ sign in the parameter of the `value()` option.

message()

Synopsis:	message(regexp)

Description: Match a regular expression to the text of the log message, excluding the headers (i.e., the value returned by the `MSG` macros). Note that in syslog-ng version 2.1 and earlier, this functionality was performed by the `match()` filter.

netmask()

Synopsis:	netmask(ip/mask)

Description: Select only messages sent by a host whose IP address belongs to the specified IP subnet. Note that this filter checks the IP address of the last-hop relay (the host that actually sent the message to syslog-ng), not the contents of the `HOST` field of the message.

program()

Synopsis:	program(regexp)

Description: Match messages by using a regular expression against the program name field of log messages.

source()

Synopsis: source id

Description: Select messages of a source statement. This filter can be used in embedded log statements if the parent statement contains multiple source groups — only messages originating from the selected source group are sent to the destination of the embedded log statement.

tags()

Synopsis: tag

Description: Select messages labeled with the specified tag. Every message automatically has the tag of its source in `.source.<id_of_the_source_statement>` format. This option is available only in syslog-ng 3.1 and later.

Example 6.23. Adding tags and filtering messages with tags

```
source s_tcp {
tcp(ip(192.168.1.1) port(1514) tags("tcp", "router"));
};
```

Use the `tags()` option of the filters to select only specific messages:

```
filter f_tcp {
   tags(".source.s_tcp");
};

filter f_router {
   tags("router");
};
```

6.4.1. Using regular expressions in filters

The `host()`, `match()`, and `program()` filter functions accept regular expressions as parameters. The exact type of the regular expression to use can be specified with the `type()` option. The following expression types are available:

posix

Description: Use POSIX regular expressions. If the `type()` parameter is not specified, syslog-ng uses POSIX regular expressions by default. For additional details on the use and flags of regular expressions, see *Section 6.8, Regular expressions (p. 168)*.

pcre

Description: Use PCRE regular expressions. Execute the `syslog-ng -V` command to check if your binary supports PCRE regular expressions. Starting with syslog-ng OSE version 3.1, PCRE expressions are supported on every platform. For additional details on the use and flags of regular expressions, see *Section 6.8, Regular expressions (p. 168)*.

string

Description: Match the strings literally, without regular expression support. By default, only identical strings are matched. For partial matches, use the `flags("prefix")` or the `flags("substring")` flags.

6.5. Macros

Certain parts of syslog-ng (for example destination filenames and message content templates) can refer to one or more macros, which get expanded as a message is processed. The table below summarizes the macros available in syslog-ng.

Macros can be included by prefixing the macro name with a `$` sign, just like in Bourne compatible shells. Regarding braces around macro names, the following two formats are equivalent `"$MSG"` and `"${MSG}"`.

Default values for macros can also be specified by appending the `:-` characters and the default value to the macro, for example

```
${HOST:-default_hostname}
```

Macros can be used to format messages, and also in the name of destination files. However, they cannot be used in sources as wildcards, for example, to read messages from files or directories that include a date in their name.

BSDTAG

Description: Facility/priority information in the format used by the FreeBSD syslogd: a priority number followed by a letter that indicates the facility. The priority number can range from `0` to `7`. The facility letter can range from `A` to `Y`, where `A` corresponds to facility number zero (LOG_KERN), `B` corresponds to facility 1 (LOG_USER), etc.

DATE, R_DATE, S_DATE

Description: Date of the message using the BSD-syslog style timestamp format (month/day/hour/minute/second, each expressed in two digits). This is the original syslog time stamp without year information, for example: `Jun 13 15:58:00`.

DAY, R_DAY, S_DAY

Description: The day the message was sent.

FACILITY

Description: The name of the facility (for example, `kern`) that sent the message.

FACILITY_NUM

Description: The numerical code of the facility (for example, `0`) that sent the message.

FULLDATE, R_FULLDATE, S_FULLDATE

Description: A nonstandard format for the date of the message using the same format as `DATE`, but including the year as well, for example: `2006 Jun 13 15:58:00`.

FULLHOST

Description: The full FQDN of the host name chain (without trimming chained hosts), including the domain name. To use this macro, make sure that the `keep_hostname()` option is enabled.

FULLHOST_FROM
Description: FQDN of the host that sent the message to syslog-ng as resolved by syslog-ng using DNS. If the message traverses several hosts, this is the last host in the chain. To use this macro, make sure that the *keep_hostname()* option is enabled.

HOUR, R_HOUR, S_HOUR
Description: The hour of day the message was sent.

HOST
Description: The name of the source host where the message originates from. If the message traverses several hosts and the *_chain_hostnames()* option is on, the first host in the chain is used. To use this macro, make sure that the *keep_hostname()* option is enabled.

HOST_FROM
Description: Name of the host that sent the message to syslog-ng, as resolved by syslog-ng using DNS. If the message traverses several hosts, this is the last host in the chain. To use this macro, make sure that the *keep_hostname()* option is enabled.

ISODATE, R_ISODATE, S_ISODATE
Description: Date of the message in the ISO 8601 compatible standard timestamp format (yyyy-mm-ddThh:mm:ss+-ZONE), for example: *2006-06-13T15:58:00.123+01:00*. If possible, it is recommended to use *ISODATE* for timestamping. Note that syslog-ng can produce fractions of a second (for example milliseconds) in the timestamp by using the *frac_digits()* global or per-destination option.

LEVEL_NUM
Description: The priority (also called severity) of the message, represented as a numeric value, for example, *3*.

MIN, R_MIN, S_MIN
Description: The minute the message was sent.

MONTH, R_MONTH, S_MONTH
Description: The month the message was sent as a decimal value, prefixed with a zero if smaller than 10.

MONTH_ABBREV, R_MONTH_ABBREV, S_MONTH_ABBREV
Description: The English abbreviation of the month name (3 letters).

MONTH_NAME, R_MONTH_NAME, S_MONTH_NAME
Description: The English name of the month name.

MONTH_WEEK, R_MONTH_WEEK, S_MONTH_WEEK
Description: The number of the week in the given month (0-5). The week with numerical value 1 is the first week containing a Monday. The days of month before the first Monday are considered week 0. For example, if a 31-day month begins on a Sunday, then the 1st of the month is week 0, and the end of the month (the 30th and 31st) is week 5.

MSG or MESSAGE

Description: Text contents of the log message without the program name and pid. Note that this has changed in syslog-ng version 3.0; in earlier versions this macro included the program name and the pid. In syslog-ng 3.0, the `MSG` macro became equivalent with the `MSGONLY` macro. The program name and the pid together are available in the `MSGHDR` macro.

MSGHDR

Description: The name and the pid of the program that sent the log message in `PROGRAM: PID` format. Includes a trailing whitespace. Note that the macro returns an empty value if both the program and pid fields of the message are empty.

MSGONLY

Description: Message contents without the program name or pid.

PID

Description: The PID of the program sending the message.

PRI

Description: The priority and facility encoded as a 2 or 3 digit decimal number as it is present in syslog messages.

PRIORITY or LEVEL

Description: The priority (also called severity) of the message, for example, `error`.

PROGRAM

Description: The name of the program sending the message. Note that the content of the $PROGRAM variable may not be completely trusted as it is provided by the client program that constructed the message.

SDATA, .SDATA.SDID.SDNAME

Description: The syslog-ng application automatically parses the STRUCTURED-DATA part of IETF-syslog messages, which can be referenced in macros. The `$SDATA` macro references the entire STRUCTURED-DATA part of the message, while structured data elements can be referenced using the `$.SDATA.SDID.SDNAME` macro.

Note

When using STRUCTURED-DATA macros, consider the following:

- When referencing an element of the structured data, the macro must begin with the dot (.) character. For example, `$.SDATA.timeQuality.isSynced`.
- The SDID and SDNAME parts of the macro names are case sensitive: `$.SDATA.timeQuality.isSynced` is not the same as `$.SDATA.TIMEQUALITY.ISSYNCED`.

Example 6.24. Using SDATA macros

For example, if a log message contains the following structured data: `[exampleSDID@0 iut="3" eventSource="Application" eventID="1011"][examplePriority@0 class="high"]` you can use macros like: `${.SDATA.exampleSDID@0.eventSource}` — this would return the `Application` string in this case.

SEC, R_SEC, S_SEC
Description: The second the message was sent.

SEQNUM
Description: The sequence number of the message is a unique identifier of the message between the end-points. The syslog-ng client calculates this number when processing a new message from a local source; it is not calculated for relayed messages. The sequence number increases for every message, and is not lost even if syslog-ng is reloaded or restarted. The sequence number is a part of every message that uses the new IETF-syslog protocol (`.SDATA.meta.sequenceId`), and can be added to BSD-syslog messages using this macro.

SOURCEIP
Description: IP address of the host that sent the message to syslog-ng. (I.e. the IP address of the host in the `FULLHOST_FROM` macro.) Please note that when a message traverses several relays, this macro contains the IP of the last relay.

STAMP, R_STAMP, S_STAMP
Description: A timestamp formatted according to the `ts_format()` global or per-destination option.

TAG
Description: The priority and facility encoded as a 2 digit hexadecimal number.

TAGS
Description: A comma-separated list of the tags assigned to the message.

Note
Note that the tags are not part of the log message and are not automatically transferred from a client to the server. For example, if a client uses a pattern database to tag the messages, the tags are not transferred to the server. A way of transferring the tags is to explicitly add them to the log messages using a template and the `$TAGS` macro, or to add them to the structured metadata part of messages when using the IETF-syslog message format.

When sent as structured metadata, it is possible to reference to the list of tags on the central server, and for example, to add them to a database column.

TZ, R_TZ, S_TZ
Description: Equivalent to TZOFFSET, used to mean the time zone name abbreviation in syslog-ng 1.6.x.

TZOFFSET, R_TZOFFSET, S_TZOFFSET
Description: The time-zone as hour offset from GMT; for example: `-07:00`. In syslog-ng 1.6.x this used to be `-0700` but as `ISODATE` requires the colon it was added to `TZOFFSET` as well.

UNIXTIME, R_UNIXTIME, S_UNIXTIME
Description: Standard unix timestamp, represented as the number of seconds since `1970-01-01T00:00:00`.

YEAR, R_YEAR, S_YEAR
Description: The year the message was sent.

WEEK, R_WEEK, S_WEEK

Description: The week number of the year, prefixed with a zero for the first nine week of the year. (The first Monday in the year marks the first week.)

WEEK_ABBREV, R_WEEK_ABBREV, S_WEEK_ABBREV

Description: The 3-letter English abbreviation of the name of the day the message was sent, for example `Thu`.

WEEK_DAY, R_WEEK_DAY, S_WEEK_DAY

Description: The day of the week as a numerical value (1-7).

WEEKDAY, R_WEEKDAY, S_WEEKDAY

Description: These macros are deprecated, use *WEEK_ABBREV, R_WEEK_ABBREV, S_WEEK_ABBREV* instead. The 3-letter name of the day of week the message was sent, for example `Thu`.

WEEK_DAY_NAME, R_WEEK_DAY_NAME, S_WEEK_DAY_NAME

Description: The English name of the day.

6.6. Message parsers

The following sections provide reference for the parsers available in syslog-ng.

- To segment structured messages like comma-separated values, see *Section 6.6.1, CSV parsers (p. 158)*.
- To classify messages using a pattern database, see *Section 6.6.2, Pattern databases (p. 161)*.

6.6.1. CSV parsers

The syslog-ng application can separate parts of log messages (i.e., the contents of the $MSG macro) to named fields (columns). These fields act as user-defined macros that can be referenced in message templates, file- and tablenames, etc.

To create a parser, define the columns of the message, the delimiter or separator characters, and optionally the characters that are used to escape the delimiter characters (quote-pairs).

```
Declaration:
    parser parser_name {
        csv-parser(column1, column2, ...)
        delimiters()
        quote-pairs()
    };
```

Column names work like macros. Always use a prefix to identify the columns of the parsers, for example `MYPARSER1.COLUMN1, MYPARSER2.COLUMN2`, etc. Column names starting with a dot (for example `.HOST`) are reserved for use by syslog-ng.

csv-parser

Synopsis:	csv-parser(columns("PARSER.COLUMN1", "PARSER.COLUMN2", ...))

Description: Specifies the type of parser to use, and the name of the columns to separate messages to. Currently only the *csv-parser* is implemented, which can separate columns based on delimiter characters and strings.

delimiters

Synopsis:	delimiters("<delimiter_characters>")

Description: The character that separates the columns in the message.

flags()

Synopsis:	drop-invalid, escape-none, escape-backslash, escape-double-char, greedy, strip-whitespace

Description: When the *drop-invalid* option is set, the parser does not process messages that have less columns than defined in the parser. Using this option practically turns the parser into a special filter, that matches messages that have the predifined number of columns (using the specified delimiters).

The *escape-none, escape-backslash, escape-double-char* flags set the escaping rules used by the parser.

The *greedy* option assigns the remainder of the message to the last column, regardless of the delimiter characters set. You can use this option to process messages where the number of columns varies.

The *strip-whitespace* flag removes trailing whitespaces from the beginning and the end of the columns.

quote-pairs()

Synopsis:	quote-pairs('<quote_pairs>')

Description: List quote-pairs between single quotes. Delimiter characters enclosed between quote characters are ignored. Note that the beginning and ending quote character does not have to be identical, for example *[]* can also be a quote-pair.

template()

Synopsis:	template("${<macroname>}")

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, this is empty and the parser processes the entire message.

Example 6.25. Segmenting hostnames separated with a dash
The following example separates hostnames like *example-1* and *example-2* into two parts.

```
parser p_hostname_segmentation {
    csv-parser(columns("HOSTNAME.NAME", "HOSTNAME.ID")
    delimiters("-")
    flags(escape-none)
    template("${HOST}"));
};
destination d_file { file("/var/log/messages-${HOSTNAME.NAME:-examplehost}"); };
log { source(s_local); parser(p_hostname_segmentation); destination(d_file);};
```

Example 6.26. Parsing Apache log files
The following parser processes the log of Apache web servers and separates them into different fields. Apache log messages can be formatted like:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %T %v"
```

Here is a sample message:

```
192.168.1.1 - - [31/Dec/2007:00:17:10 +0100] "GET /cgi-bin/example.cgi HTTP/1.1" 200 2708
 "-" "curl/7.15.5 (i4 86-pc-linux-gnu) libcurl/7.15.5 OpenSSL/0.9.8c zlib/1.2.3 libidn/0.6.5"
 2 example.balabit
```

To parse such logs, the delimiter character is set to a single whitespace (*delimiters(" ")*). Whitespaces between quotes and brackets are ignored (*quote-pairs('""[]')*).

```
parser p_apache {
    csv-parser(columns("APACHE.CLIENT_IP", "APACHE.IDENT_NAME", "APACHE.USER_NAME",
        "APACHE.TIMESTAMP", "APACHE.REQUEST_URL", "APACHE.REQUEST_STATUS",
        "APACHE.CONTENT_LENGTH", "APACHE.REFERER", "APACHE.USER_AGENT",
        "APACHE.PROCESS_TIME", "APACHE.SERVER_NAME")
         flags(escape-double-char,strip-whitespace)
         delimiters(" ")
         quote-pairs('""[]')
         );
};
```

The results can be used for example to separate log messages into different files based on the APACHE.USER_NAME field. If the field is empty, the *nouser* name is assigned.

```
log { source(s_local);
    parser(p_apache); destination(d_file);};
};
destination d_file { file("/var/log/messages-${APACHE.USER_NAME:-nouser}"); };
```

Example 6.27. Segmenting a part of a message
The following example splits the timestamp of a parsed Apache log message into separate fields.

```
parser p_apache_timestamp {
    csv-parser(columns("APACHE.TIMESTAMP.DAY", "APACHE.TIMESTAMP.MONTH",
"APACHE.TIMESTAMP.YEAR", "APACHE.TIMESTAMP.HOUR", "APACHE.TIMESTAMP.MIN",
"APACHE.TIMESTAMP.MIN", "APACHE.TIMESTAMP.ZONE")
    delimiters("/: ")
    flags(escape-none)
    template("${APACHE.TIMESTAMP}"));
    };
log { source(s_local);
    log { parser(p_apache); parser(p_apache_timestamp); destination(d_file);};
};
```

Example 6.28. Adding the end of the message to the last column
If the *greedy* option is enabled, the syslog-ng application adds the not-yet-parsed part of the message to the last column, ignoring any delimiter characters that may appear in this part of the message.

For example, you receive the following comma-separated message: *example 1, example2, example3*, and you segment it with the following parser:

```
csv_parser(columns("COLUMN1", "COLUMN2", "COLUMN3") delimiters(","));
```

The *COLUMN1*, *COLUMN2*, and *COLUMN3* variables will contain the strings *example1*, *example2*, and *example3*, respectively. If the message looks like *example 1, example2, example3, some more information*, then any text appearing after the third comma (i.e., *some more information*) is not parsed, and possibly lost if you use only the variables to re-construct the message (for example, to send it to different columns of an SQL table).

Using the *greedy* flag will assign the remainder of the message to the last column, so that the *COLUMN1*, *COLUMN2*, and *COLUMN3* variables will contain the strings *example1*, *example2*, and *example3, some more information*.

```
csv_parser(columns("COLUMN1", "COLUMN2", "COLUMN3") delimiters(",") flags(greedy));
```

6.6.2. Pattern databases

6.6.2.1. Using pattern parsers

Pattern parsers attempt to parse a part of the message using rules specific to the type of the parser. Parsers are enclosed between @ characters. The syntax of parsers is the following:

- a beginning `@` character;
- the type of the parser written in capitals;
- optionally a name;
- parameters of the parser, if any;
- a closing `@` character.

Example 6.29. Pattern parser syntax
A simple parser:

```
@STRING@
```

A named parser:

```
@STRING:myparser_name@
```

A named parser with a parameter:

```
@STRING:myparser_name:*@
```

A parser with a parameter, but without a name:

```
@STRING::*@
```

The following parsers are available:

- *@ANYSTRING@*: Parses everything to the end of the message; you can use it to collect everything that is not parsed specifically to a single macro. In that sense its behavior is similar to the `greedy()` option of the CSV parser.
- *@DOUBLE@*: An obsolete alias of the `@FLOAT@` parser.
- *@ESTRING@*: This parser has a required parameter that acts as the stopcharacter: the parser parses everything until it finds the stopcharacter. For example to stop by the next `"` (double quote) character, use `@ESTRING::"@`. As of syslog-ng 3.1, it is possible to specify a stopstring instead of a single character, e.g., `@ESTRING::stop_here.@`.
- *@FLOAT@*: A floating-point number that may contain a dot (.) character. (Up to syslog-ng 3.1, the name of this parser was `@DOUBLE@`.)
- *@IPv4@*: Parses an IPv4 IP address (numbers separated with a maximum of 3 dots).
- *@IPv6@*: Parses any valid IPv6 IP address.
- *@IPvANY@*: Parses any IP address.
- *@NUMBER@*: A sequence of decimal (0-9) numbers (e.g., 1, 0687, etc.). Note that if the number starts with the 0x characters, it is parsed as a hexadecimal number, but only if at least one valid character follows 0x.

- *@QSTRING@*: Parse a string between the quote characters specified as parameter. Note that the quote character can be different at the beginning and the end of the quote, e.g.: `@QSTRING::"@` parses everything between two quotation marks (`"`), while `@QSTRING:<>@` parses from an opening bracket to the closing bracket.
- *@STRING@*: A sequence of alphanumeric characters (0-9, A-z), not including any whitespace. Optionally, other accepted characters can be listed as parameters (e.g., to parse a complete sentence, add the whitespace as parameter, like: `@STRING:: @`). Note that the `@` character cannot be a parameter, nor can line-breaks or tabs.

Patterns and literals can be mixed together. For example, to parse a message that begins with the `Host:` string followed by an IP address (e.g., `Host: 192.168.1.1`), the following pattern can be used: `Host:@IPv4@`.

Note
Note that using parsers is a CPU-intensive operation. Use the ESTRING and QSTRING parsers whenever possible, as these can be processed much faster than the other parsers.

Example 6.30. Using the STRING and ESTRING parsers
For example, if the message is `user=joe96 group=somegroup`, `@STRING:mytext:@` parses only to the first non-alphanumeric character (=), parsing only `user`. `@STRING:mytext:=@` parses the equation mark as well, and proceeds to the next non-alphanumeric character (the whitespace), resulting in `user=joe96` being parsed. `@STRING:mytext:= @` will parse the whitespace as well, and proceed to the next non-alphanumeric non-equation mark non-whitespace character, resulting in `user=joe96 group=somegroup`.

Of course, usually it is better to parse the different values separately, like this: `"user=@STRING:user@ group=@STRING:group@"`.

If the username or the group may contain non-alphanumeric characters, you can either include these in the second parameter of the parser (as shown at the beginning of this example), or use an ESTRING parser to parse the message till the next whitespace: `"user=@ESTRING:user: @group=@ESTRING:group: @"`.

6.6.2.2. Filtering messages based on classification

The results of message classification and parsing can be used in custom filters and file and database templates as well. There are two built-in macros in syslog-ng OSE that allow you to use the results of the classification: the `.classifier.class` macro contains the class assigned to the message (for example violation, security, or unknown), while the `.classifier.rule_id` macro contains the identifier of the message pattern that matched the message.

Example 6.31. Using classification results for filtering messages
To filter on a specific message class, create a filter that checks the **.classifier_class** macro, and use this filter in a log statement.

```
filter fi_class_violation {
                        match("violation"
                        value(".classifier.class")
                        type("string")
                        );
                        };
```

```
log {
                        source(s_all);
                        parser(pattern_db);
                        filter(fi_class_violation);
                        destination(di_class_violation);
                        };
```

Filtering on the *unknown* class selects messages that did not match any rule of the pattern database. Routing these messages into a separate file allows you to periodically review new or unknown messages.

To filter on messages matching a specific classification rule, create a filter that checks the **.classifier_rule_id** macro. The unique identifier of the rule (for example *e1e9c0d8-13bb-11de-8293-000c2922ed0a*) is the *id* attribute of the rule in the XML database.

```
filter fi_class_rule {
                        match("e1e9c0d8-13bb-11de-8293-000c2922ed0a"
                        value(".classifier_rule_id")
                        type("string")
                        );
                        };
```

The message-segments parsed by the pattern parsers can also be used as macros as well. To accomplish this, you have to add a name to the parser, and then you can use this name as a macro that refers to the parsed value of the message.

Example 6.32. Using pattern parsers as macros
For example, you want to parse messages of an application that look like *"Transaction: <type>."*, where <type> is a string that has different values (for example refused, accepted, incomplete, etc.). To parse these messages, you can use the following pattern:

```
'Transaction: @ESTRING::.@'
```

Here the @ESTRING@ parser parses the message until the next full stop character. To use the results in a filter or a filename template, include a name in the parser of the pattern, for example:

```
'Transaction: @ESTRING:TRANSACTIONTYPE:.@'
```

After that, add a custom template to the logpath that uses this template. For example, to select every *accepted* transaction, use the following custom filter in the log path:

```
match("accepted" value("TRANSACTIONTYPE"));
```

Note
The above macros can be used in database columns and filename templates as well, if you create custom templates for the destination or logspace.

Use a consistent naming scheme for your macros, for example, *APPLICATIONNAME_MACRONAME*.

6.6.2.3. Creating pattern databases

Pattern databases are XML files that contain rules describing the message patterns. For sample pattern databases, see *Section 4.9.1, Downloading sample pattern databases (p. 73)*.

What's new in the syslog-ng pattern database format V3

The V3 database format has the following differences compared to the original V1 format:

- The rules that are applied to the messages of a program can be separated into multiple rulesets.
- The program pattern of the rulesets can be empty; such rulesets act as fallback rulesets that are applied to the log messages if no program pattern is matching or when a message does not have a program part.
- Rules can contain multiple patterns to cover messages that have multiple formats (for example multilingual messages).

- Tags can be defined in the rules; these tags are automatically assigned to messages matching the patterns of the rule.
- Static named values can be defined in the rules; these are automatically assigned to messages matching the patterns of the rule. The assigned values can be used in filters and macros.
- It is also possible to include sample messages in the rules, and also the expected values of the parsers. These can be used to test the behavior of the patterns.

The syslog-ng pattern database format

The following scheme describes the V3 format of the pattern database. This format is used by syslog-ng 3.1 and later, and the syslog-ng Store Box (SSB) appliance version 1.1 and later (see the *syslog-ng Store Box web page* for details on SSB).

For a sample database containing only a single pattern, see *Example 6.33, A V3 pattern database containing a single rule (p. 166)*.

For earlier versions of the syslog-ng pattern database formats, see *Appendix 3, Deprecated pattern database schemes (p. 201)*.

For a summary of differences between the different syslog-ng pattern database formats, see *Section What's new in the syslog-ng pattern database format V3 (p. 163)*.

Tip
Use the `pdbtool` utility that is bundled with syslog-ng to test message patterns and convert existing databases to the latest format. See *pdbtool(1) (p. 187)* for details.

- **<patterndb>**: The container element of the pattern database. For example:

```
<patterndb version='3' pub_date='2009-10-25'>
```

- *version*: The schema version of the pattern database. The current version is `3`.
- *pubdate*: The publication date of the XML file.
- **<ruleset>**: A container element to group log patterns for an application or program. For example:

```
<ruleset name='su' id='480de478-d4a6-4a7f-bea4-0c0245d361e1'>
```

A `<patterndb>` element may contain any number of **<ruleset>** elements.

- *name*: The name of the application. Note that the function of this attribute is to make the database more readable, syslog-ng uses the `<pattern>` element to identify the applications sending log messages.
- *id*: A unique ID of the application, for example, the md5 sum of the `name` attribute.
- **description**: OPTIONAL — A description of the ruleset or the application.
- **url**: OPTIONAL — An URL referring to further information about the ruleset or the application.

- **pattern**: The name of the application — syslog-ng matches this value to the $PROGRAM header of the syslog message to find the rulesets applicable to the syslog message. This element is also called *program pattern*. For example

```
<pattern>su</pattern>
```

Note
If the *<pattern>* element of a ruleset is not specified, -ng will use this ruleset as a fallback ruleset: it will apply the ruleset to messages that have an empty PROGRAM header, or if none of the program patterns matched the PROGRAM header of the incoming message.

- **<rules>**: A container element for the rules of the ruleset.
  - **<rule>**: An element containing message patterns and how a message that matches these patterns is classified. For example:

```
<rule provider='balabit'

id='f57196aa-75fd-11dd-9bba-001e6806451b'
                                                class='violation'>
```

Note
If the following characters appear in the message, they must be escaped in the rule as follows:
- *@*: Use @@, for example *user@@example.com*
- <: Use *<*
- >: Use *>*
- &: Use *&*

The **<rules>** element may contain any number of **<rule>** elements.

- *provider*: The provider of the rule. This is used to distinguish between who supplied the rule; i.e., if it has been created by BalaBit, or added to the xml by a local user.
- *id*: The globally unique ID of the rule.
- *class*: The class of the rule — syslog-ng assigns this class to the messages matching a pattern of this rule.
- **<patterns>**: An element containing the patterns of the rule. If a **<patterns>** element contains multiple **<pattern>** elements, the class of the **<rule>** is assigned to every syslog message matching any of the patterns.
  - **<pattern>**: A pattern describing a log message. This element is also called *message pattern*. For example:

```
<pattern>+ ??? root-</pattern>
```

  - **description**: OPTIONAL — A description of the pattern or the log message matching the pattern.

- **urls**: OPTIONAL — An element containing one or more URLs referring to further information about the patterns or the matching log messages.
  - **url**: OPTIONAL — An URL referring to further information about the patterns or the matching log messages.
- **values**: OPTIONAL — Name-value pairs that are assigned to messages matching the patterns, for example, the representation of the event described in the message in Common Event Format (CEF). The names can be used as macros to reference the assigned values.
  - **value**: OPTIONAL — Contains the value of the name-value pair that is assigned to the message. For example:

    ```
    <value name=".classifier.outcome">/Success</value>
    ```

  - *name*: The name of the name-value pair. It can also be used as a macro to reference the assigned value.
- **examples**: OPTIONAL — A container element for sample log messages that should be recognized by the pattern. These messages can be used also to test the patterns and the parsers.
  - **example**: OPTIONAL — A container element for a sample log message.
    - **test_message**: OPTIONAL — A sample log message that should match this pattern. For example:

      ```
      <test_message>Content filter has been enabled</test_message>
      ```

    - **test_values**: OPTIONAL — A container element to test the results of the parsers used in the pattern.
      - **test_value**: OPTIONAL — The expected value of the parser when matching the pattern to the test message. For example:

        ```
        <test_value name=".dict.ContentFilter">enabled</test_value>
        ```

      - *name*: The name of the parser to test.
- **tags**: OPTIONAL — An element containing custom keywords (tags) about the messages matching the patterns. The tags can be used to label specific events (for example user logons). It is also possible to filter on these tags later (see *Section 4.6.3, Tagging messages (p. 69)* for details). Starting with syslog-ng Open Source Edition 3.2, the list of tags assigned to a message can be referenced with the `$TAGS` macro.
  - **tag**: OPTIONAL — A keyword or tags applied to messages matching the rule. For example:

    ```
    <tags><tag>UserLogin</tag></tags>
    ```

Example 6.33. A V3 pattern database containing a single rule
The following pattern database contains a single rule that matches a log message of the `ssh` application. A sample log message looks like:

```
Accepted password for sampleuser from 10.50.0.247 port 42156 ssh2
```

The following is a simple pattern database containing a matching rule.

```
<patterndb version='3' pub_date='2009-04-17'>
    <ruleset name='ssh' id='123456678'>
        <pattern>ssh</pattern>
            <rules>
                <rule provider='me' id='182437592347598' class='system'>
                    <patterns>
                        <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @
for@QSTRING:SSH_USERNAME: @from\ @QSTRING:SSH_CLIENT_ADDRESS: @port @NUMBER:SSH_PORT_NUMBER:@
 ssh2</pattern>
                    </patterns>
                </rule>
            </rules>
    </ruleset>
</patterndb>
```

Note that the rule uses macros that refer to parts of the message, for example, you can use the *$SSH_USERNAME* macro refer to the username used in the connection.

The following is the same example, but with a test message and test values for the parsers.

```
<patterndb version='3' pub_date='2009-04-17'>
    <ruleset name='ssh' id='123456678'>
        <pattern>ssh</pattern>
            <rules>
                <rule provider='me' id='182437592347598' class='system'>
                    <patterns>
                        <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @
for@QSTRING:SSH_USERNAME: @from\ @QSTRING:SSH_CLIENT_ADDRESS: @port @NUMBER:SSH_PORT_NUMBER:@
 ssh2</pattern>
                    </patterns>
                    <examples>
                        <example>
                          <test_message>Accepted password for sampleuser from 10.50.0.247
 port 42156 ssh2</test_message>
                            <test_values>
                                <test_value name="SSH.AUTH_METHOD">password</test_value>
                                <test_value name="SSH_USERNAME">sampleuser</test_value>
                                <test_value
name="SSH_CLIENT_ADDRESS">10.50.0.247</test_value>
                                <test_value name="SSH_PORT_NUMBER">42156</test_value>
                            </test_values>
                        </example>
                    </examples>
                </rule>
            </rules>
    </ruleset>
</patterndb>
```

6.7. Rewriting messages

The syslog-ng application can rewrite parts of log messages: it can search and replace text, and also set a specific field to a specified value. Rewriting messages is often used in conjunction with message parsing *Section 6.6, Message parsers (p. 158)*.

To create replace a part of the log message, define the string or regular expression to replace, the string to replace the original text (macros can be used as well), and the field of the message that the rewrite rule should process. Substitution rules can operate on any value available via macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (see *Section 6.6, Message parsers (p. 158)* for details). The only exceptions are the FACILITY, SEVERITY, TAGS, and the date-related fields, which cannot be rewritten.

As of syslog-ng 3.1, it is also possible to rewrite the structured-data fields of messages complying to the RFC5424 (IETF-syslog) message format. Substitution rules use the following syntax:

```
Declaration:
    rewrite <name_of_the_rule>
    {subst("<string or regular expression to find>", "<replacement string>",
value(<field name>) type() flags());};
```

The *type()* and *flags()* options are optional. The *type()* specifies the type of regular expression to use; while the *flags()* are the flags of the regular expressions. For details on regular expressions, see *Section 6.8, Regular expressions (p. 168)*.

Example 6.34. Using substitution rules
The following example replaces the first occurrence of the string *IP* in the text of the message with the string *IP-Address*.

```
rewrite r_rewrite_subst{subst("IP", "IP-Address", value("MESSAGE"));};
```

To replace every occurrence, use:

```
rewrite r_rewrite_subst{subst("IP", "IP-Address", value("MESSAGE"), flags("global"));};
```

Multiple substitution rules are applied sequentially; the following rules replace the first occurrence of the string *IP* with the string *IP-Addresses*.

```
rewrite r_rewrite_subst{subst("IP", "IP-Address", value("MESSAGE")); subst("Address",
"Addresses", value("MESSAGE"));};
```

To set a field of the message to a specific value, define the string to include in the message, and the field where it should be included. Setting a field can operate on any value available via macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (see *Section 6.6, Message parsers (p. 158)* for details). The only exceptions are the FACILITY, SEVERITY, TAGS, and the date-related fields, which cannot be rewritten. Note that this operation completely replaces any previous value of that field. Use the following syntax:

```
Declaration:
    rewrite <name_of_the_rule>
    {set("<string to include>", value(<field name>)  flags());};
```

Example 6.35. Setting message fields to a particular value
The following example sets the HOST field of the message to *myhost*.

```
rewrite r_rewrite_set{set("myhost", value("HOST"));};
```

The following example sets the sequence ID field of the RFC5424-formatted (IETF-syslog) messages to a fixed value.

```
rewrite r_sd { set("55555" value(".SDATA.meta.sequenceId")); };
```

It is also possible to set the value of a field that does not exist yet, and create a new name-value pair that is associated with the message. The following example created the *MODIFIED* field and sets its value to *yes*. If you use the *$MODIFIED* macro in a template or SQL table, its value will be *yes* for every message that was processed with this rewrite rule, and empty for every other message.

```
rewrite r_rewrite_set{set("yes", value("MODIFIED"));};
```

6.8. Regular expressions

Filters and substitution rewrite rules can use regular expressions. The regular expressions can use up to 255 regexp matches (*${1} ... ${255}*), but only from the last filter and only if the *flags("store-matches")* flag was set for the filter. For case-insensitive searches, use the *flags("ignore-case")* option.

By default, syslog-ng uses POSIX-style regular expressions. To use other expression types, add the `type()` option after the regular expression.

The syslog-ng OSE application supports the following expression types:

- *POSIX regular expressions*
- *Perl Compatible Regular Expressions (PCRE)*
- *Literal string searches*
- *Glob patterns without regular expression support*

posix

Description: Use POSIX regular expressions. If the `type()` parameter is not specified, syslog-ng uses POSIX regular expressions by default.

Posix regular expressions have the following flag options:

global: Usable only in rewrite rules; match for every occurrence of the expression, not only the first one.

ignore-case: Disable case-sensitivity.

store-matches: Store the matches of the regular expression into the `$1, ... $255` variables. Matches from the last filter expression can be referenced in regular expressions.

utf8: Use UTF-8 matching.

Example 6.36. Using Posix regular expressions

```
filter f_message { message("keyword" flags("utf8" "ignore-case") );
```

pcre

Description: Use Perl Compatible Regular Expressions (PCRE). PCRE expressions can be used if syslog-ng OSE was compiled with the `--enable-pcre` option enabled. Execute the `syslog-ng -V` command to check if your binary supports PCRE regular expressions. Starting with syslog-ng OSE version 3.1, PCRE expressions are supported on every platform.

PCRE regular expressions have the following flag options:

global: Usable only in rewrite rules; match for every occurrence of the expression, not only the first one.

ignore-case: Disable case-sensitivity.

nobackref: Do not store back references for the matches — improves performance.

store-matches: Store the matches of the regular expression into the `$1, ... $255` variables. Named matches (also called named subpatterns), for example `(?<name>...)`, are stored as well. Matches from the last filter expression can be referenced in regular expressions.

unicode: Use Unicode support for UTF-8 matches: UTF-8 character sequences are handled as single characters.

utf8: An alias for the *unicode* flag.

Example 6.37. Using PCRE regular expressions

```
rewrite r_rewrite_subst
        {subst("a*", "?", field("message") type("pcre") flags("utf8" "global"));  };
```

string

Description: Match the strings literally, without regular expression support. By default, only identical strings are matched. For partial matches, use the *flags("prefix")* or the *flags("substring")* flags.

glob

Description: Use glob patterns (that is, wildcards and character ranges) without regular expression support. The advantage of glob patterns to regular expressions is that globs can be processed much faster. For details on glob patterns, see the *glob manual page* (`man glob`).

6.9. Global options

The following options can be specified in the options statement, as described in *Section 4.11, Configuring global syslog-ng options (p. 76)*.

bad_hostname()

Accepted values:	regular expression
Default:	no

Description: A regexp containing hostnames which should not be handled as hostnames.

chain_hostnames()

Accepted values:	*yes* | *no*
Default:	*no*

Description: Enable or disable the chained hostname format.

check_hostname()

Accepted values:	*yes* | *no*
Default:	*no*

Description: Enable or disable checking whether the hostname contains valid characters.

create_dirs()

Accepted values:	*yes* | *no*
Default:	*no*

Description: Enable or disable directory creation for destination files.

dir_group()

Accepted values:	groupid
Default:	root

Description: The default group for newly created directories.

dir_owner()

Accepted values:	userid
Default:	root

Description: The default owner of newly created directories.

dir_perm()

Accepted values:	permission value
Default:	0700

Description: The default permission for newly created directories.

dns_cache()

Accepted values:	*yes* | *no*
Default:	*yes*

Description: Enable or disable DNS cache usage.

dns_cache_expire()

Accepted values:	number
Default:	3600

Description: Number of seconds while a successful lookup is cached.

dns_cache_expire_failed()

Accepted values:	number
Default:	60

Description: Number of seconds while a failed lookup is cached.

dns_cache_hosts()

Accepted values:	filename
Default:	unset

Description: Name of a file in `/etc/hosts` format that contains static IP->hostname mappings. Use this option to resolve hostnames locally without using a DNS. Note that any change to this file triggers a reload in syslog-ng and is instantaneous.

dns_cache_size()

Accepted values:	number
Default:	1007

Description: Number of hostnames in the DNS cache.

time_zone()

Type:	timezone in +/-HH:MM format
Default:	unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

flush_lines()

Accepted values:	number
Default:	0

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *`flush_timeout`* option.

flush_timeout()

Accepted values:	time in milliseconds
Default:	10000

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. See the *`flush_lines()`* option for more information.

group()

Accepted values:	groupid
Default:	root

Description: The default group of output files. By default, syslog-ng changes the privileges of accessed files (for example */dev/null*) to *`root.root 0600`*. To disable modifying privileges, use this option with the *-1* value.

keep_hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting. Enable this option to use hostname-related macros. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available. When relaying messages, enable this option on the syslog-ng server and also on every relay, otherwise syslog-ng will treat incoming messages as if they were sent by the last relay.

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fifo_size()

Accepted values:	number
Default:	1000

Description: The number of lines fitting to the output queue. Note that it is not possible to set this option lower than 1000.

log_msg_size()

Accepted values:	number
Default:	8192

Description: Maximum length of a message in bytes.

normalize_hostnames()

Accepted values:	*yes* | *no*
Default:	*no*

Description: Normalize hostnames, which currently translates to converting them to lower case. (requires 1.9.9)

owner()

Accepted values:	userid
Default:	root

Description: The default owner of output files. By default, syslog-ng changes the privileges of accessed files (for example */dev/null*) to *root.root 0600*. To disable modifying privileges, use this option with the *-1* value.

mark()

Accepted values:	number
Default:	1200

Description: An alias for the obsolete `mark_freq()` option, retained for compatibility with syslog-ng version 1.6.x.

mark_freq()

Accepted values:	number
Default:	1200

Description: The number of seconds between two `MARK` messages. `MARK` messages are generated when there was no message traffic to inform the receiver that the connection is still alive. Note that only local messages postpone the sending of the `MARK` message, relayed messages do not. If set to zero (`0`), no `MARK` messages are sent.

perm()

Accepted values:	permission value
Default:	0600

Description: The default permission for output files. By default, syslog-ng changes the privileges of accessed files (for example `/dev/null`) to `root.root 0600`. To disable modifying privileges, use this option with the `-1` value.

recv_time_zone()

Accepted values:	time offset (for example: `+03:00`)
Default:	local timezone

Description: Specifies the time zone associated with the incoming messages, if not specified otherwise in the message or in the source driver. See also *Section 2.5, Timezone handling (p. 10)* and *Section 5.7, A note on timezones and timestamps (p. 90)* for details.

send_time_zone()

Accepted values:	time offset (for example: `+03:00`)
Default:	local timezone

Description: Specifies the time zone associated with the messages sent by syslog-ng, if not specified otherwise in the message or in the destination driver. See *Section 2.5, Timezone handling (p. 10)* for details.

stats_freq()

Accepted values:	number
Default:	600

Description: The period between two STATS messages in seconds. STATS are log messages sent by syslog-ng, containing statistics about dropped log messages. Set to `0` to disable the STATS messages.

stats_level()

Accepted values:	`0 | 1 | 2 | 3`
Default:	`0`

Description: Specifies the detail of statistics syslog-ng collects about the processed messages.

- Level 0 collects only statistics about the sources and destinations
- Level 1 contains details about the different connections and log files, but has a slight memory overhead
- Level 2 contains detailed statistics based on the hostname.
- Level 3 contains detailed statistics based on various message parameters like facility, severity, or tags.

Note that level 2 and 3 increase the memory requirements and CPU load.

sync() or sync_freq() (DEPRECATED)

Accepted values:	number
Default:	0

Description: Obsolete aliases for `flush_lines()`

time_reap()

Accepted values:	number
Default:	60

Description: The time to wait in seconds before an idle destination file is closed.

time_reopen()

Accepted values:	number
Default:	60

Description: The time to wait in seconds before a dead connection is reestablished.

time_sleep()

Accepted values:	number
Default:	0

Description: The time to wait in milliseconds between each invocation of the `poll()` iteration.

ts_format()

Accepted values:	*rfc3164* | *bsd* | *rfc3339* | *iso*
Default:	*rfc3164*

Description: Specifies the timestamp format used when syslog-ng itself formats a timestamp and nothing else specifies a format (for example: *STAMP* macros, internal messages, messages without original timestamps). See also *Section 5.7, A note on timezones and timestamps (p. 90)*.

use_dns()

Type:	yes, no, persist_only
Default:	yes

Description: Enable or disable DNS usage. The *persist_only* option attempts to resolve hostnames locally from file (for example from /etc/hosts). 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 are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

use_fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

use_time_recvd() (DEPRECATED)

Accepted values:	*yes* | *no*
Default:	*no*

Description: This option controls how the time related macros are expanded in filename and content templates. If set to yes, then the non-prefixed versions of the time related macros (for example: *HOUR* instead of *R_HOUR* and *S_HOUR*) refer to the time when the message was received, otherwise it refers to the timestamp which is in the message.

Note
The timestamps in the messages are generated by the originating host and might not be accurate.

This option is deprecated as many users assumed that it controls the timestamp as it is written to logfiles/destinations, which is not the case. To change how messages are formatted, specify a content-template referring to the appropriate prefixed (*S_* or *R_*) time macro.

6.10. TLS options

The syslog-ng application is able to encrypt incoming and outgoing syslog message flows using SSL/TLS, if the TCP transport protocol (the `tcp()` or `tcp6()` sources or destination) is used.

Note
The format of the TLS connections used by syslog-ng is similar to using syslog-ng and stunnel, but the source IP information is not lost.

To encrypt connections, use the `tls()` option in the source and destination statements.

The tls() option can include the following settings:

ca_dir()

Accepted values:	Directory name
Default:	none

Description: Name of a directory, that contains a set of trusted CA certificates in PEM format. The CA certificate files has to be named after the 32-bit hash of the subject's name. This naming can be created using the c_rehash utility in openssl.

cert_file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate in PEM format, suitable as a TLS certificate, matching the private key.

crl_dir()

Accepted values:	Directory name
Default:	none

Description: Name of a directory that contains the Certificate Revocation Lists for trusted CAs. Similarly to `ca_dir()` files, use the 32-bit hash of the name of the issuing CAs as filenames. The extension of the files must be `.r0`.

key_file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an unencrypted private key in PEM format, suitable as a TLS key.

peer_verify()

Accepted values:	*optional-trusted* | *optional-untrusted* | *required-trusted* | *required-untrusted*
Default:	*required-trusted*

Description: Verification method of the peer, the four possible values is a combination of two properties of validation: whether the peer is required to provide a certificate (required or optional prefix), and whether the certificate provided needs to be trusted or not. For untrusted certificates only the existence of the certificate is checked, but it does not have to be valid — syslog-ng accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatch.

trusted_dn()

Accepted values:	list of accepted distinguished names
Default:	none

Description: To accept connections only from hosts using certain certificates signed by the trusted CAs, list the distinguished names of the accepted certificates in this parameter. For example using *trusted_dn("*, O=Example Inc, ST=Some-State, C=*")* will accept only certificates issued for the *Example Inc* organization in *Some-State* state.

trusted_keys()

Accepted values:	list of accepted SHA-1 fingerprints
Default:	none

Description: To accept connections only from hosts using certain certificates having specific SHA-1 fingerprints, list the fingerprints of the accepted certificates in this parameter. For example *trusted_keys("SHA1:00:EF:ED:A4:CE:00:D1:14:A4:AB:43:00:EF:00:91:85:FF:89:28:8F", "SHA1:0C:42:00:3E:B2:60:36:64:00:E2:83:F0:80:46:AD:00:A8:9D:00:15")*.

Note

When using the *trusted_keys()* and *trusted_dn()* parameters, note the following:

- First, the *trusted_keys()* parameter is checked. If the fingerprint of the peer is listed, the certificate validation is performed.
- If the fingerprint of the peer is not listed in the *trusted_keys()* parameter, the *trusted_dn()* parameter is checked. If the DN of the peer is not listed in the *trusted_dn()* parameter, the authentication of the peer fails and the connection is closed.

Appendix 1. The syslog-ng manual pages

Name

syslog-ng — syslog-ng system logger application

Synopsis

`syslog-ng` [options]

Description

NOTE: This manual page covers both editions of syslog-ng: syslog-ng Open Source Edition and the commercial syslog-ng Premium Edition. Features that are only included in the Premium Edition are marked with an asterisk (*). For details, see the official syslog-ng website: *http://www.balabit.com/network-security/syslog-ng/*.

This manual page is only an abstract; for the complete documentation of syslog-ng, see *The syslog-ng Administrator Guide*.

The syslog-ng application is a flexible and highly scalable system logging application. Typically, syslog-ng is used to manage log messages and implement centralized logging, where the aim is to collect the log messages of several devices on a single, central log server. The different devices - called syslog-ng clients - all run syslog-ng, and collect the log messages from the various applications, files, and other *sources*. The clients send all important log messages to the remote syslog-ng server, where the server sorts and stores them.

Options

`--cfgfile <file>` or `-f <file>`	Use the specified configuration file.
`--chroot <dir>` or `-C <dir>`	Change root to the specified directory after reading the configuration file. The directory must be set up accordingly. Note that it is not possible to reload the syslog-ng configuration after chrooting.
`--debug` or `-d`	Start syslog-ng in debug mode.
`--enable-core`	Enable syslog-ng to write core files in case of a crash to help support and debugging.
`--fd-limit`	Set the minimal number of required file descriptors (fd-s); this sets how many files syslog-ng can keep open simultaneously. Default value: *4096*. Note that this does not override the global ulimit setting of the host.
`--foreground` or `-F`	Do not daemonize, run in the foreground.
`--group <group>` or `-g <group>`	Switch to the specified group after initializing the configuration file.
`--help` or `-h`	Display a brief help message.
`--no-caps`	Run syslog-ng as root, without capability-support. This is the default behavior. On Linux, it is possible to run syslog-ng as non-root with capability-support if syslog-ng was compiled with the *--enable-linux-caps* option enabled. (Execute `syslog-ng --version` to display the list of enabled build parameters.)

`--persist-file <persist-file>` or `-R <persist-file>`	Set the path and name of the `syslog-ng.persist` file where the persistent options and data are stored.
`--pidfile <pidfile>` or `-p <pidfile>`	Set path to the PID file where the pid of the main process is stored.
`--process-mode <pidfile>`	Sets how to run syslog-ng: in the *`foreground`* (mainly used for debugging), in the *`background`* as a daemon, or in *`safe-background`* mode. By default, syslog-ng runs in *`safe-background`* mode. This mode creates a supervisor process called *`supervising syslog-ng`* , that restarts syslog-ng if it crashes.
`--qdisk-dir <path>` or `-Q <path>`	Specify the location of the file used for disk-based buffering. By default, this file is located at `/var/lib/syslog-ng/`.
`--stderr` or `-e`	Log internal messages of syslog-ng to stderr. Mainly used for debugging purposes in conjunction with the *`--foreground`* option. If not specified, syslog-ng will log such messages to its internal source.
`--syntax-only` or `-s`	Verify that the configuration file is syntactically correct and exit.
`--user <user>` or `-u <user>`	Switch to the specified user after initializing the configuration file (and optionally chrooting). Note that it is not possible to reload the syslog-ng configuration if the specified user has no privilege to create the `/dev/log` file.
`--verbose` or `-v`	Enable verbose logging used to troubleshoot syslog-ng.
`--version` or `-V`	Display version number and compilation information.

Files

`/opt/syslog-ng/etc/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng/syslog-ng.conf`

See also

`syslog-ng.conf(5)`

`The syslog-ng Administrator Guide`

If you experience any problems or need help with syslog-ng, visit the *`syslog-ng mailing list`*

For news and notifications about the documentation of syslog-ng, visit the *`BalaBit Documentation Blog`*.

Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

Copyright

Name

syslog-ng.conf — syslog-ng configuration file

Synopsis

`syslog-ng.conf`

Description

NOTE: This manual page covers both editions of syslog-ng: syslog-ng Open Source Edition and the commercial syslog-ng Premium Edition. Features that are only included in the Premium Edition are marked with an asterisk (*). For details, see the official syslog-ng website: *http://www.balabit.com/network-security/syslog-ng/*.

This manual page is only an abstract; for the complete documentation of syslog-ng, see *`The syslog-ng Administrator Guide`*.

The syslog-ng application is a flexible and highly scalable system logging application. Typically, syslog-ng is used to manage log messages and implement centralized logging, where the aim is to collect the log messages of several devices on a single, central log server. The different devices - called syslog-ng clients - all run syslog-ng, and collect the log messages from the various applications, files, and other *sources*. The clients send all important log messages to the remote syslog-ng server, where the server sorts and stores them.

The syslog-ng application reads incoming messages and forwards them to the selected *destinations*. The syslog-ng application can receive messages from files, remote hosts, and other *sources*.

Log messages enter syslog-ng in one of the defined sources, and are sent to one or more *destinations*.

Sources and destinations are independent objects; *log paths* define what syslog-ng does with a message, connecting the sources to the destinations. A log path consists of one or more sources and one or more destinations; messages arriving to a source are sent to every destination listed in the log path. A log path defined in syslog-ng is called a *log statement*.

Optionally, log paths can include *filters*. Filters are rules that select only certain messages, for example, selecting only messages sent by a specific application. If a log path includes filters, syslog-ng sends only the messages satisfying the filter rules to the destinations set in the log path.

Configuring syslog-ng

Global objects (for example sources, destinations, log paths, or filters) are defined in the syslog-ng configuration file. Object definitions consist of the following elements:

- *Type of the object*: One of `source`, `destination`, `log`, `filter`, `parser`, `rewrite` rule, or `template`.
- *Identifier of the object*: A unique name identifying the object. When using a reserved word as an identifier, enclose the identifier in quotation marks.

Tip
Use identifiers that refer to the type of the object they identify. For example, prefix source objects with `s_`, destinations with `d_`, and so on.

- *Parameters*: The parameters of the object, enclosed in braces `{parameters}`.
- *Semicolon*: Object definitions end with a semicolon (`;`).

The syntax is summarized as follows:

The syntax of log statements is as follows:

```
log {
    source(s1); source(s2); ...
    optional_element(filter1|parser1|rewrite1);
optional_element(filter2|parser2|rewrite2);...
    destination(d1); destination(d2); ...
    flags(flag1[, flag2...]);
    };
```

The following log statement sends all messages arriving to the localhost to a remote server.

```
source s_localhost { tcp(ip(127.0.0.1) port(1999) ); };
destination d_tcp { tcp("10.1.2.3" port(1999); localport(999)); };
log { source(s_localhost); destination(d_tcp); };
```

The syslog-ng application has a number of global options governing DNS usage, the timestamp format used, and other general points. Each option may have parameters, similarly to driver specifications. To set global options, add an option statement to the syslog-ng configuration file using the following syntax:

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

The sources, destinations, and filters available in syslog-ng are listed below. For details, see *The syslog-ng Administrator Guide*.

Name	Description
internal()	Messages generated internally in syslog-ng.
file()	Opens the specified file and reads messages.
pipe(), fifo	Opens the specified named pipe and reads messages.
program()	Opens the specified application and reads messages from its standard output.
sun-stream(), sun-streams()	Opens the specified `STREAMS` device on Solaris systems and reads incoming messages.
syslog()	Listens for incoming messages using the new *IETF-standard syslog protocol*.
tcp(), tcp6()	Listens on the specified TCP port for incoming messages using the *BSD-syslog protocol* over IPv4 and IPv6 networks, respectively.
udp(), udp6()	Listens on the specified UDP port for incoming messages using the *BSD-syslog protocol* over IPv4 and IPv6 networks, respectively.

Name	Description
unix-dgram()	Opens the specified unix socket in `SOCK_DGRAM` mode and listens for incoming messages.
unix-stream()	Opens the specified unix socket in `SOCK_STREAM` mode and listens for incoming messages.

Table 1.1. Source drivers available in syslog-ng

Name	Description
file()	Writes messages to the specified file.
fifo(), pipe()	Writes messages to the specified named pipe.
program()	Forks and launches the specified program, and sends messages to its standard input.
sql()	Sends messages into an SQL database. In addition to the standard syslog-ng packages, the `sql()` destination requires database-specific packages to be installed. Refer to the section appropriate for your platform in *Chapter 3, Installing syslog-ng (p. 26)*.
syslog()	Sends messages to the specified remote host using the *IETF-syslog protocol*. The IETF standard supports message transport using the UDP, TCP, and TLS networking protocols.
tcp() and tcp6()	Sends messages to the specified TCP port of a remote host using the *BSD-syslog protocol* over IPv4 and IPv6, respectively.
udp() and udp6()	Sends messages to the specified UDP port of a remote host using the *BSD-syslog protocol* over IPv4 and IPv6, respectively.
unix-dgram()	Sends messages to the specified unix socket in `SOCK_DGRAM` style (BSD).
unix-stream()	Sends messages to the specified unix socket in `SOCK_STREAM` style (Linux).
usertty()	Sends messages to the terminal of the specified user, if the user is logged in.

Table 1.2. Destination drivers available in syslog-ng

Files

`/opt/syslog-ng/etc/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng/syslog-ng.conf`

See also

`syslog-ng(8)`

`The syslog-ng Administrator Guide`

If you experience any problems or need help with syslog-ng, visit the *`syslog-ng mailing list`*

For news and notifications about the documentation of syslog-ng, visit the *`BalaBit Documentation Blog`*.

Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

Copyright

Name

pdbtool — An application to test and convert syslog-ng pattern database rules

Synopsis

`pdbtool` [command] [options]

Description

This manual page is only an abstract; for the complete documentation of syslog-ng and pdbtool, see *The syslog-ng Administrator Guide*.

The syslog-ng application can match the contents of the log messages to a database of predefined message patterns (also called patterndb). By comparing the messages to the known patterns, syslog-ng is able to identify the exact type of the messages, tag the messages, and sort them into message classes. The message classes can be used to classify the type of the event described in the log message. The functionality of the pattern database is similar to that of the logcheck project, but the syslog-ng approach is faster, scales better, and is much easier to maintain compared to the regular expressions of logcheck.

The `pdbtool` application is a utility that can be used to:

- test message patterns;
- convert an older pattern database to the latest database format;
- merge pattern databases into a single file;
- dump the RADIX tree built from the pattern database (or a part of it) to explore how the pattern matching works.

The match command

`match` [options]

Use the `match` command to test the rules in a pattern database. The command tries to match the specified message against the patterns of the database, evaluates the parsers of the pattern, and also displays which part of the message was parsed successfully. The command returns with a *0* (success) or *1* (no match) return code and displays the following information:

- the class assigned to the message (e.g., system, violation, etc.),
- the ID of the rule that matched the message, and
- the values of the parsers (if there were parsers in the matching pattern).

The `match` command has the following options:

`--color-out` or `-c`	Color the terminal output to highlight the part of the message that was successfully parsed.
`--debug-pattern` or `-D`	Print debugging information about the pattern matching.

`--message` or `-M`	The text of the log message to match (only the *$MESSAGE* part without the syslog headers).
`--pdb` or `-p`	Name of the pattern database file to use.
`--program` or `-P`	Name of the program to use, as contained in the *$PROGRAM* part of the syslog message.

Example:

```
pdbtool match -p patterndb.xml -P sshd -M "Accepted publickey for myuser from
127.0.0.1 port 59357 ssh2"
```

The merge command

`merge` [options]

Use the `merge` command to combine separate pattern database files into a single file (pattern databases are usually stored in separate files per applications to simplify maintenance). If a file uses an older database format, it is automatically updated to the latest format (V3). See the *The syslog-ng Administrator Guide* for details on the different pattern database versions.

`--directory` or `-D`	The directory that contains the pattern database XML files to be merged.
`--pdb` or `-p`	Name of the output pattern database file.

Example:

```
pdbtool merge --directory /home/me/mypatterns/  --pdb
/var/lib/syslog-ng/patterndb.xml
```

Currently it is not possible to convert a file without merging, so if you only want to convert an older pattern database file to the latest format, you have to copy it into an empty directory.

The dump command

`dump` [options]

Display the RADIX tree built from the patterns. This shows how are the patterns represented in syslog-ng and it might also help to track down pattern-matching problems. The dump utility can dump the tree used for matching the PROGRAM or the MSG parts.

`--pdb` or `-p`	Name of the pattern database file to use.
`--program` or `-P`	Displays the RADIX tree built from the patterns belonging to the *$PROGRAM* application.
`--program-tree` or `-T`	Display the *$PROGRAM* tree.

Example and sample output:

```
pdbtool dump -p patterndb.xml  -P 'sshd'
```

```
'p'
   'assword for'
     @QSTRING:@
```

```
      'from'
       @QSTRING:@
         'port '
          @NUMBER:@ rule_id='fc49054e-75fd-11dd-9bba-001e6806451b'
            ' ssh' rule_id='fc55cf86-75fd-11dd-9bba-001e6806451b'
               '2' rule_id='fc4b7982-75fd-11dd-9bba-001e6806451b'
   'ublickey for'
     @QSTRING:@
       'from'
        @QSTRING:@
          'port '
           @NUMBER:@ rule_id='fc4d377c-75fd-11dd-9bba-001e6806451b'
             ' ssh' rule_id='fc5441ac-75fd-11dd-9bba-001e6806451b'
                '2' rule_id='fc44a9fe-75fd-11dd-9bba-001e6806451b'
```

Files

`/opt/syslog-ng/bin/pdbtool`

`/opt/syslog-ng/etc/syslog-ng/syslog-ng.conf`

See also

The syslog-ng Administrator Guide

syslog-ng.conf(5)

syslog-ng(8)

If you experience any problems or need help with syslog-ng, visit the *syslog-ng mailing list*

For news and notifications about the documentation of syslog-ng, visit the *BalaBit Documentation Blog*.

Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

Copyright

Name

loggen — Generate syslog messages at a specified rate

Synopsis

`loggen` [options]`target` [port]

Description

NOTE: The loggen application is distributed with the syslog-ng system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at the official syslog-ng website: *http://www.balabit.com/network-security/syslog-ng/*.

This manual page is only an abstract; for the complete documentation of syslog-ng, see *The syslog-ng Administrator Guide*.

The loggen application is tool to test and stress-test your syslog server and the connection to the server. It can send syslog messages to the server at a specified rate, using a number of connection types and protocols.

Options

`--csv` or `-C`	Send statistics of the sent messages to stdout as CSV. This can be used for plotting the message rate.
`--dgram` or `-D`	Use datagram socket (UDP or unix-dgram) to send the messages to the target.
`--help` or `-h`	Display a brief help message.
`--inet` or `-i`	Use the TCP (by default) or UDP (when used together with the --dgram option) protocol to send the messages to the target.
`--interval <seconds>` or `-I <seconds>`	The number of seconds loggen will run. Default value: 10
`--no-framing` or `-F`	Do not use the framing of the IETF-syslog protocol style, even if the *`syslog-proto`* option is set.
`--rate <message/second>` or `-r <message/second>`	The number of messages generated per second. Default value: 1000
`--read-file` or `-R`	Read the messages from a file and send them to the target. See also the *`--skip-tokens`* option.
`--size` or `-s`	The size of a syslog message in bytes. Default value: 256
`--skip-tokens`	Skip the specified number of space-separated tokens (words) at the beginning of every line. For example, if the messages in the file look like *`foo bar message`*, *`--skip-tokens 2`* skips the *`foo bar`* part of the line, and sends only the *`message`* part. Works only when used together with the *`--read-file`* parameter.
`--stream` or `-S`	Use a stream socket (TCP or unix-stream) to send the messages to the target.

`--syslog-proto` or `-P`	Use the new IETF-syslog message format as specified in RFC5424. By default, loggen uses the legacy BSD-syslog message format (as described in RFC3164). See also the *`--no-framing`* option.
`--unix` or `-x`	Use a UNIX domain socket to send the messages to the target.
`--use-ssl` or `-U`	Use an SSL-encrypted channel to send the messages to the target. Note that it is not possible to check the certificate of the target, or to perform mutual authentication.
`--verbose` or `-V`	Display the actual speed of sending messages in messages/second.

Example

The following command generates 100 messages per second for ten minutes, and sends them to port 2010 of the localhost via TCP. Each message is 300 bytes long.

```
loggen --size 300 --rate 100 --interval 600 127.0.0.1 2010
```

The following command is similar to the one above, but uses the UDP protocol.

```
loggen --inet --dgram --size 300 --rate 100 --interval 600 127.0.0.1 2010
```

Files

```
/opt/syslog-ng/bin/loggen
```

See also

syslog-ng.conf(5)

The syslog-ng Administrator Guide

If you experience any problems or need help with loggen or syslog-ng, visit the *syslog-ng mailing list*

Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

Copyright

Name

syslog-ng-ctl — Display message statistics and enable verbose, debug and trace modes in syslog-ng Open Source Edition

Synopsis

`syslog-ng-ctl` [command] [options]

Description

NOTE: The syslog-ng-ctl application is distributed with the syslog-ng Open Source Edition system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at the official syslog-ng website: *http://www.balabit.com/network-security/syslog-ng/*.

This manual page is only an abstract; for the complete documentation of syslog-ng, see *The syslog-ng Open Source Edition Administrator Guide*.

The `syslog-ng-ctl` application is a utility that can be used to:

- enable/disable various syslog-ng messages for troubleshooting;
- display statistics about the processed messages.

Enabling troubleshooting messages

`command` [options]

Use the `syslog-ng-ctl <command> --set=on` command to display verbose, trace, or debug messages. If you are trying to solve configuration problems, the debug (and occassionally trace) messages are usually sufficient; debug messages are needed mostly for finding software errors. After solving the problem, do not forget to turn these messages off using the `syslog-ng-ctl <command> --set=off`. Note that enabling debug messages does not enable verbose and trace messages.

Use `syslog-ng-ctl <command>` without any parameters to display whether the particular type of messages are enabled or not.

If you need to use a non-standard control socket to access syslog-ng, use the `syslog-ng-ctl <command> --set=on --control=<socket>` command to specify the socket to use.

`verbose` Print verbose messages. If syslog-ng was started with the *`--stderr`* or `-e` option, the messages will be sent to stderr. If not specified, syslog-ng will log such messages to its internal source.

`trace` Print trace messages of how messages are processed. If syslog-ng was started with the *`--stderr`* or `-e` option, the messages will be sent to stderr. If not specified, syslog-ng will log such messages to its internal source.

`debug` Print debug messages. If syslog-ng was started with the *`--stderr`* or `-e` option, the messages will be sent to stderr. If not specified, syslog-ng will log such messages to its internal source.

Example:

```
syslog-ng-ctl verbose --set=on
```

The stats command

`stats` [options]

Use the `validate` command to validate the signatures and timestamps of a logstore file. The `validate` command has the following options:

`--control=<socket>` or `-c` — Specify the socket to use to access syslog-ng. Only needed when using a non-standard socket.

Example:

```
syslog-ng-ctl stats
```

An example output:

```
src.internal;s_all#0;;a;processed;6445
src.internal;s_all#0;;a;stamp;1268989330
destination;df_auth;;a;processed;404
destination;df_news_dot_notice;;a;processed;0
destination;df_news_dot_err;;a;processed;0
destination;d_ssb;;a;processed;7128
destination;df_uucp;;a;processed;0
source;s_all;;a;processed;7128
destination;df_mail;;a;processed;0
destination;df_user;;a;processed;1
destination;df_daemon;;a;processed;1
destination;df_debug;;a;processed;15
destination;df_messages;;a;processed;54
destination;dp_xconsole;;a;processed;671
dst.tcp;d_network#0;10.50.0.111:514;a;dropped;5080
dst.tcp;d_network#0;10.50.0.111:514;a;processed;7128
dst.tcp;d_network#0;10.50.0.111:514;a;stored;2048
destination;df_syslog;;a;processed;6724
destination;df_facility_dot_warn;;a;processed;0
destination;df_news_dot_crit;;a;processed;0
destination;df_lpr;;a;processed;0
destination;du_all;;a;processed;0
destination;df_facility_dot_info;;a;processed;0
center;;received;a;processed;0
destination;df_kern;;a;processed;70
center;;queued;a;processed;0
destination;df_facility_dot_err;;a;processed;0
```

Files

`/opt/syslog-ng/sbin/syslog-ng-ctl`

See also

The syslog-ng Administrator Guide

syslog-ng.conf(5)

syslog-ng(8)

If you experience any problems or need help with syslog-ng, visit the *syslog-ng mailing list*

For news and notifications about the documentation of syslog-ng, visit the *BalaBit Documentation Blog*.

Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

Copyright

Appendix 2. GNU General Public License

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor,
Boston, MA 02110-1301
USA

Version 2, June 1991

2.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

2.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

2.2.1. Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2.2.2. Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2.2.3. Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of *Section 1* above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: If the Program itself is interactive but does not

normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

2.2.4. Section 3

You may copy and distribute the Program (or a work based on it, under *Section 2* in object code or executable form under the terms of *Sections 1* and *2* above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

2.2.5. Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

2.2.6. Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

2.2.7. Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

2.2.8. Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

2.2.9. Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

2.2.10. Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

2.2.11. Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

2.2.12. NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

2.2.13. Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

2.3. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type "show w". This is free software, and you are welcome to redistribute it under certain conditions; type "show c" for details.

The hypothetical commands "show w" and "show c" should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than "show w" and "show c"; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program "Gnomovision" (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix 3. Deprecated pattern database schemes

This appendix describes the older versions of the syslog-ng pattern database. These descriptions are only for reference, if you want to create a pattern database, you are recommended to use the latest format: V3, which is supported by syslog-ng version 3.1 and later, and the syslog-ng Store Box version 1.1 and later.

Note
Note that V3 is backwards compatible with the V2 format, meaning that applications that support V3 can use pattern databases in the V2 format as well (but not V1). To convert your existing pattern databases to the V3 format, use the `pdbtool` application bundled with syslog-ng 3.1 and later. See the manual page of `pdbtool` for details.

3.1. The syslog-ng pattern database format V1

The XML schema of the V1 pattern database used in syslog-ng OSE and PE 3.0.X is described below. Newer versions syslog-ng 3.1 and later use an updated V3 format that is described in *Section 6.6.2.3, Creating pattern databases (p. 163)*.

Note
Note that V3 is backwards compatible with the V2 format, meaning that applications that support V3 can use pattern databases in the V2 format as well (but not V1). To convert your existing pattern databases to the V3 format, use the `pdbtool` application bundled with syslog-ng 3.1 and later. See the manual page of `pdbtool` for details.

- **<patterndb>**: The container element of the pattern database. For example:

```
<patterndb version='1' pub_date='2008-08-25'>
```

- *version*: The schema version of the pattern database. The current version is *2*.
- *pubdate*: The publication date of the XML file.
- **<program>**: A container element to group log patterns for an application or program. For example:

```
<program name='su' id='480de478-d4a6-4a7f-bea4-0c0245d361e1'>
```

<patterndb> element may contain any number of **<program>** elements.

  - *name*: The name of the application. Note that the function of this attribute is to make the database more readable, syslog-ng uses the *<pattern>* element to identify the applications sending log messages.
  - *id*: A unique ID of the application, for example, the md5 sum of the *name* attribute.
  - **pattern**: The name of the application — syslog-ng matches this value to the $PROGRAM header of the syslog message to find the rulesets applicable to the syslog message. This element is also called *program pattern*. E.g.,

```
<pattern>su</pattern>
```

- **description**: OPTIONAL — A description of the ruleset or the application.
- **url**: OPTIONAL — An URL referring to further information about the ruleset or the application.
- **<rules>**: A container element for the rules of the ruleset.
  - **<rule>**: An element containing message patterns and how a message that matches these patterns is classified. For example:

```
<rule provider='balabit' id='f57196aa-75fd-11dd-9bba-001e6806451b'
class='violation'>
```

Note
If the following characters appear in the message, they must be escaped in the rule as follows:

- *@*: Use @@, e.g., *user@@example.com*
- <: Use *<*
- >: Use *>*
- &: Use *&*

The **<rules>** element may contain any number of **<rule>** elements.

- *provider*: The provider of the rule. This is used to distinguish between who supplied the rule; i.e., if it has been created by BalaBit, or added to the xml by a local user.
- *id*: The globally unique ID of the rule.
- *class*: The class of the rule — syslog-ng assigns this class to the messages matching a pattern of this rule.
- **<pattern>**: A pattern describing a log message. This element is also called *message pattern*. For example:

```
<pattern>+ ??? root-</pattern>
```

Example 3.1. A V1 pattern database containing a single rule
The following pattern database contains a single rule that matches log messages of the *PF* packet-filtering application. A sample log message looks like:

```
PF: DROP filter/INPUT IN=eth0 OUT= MAC=00:1A:4B:80:90:C9:00:1A:4B:80:90:C6 SRC=192.168.155.11
 DST=192.168.155.1 LEN=60 TOS=0x10 PREC=0x00 TTL=64 ID=51939 DF PROTO=TCP SPT=34407 DPT=80
 WINDOW=32792 RES=0x00 SYN URGP=0
```

The following is a simple pattern database containing a matching rule.

```
<patterndb version='1' pub_date='2009-04-17'>
    <program name='PF'>
        <pattern>PF</pattern>
            <rule id='1' class='pf'>
                <pattern>@STRING:PF.VERDICT@ @STRING:PF.CHAIN:/@ IN=@STRING:PF.IN_IFACE@
OUT= MAC=@STRING:PF.MAC::@ SRC=@IPV4:PF.SRC_IP@ DST=@IPV4:PF.DST_IP@ LEN=@NUMBER:PF.PKT_LEN@
 TOS=@STRING:PF.TOS@ PREC=@STRING:PF.PREC@ TTL=@NUMBER:PF.TTL@ ID=@NUMBER:PF.ID@ DF
PROTO=@STRING:PF.PROTO@ SPT=@NUMBER:PF.SRC_PORT@ DPT=@NUMBER:PF.DST_PORT@
WINDOW=@NUMBER:PF.TCP_WINDOW@ RES=@STRING:PF.RES@ SYN URGP=@NUMBER:PF.TCP_URGP@</pattern>

            </rule>
    </program>
</patterndb>
```

Note that the rule uses macros that refer to parts of the message, for example, you can use the `$PF.DST_IP` macro refer to the destination IP address of the logged connection+.

3.2. The syslog-ng pattern database format V2

The XML schema of the V2 pattern database is described below. Newer versions syslog-ng and the syslog-ng Store Box use an updated V3 format that is described in *Section 6.6.2.3, Creating pattern databases (p. 163)*.

Note
Note that V3 is backwards compatible with the V2 format, meaning that applications that support V3 can use pattern databases in the V2 format as well (but not V1). To convert your existing pattern databases to the V3 format, use the `pdbtool` application bundled with syslog-ng 3.1 and later. See the manual page of `pdbtool` for details.

The following scheme describes the V2 format of the pattern database. This format is used by the syslog-ng Store Box (SSB) appliance version 1.0.x (see *http://www.balabit.com/network-security/syslog-ng/log-server-appliance/* for details).

For a sample database containing only a single pattern, see *Example 3.2, A V2 pattern database containing a single rule (p. 205)*.

- **<patterndb>**: The container element of the pattern database. For example:

```
<patterndb version='2' pub_date='2008-08-25'>
```

- *version*: The schema version of the pattern database. The current version is *2*.
- *pubdate*: The publication date of the XML file.
- **<ruleset>**: A container element to group log patterns for an application or program. For example:

```
<ruleset name='su' id='480de478-d4a6-4a7f-bea4-0c0245d361e1'>
```

A *<patterndb>* element may contain any number of **<ruleset>** elements.

  - *name*: The name of the application. Note that the function of this attribute is to make the database more readable, syslog-ng uses the *<pattern>* element to identify the applications sending log messages.
  - *id*: A unique ID of the application, for example, the md5 sum of the *name* attribute.
  - **description**: OPTIONAL — A description of the ruleset or the application.
  - **url**: OPTIONAL — An URL referring to further information about the ruleset or the application.
  - **pattern**: The name of the application — syslog-ng matches this value to the $PROGRAM header of the syslog message to find the rulesets applicable to the syslog message. This element is also called *program pattern*. E.g.,

```
<pattern>su</pattern>
```

Note
If the *<pattern>* element of a ruleset is not specified, -ng will use this ruleset as a fallback ruleset: it will apply the ruleset to messages that have an empty PROGRAM header, or if none of the program patterns matched the PROGRAM header of the incoming message.

- **<rules>**: A container element for the rules of the ruleset.
  - **<rule>**: An element containing message patterns and how a message that matches these patterns is classified. For example:

```
<rule provider='balabit'

id='f57196aa-75fd-11dd-9bba-001e6806451b'
                                                class='violation'>
```

Note
If the following characters appear in the message, they must be escaped in the rule as follows:
- *@*: Use @@, e.g., *user@@example.com*
- <: Use *<*
- >: Use *>*
- &: Use *&*

The **<rules>** element may contain any number of **<rule>** elements.

- *provider*: The provider of the rule. This is used to distinguish between who supplied the rule; i.e., if it has been created by BalaBit, or added to the xml by a local user.
- *id*: The globally unique ID of the rule.
- *class*: The class of the rule — syslog-ng assigns this class to the messages matching a pattern of this rule.
- **<patterns>**: An element containing the patterns of the rule. If a **<patterns>** element contains multiple **<pattern>** elements, the class of the **<rule>** is assigned to every syslog message matching any of the patterns.
  - **<pattern>**: A pattern describing a log message. This element is also called *message pattern*. For example:

```
<pattern>+ ??? root-</pattern>
```

  - **description**: OPTIONAL — A description of the pattern or the log message matching the pattern.
  - **urls**: OPTIONAL — An element containing one or more URLs referring to further information about the patterns or the matching log messages.
    - **url**: OPTIONAL — An URL referring to further information about the patterns or the matching log messages.
  - **tags**: OPTIONAL — An element containing custom keywords (tags) about the rules. The tags can be used to label specific events (e.g., user logons).

- **tag**: OPTIONAL — A keyword or tags applied to messages matching the rule. For example:

```
<tags><tag>UserLogin</tag></tags>
```

Example 3.2. A V2 pattern database containing a single rule

The following pattern database contains a single rule that matches a log message of the *ssh* application. A sample log message looks like:

```
Accepted password for sampleuser from 10.50.0.247 port 42156 ssh2
```

The following is a simple pattern database containing a matching rule.

```
<patterndb version='2' pub_date='2009-04-17'>
    <ruleset name='ssh' id='123456678'>
        <pattern>ssh</pattern>
            <rules>
                <rule provider='me' id='182437592347598' class='system'>
                    <patterns>
                        <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @
for@QSTRING:SSH_USERNAME: @from\ @QSTRING:SSH_CLIENT_ADDRESS: @port @NUMBER:SSH_PORT_NUMBER:@
 ssh2</pattern>
                    </patterns>
                </rule>
            </rules>
    </ruleset>
</patterndb>
```

Note that the rule uses macros that refer to parts of the message, for example, you can use the *$SSH_USERNAME* macro refer to the username used in the connection.

Appendix 4. Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED. BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. *Definitions*
   a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
   b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.
   c. "Distribute" means to make available to the public the original and copies of the Work through sale or other transfer of ownership.
   d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
   e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
   f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing;

a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. *Fair Dealing Rights.* Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. *License Grant.* Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,

   b. to Distribute and Publicly Perform the Work including as incorporated in Collections.

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

4. *Restrictions.* The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every

copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.

b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

   i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

   ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

   iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this Li-

cense that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

5. *Representations, Warranties and Disclaimer* UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. *Limitation on Liability.* EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. *Termination*

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. *Miscellaneous*

   a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Glossary

alias IP	An additional IP address assigned to an interface that already has an IP address. The normal and alias IP addresses both refer to the same physical interface.
authentication	The process of verifying the authenticity of a user or client before allowing access to a network system or service.
auditing policy	The auditing policy determines which events are logged on host running Microsoft Windows operating systems.
BSD-syslog protocol	The old syslog protocol standard described in RFC 3164 *http://www.ietf.org/rfc/rfc3164.txt*. Sometimes also referred to as the legacy-syslog protocol.
CA	A Certificate Authority (CA) is an institute that issues certificates.
certificate	A certificate is a file that uniquely identifies its owner. Certificates contains information identifying the owner of the certificate, a public key itself, the expiration date of the certificate, the name of the CA that signed the certificate, and some other data.
client mode	In client mode, syslog-ng collects the local logs generated by the host and forwards them through a network connection to the central syslog-ng server or to a relay.
destination	A named collection of configured destination drivers.
destination driver	A communication method used to send log messages.
destination, network	A destination that sends log messages to a remote host (i.e., a syslog-ng relay or server) using a network connection.
destination, local	A destination that transfers log messages within the host, e.g., writes them to a file, or passes them to a log analyzing application.
disk buffer	The Premium Edition of syslog-ng can store messages on the local hard disk if the central log server or the network connection to the server becomes unavailable.
disk queue	See *disk buffer*.
domain name	The name of a network, e.g.: `balabit.com`.
embedded log statement	A log statement that is included in another log statement to create a complex log path.
filter	An expression to select messages.

gateway	A device that connect two or more parts of the network, e.g.: your local intranet and the external network (the Internet). Gateways act as entrances into other networks.
high availability	High availability uses a second syslog-ng server unit to ensure that the logs are received even if the first unit breaks down.
host	A computer connected to the network.
hostname	A name that identifies a host on the network.
IETF-syslog protocol	The syslog-protocol standard developed by the Internet Engineering Task Force (IETF), described in RFC 5424-5428 *http://www.ietf.org/internet-drafts/draft-ietf-syslog-protocol-23.txt*.
key pair	A private key and its related public key. The private key is known only to the owner; the public key can be freely distributed. Information encrypted with the private key can only be decrypted using the public key.
license	The syslog-ng license determines the number of distinct hosts (clients and relays) that can connect to the syslog-ng server.
log path	A combination of sources, filters, parsers, rewrite rules, and destinations: syslog-ng examines all messages arriving to the sources of the logpath and sends the messages matching all filters to the defined destinations.
logstore	A binary logfile format that can encrypt, compress, and timestamp log messages.
LSH	See *log source host*.
log source host	A host or network device (including syslog-ng clients and relays) that sends logs to the syslog-ng server. Log source hosts can be servers, routers, desktop computers, or other devices capable of sending syslog messages or running syslog-ng.
log statement	See *log path*.
name server	A network computer storing the IP addresses corresponding to domain names.
Oracle Instant Client	The Oracle Instant Client is a small set of libraries, which allow you to connect to an Oracle Database. A subset of the full Oracle Client, it requires minimal installation but has full functionality.
output buffer	A part of the memory of the host where syslog-ng stores outgoing log messages if the destination cannot accept the messages immediately.
output queue	Messages from the output queue are sent to the target syslog-ng server. The syslog-ng application puts the outgoing messages directly into the output queue, unless the output queue is full. The output queue can hold 64 messages, this is a fixed value and cannot be modified.

overflow queue	See *output buffer*.
parser	A set of rules to segment messages into named fields or columns.
ping	A command that sends a message from a host to another host over a network to test connectivity and packet loss.
port	A number ranging from 1 to 65535 that identifies the destination application of the transmitted data. E.g.: SSH commonly uses port 22, web servers (HTTP) use port 80, etc.
Public-key authentication	An authentication method that uses encryption key pairs to verify the identity of a user or a client.
regular expression	A regular expression is a string that describes or matches a set of strings. The syslog-ng application supports extended regular expressions (also called POSIX modern regular expressions).
relay mode	In relay mode, syslog-ng receives logs through the network from syslog-ng clients and forwards them to the central syslog-ng server using a network connection.
rewrite rule	A set of rules to modify selected elements of a log message.
template	A user-defined structure that can be used to restructure log messages or automatically generate file names.
server mode	In server mode, syslog-ng acts as a central log-collecting server. It receives messages from syslog-ng clients and relays over the network, and stores them locally in files, or passes them to other applications, e.g., log analyzers.
source	A named collection of configured source drivers.
source, network	A source that receives log messages from a remote host using a network connection. The following sources are network sources: `tcp()`, `tcp6()`, `udp()`, `udp6()`.
source, local	A source that receives log messages from within the host, e.g., from a file.
source driver	A communication method used to receive log messages.
SSL	See *TLS*.
syslog-ng	The syslog-ng application is a flexible and highly scalable system logging application, typically used to manage log messages and implement centralized logging.
syslog-ng agent	The syslog-ng agent for Windows is a log collector and forwarder application for the Microsoft Windows platform. It collects the log messages of the Windows-based host and forwards them to a syslog-ng server using regular or SSL-encrypted TCP connections.
syslog-ng client	A host running syslog-ng in client mode.

syslog-ng Premium Edition	The syslog-ng Premium Edition is the commercial version of the open-source application. It offers additional features, like encrypted message transfer and an agent for Microsoft Windows platforms.
syslog-ng relay	A host running syslog-ng in relay mode.
syslog-ng server	A host running syslog-ng in server mode.
TLS	Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols which provide secure communications on the Internet. The syslog-ng application can encrypt the communication between the clients and the server using TLS to prevent unauthorized access to sensitive log messages.
traceroute	A command that shows all routing steps (the path of a message) between two hosts.
unix domain socket	A Unix domain socket (UDS) or IPC socket (inter-procedure call socket) is a virtual socket, used for inter-process communication.

List of syslog-ng OSE parameters

Symbols

B

C

D

E

F

G

H

I

K

L

M

N

O

Index

D

E

F

G

H

I

K

L

M

N

O

P